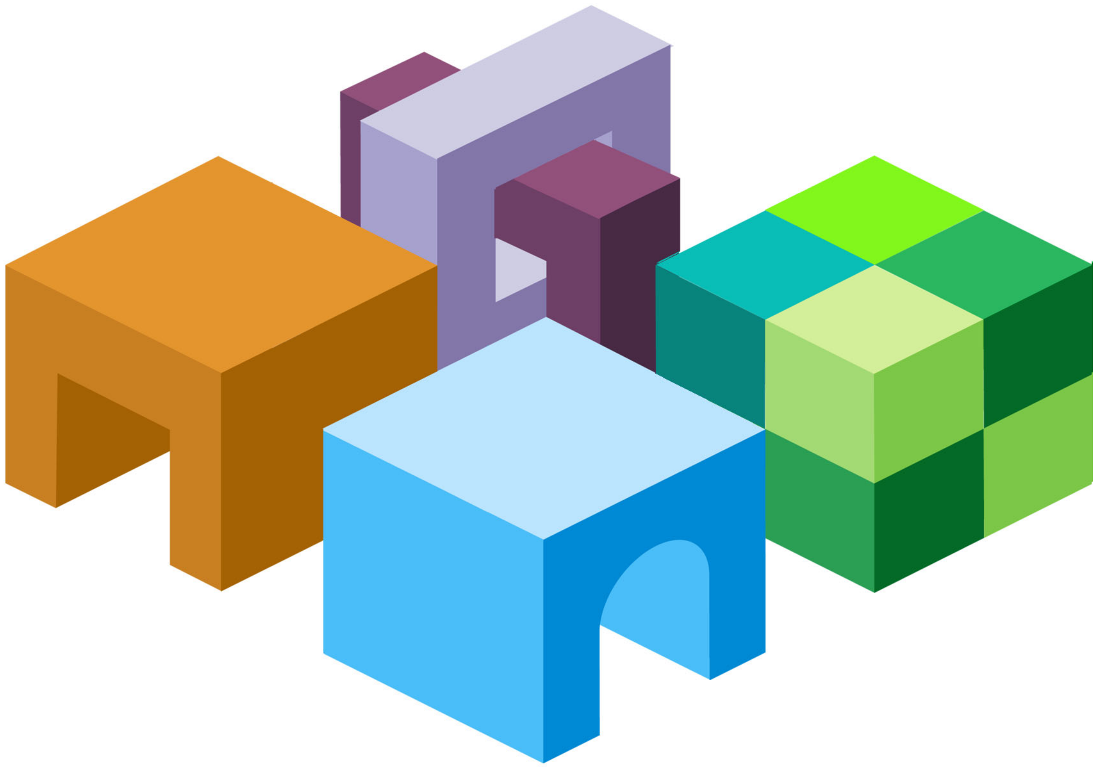


HYPERION® SQR® PRODUCTION REPORTING

RELEASE 11.1.1

DEVELOPER'S GUIDE

VOLUME 1: DESIGNING REPORTS WITH THE SQR PRODUCTION REPORTING LANGUAGE

ORACLE®
ENTERPRISE PERFORMANCE MANAGEMENT SYSTEM

Production Reporting Developer's Guide, 11.1.1

Copyright © 1996, 2008, Oracle and/or its affiliates. All rights reserved.

Authors: EPM Information Development Team

Contents

Part I

Production Reporting Basics

In Production Reporting Basics:

1 Introduction

This guide is designed to help you learn the Oracle's Hyperion® SQR® Production Reporting language, a specialized language for database processing and reporting. By working through code examples, you will learn how to write Production Reporting programs that select data from a database and present it in a report.

If this is your first time using Production Reporting, the first three parts of this guide (through Chapter 16) give you everything that you need to get started. The rest of the guide discusses advanced features and more technical issues.

This guide is filled with real examples and sample programs, and we encourage you to copy code from it. It helps you create the kinds of Production Reporting programs that are important to your organization.

The code examples in this guide demonstrate good Production Reporting programming style. We recommend that you adopt this standard style because it makes your code easier for other Production Reporting programmers to read. We encourage you to try these programs for yourself and to experiment with them. Try making some changes to the samples and see how they run.

To try the sample programs, you must first install Production Reporting Server. If you installed all of the program components, the sample programs are located in:

```
\hyperion\products\biplus\docs\samples\Production Reporting
```

If you did not include the files in the original installation, you can rerun the installation program to install just these files.

You can run the sample programs on any hardware platform, but you may find it somewhat easier to review Production Reporting program results from the Windows platform, and using Production Reporting Viewer or Web browser to check results.

You can run the sample programs without modification against the Oracle, Sybase, and Informix databases. You can also run the samples against other databases with minor modifications.

To run the sample programs, you must create a sample database by running the `loadall.sqr` program.

➤ To run`loadall.sqr`, enter the following command:

```
sqr loadall username/password
```

On Windows platforms, you can run `loadall.sqr` by double-clicking the Loadall icon. If your system does not display this icon, execute `loadall.sqr` from `\hyperion\products\biplus\docs\Server\sample`.

If an individual table exists, you are prompted to:

A—Abort the load

S—Skip the specified table

R—Reload the specified table

C—Reload all tables

➤ To run `loadall.sqr` as a batch program, enter the preferred option (A, S, R, or C) on the command-line.

For example:

```
sqr loadall username/password a
```

You can set up the sample database and run the sample programs with any user name and password. We recommend, however, that you use an account that does not hold important data.

The sample programs are in ASCII format. To successfully run the programs, you must specify an ASCII-derived encoding value in your SQR.INI file. See "Encoding Keys in the [Environment] Section" in Volume 2 of the *Production Reporting Developer's Guide*.

2 A Simple Production Reporting Program

In This Chapter

A Sample Program

The first sample program is the simplest Production Reporting program. It prints a text string.

Program ex1a.sqr

```
begin-program
   print 'Hello, World.' (1,1)
end-program
```

Note:

All program examples and their output files are included with the installation.

Creating and Running Production Reporting Programs

Open a text editor and enter the code printed above exactly as shown or open *Program ex1a.sqr* in the `\hyperion\products\biplus\docs\samples\Production Reporting` directory.

If you are writing the sample program code, save your code.Production Reporting programs are normally given a file extension of SQR. Save this program with the name *ex1a.sqr*.

To run the sample program, go to the directory in which you saved the program and enter the appropriate Production Reporting command at the command prompt or from within the graphical user interface. (On UNIX, Production Reporting is run from the command line. On Windows, you can run Production Reporting from the command line or from a graphical user interface.)

If you are entering the information from the command line, include "sqr", the Production Reporting program name, and the connectivity string all on one line as shown here:

```
[sqr] [program] [connectivity] [flags ...] [args ...] [@file ...]
```

See “Production Reporting Command-line Arguments” and “Production Reporting Command-line Flags” in Volume 2 of the *Production Reporting Developer's Guide* for information on connectivity, flags, and arguments.

In a common configuration, you may run Production Reporting on a Windows platform against an Oracle database located on another machine in your network. In this case, enter the command as:

```
sqr ex1a username/password@servername -KEEP
```

If you correctly replace *username*, *password*, and *servername* with the appropriate information, you should see a command line similar to:

```
sqr ex1a sammy/baker@rome -KEEP
```

To produce the desired output file for this exercise, we used the `-KEEP` flag in our example. Do not worry about its presence at this stage.

Production Reporting Output

Production Reporting normally places program output files in the directory from which you run the program. The output file shares the Production Reporting file name, but the file extension differs.

Output files should appear as soon as your program finishes running. If you specified `-KEEP`, one output file is in SQR Portable Format (recognizable by its SPF extension). SQR Portable Format is discussed later in this book but for now, you can easily view the sample program’s SPF file output, *ex1a.spf*, on Windows platforms with the Production Reporting ViewerGUI (sometimes referred to as an “SPF viewer”) or from within Oracle's Hyperion® SQR® Production Reporting Studio using File, then Open. Viewer is invoked from the command line with “sqrv”.

On Windows and UNIX systems, the program also produces an output file with an LIS extension. You can view LIS files from the command line with `TYPE` on Windows or `CAT`, `MORE`, and `VI` on UNIX.

Output for *Program ex1a.sqr*

```
Hello, World.
```

`^L`, or `<FF>` at the end of the output file ejects the last page. In this book, we do not show the form-feed characters.

Program ex1a.sqr consists of three lines of code, starting with `BEGIN-PROGRAM` and ending with `END-PROGRAM`. These two commands and the code between them define the `PROGRAM` section, which is used to control the processing order. The `PROGRAM` section is required, and you may have only one. It is typically placed at or near the top of the program.

The `PROGRAM` section contains a `PRINT` command, which in this case prints “Hello,World.” This text is enclosed in single quotation marks ('), which are used in Production Reporting to distinguish literal text from other program elements.

The last element of `PRINT` gives the position on the output page. Think of an output page as a grid of lines and columns. (1,1) indicates line one, column one, which is the top left corner of the page.

Note:

In Production Reporting, you must place each command on a new line. You can indent Production Reporting commands.

3 Headings and Footings

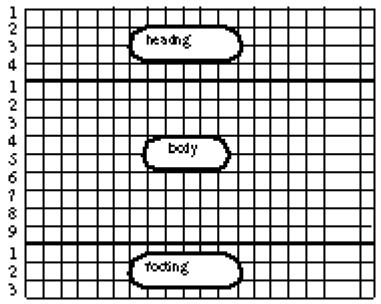
In This Chapter

Subdividing Pages

Typically, every report page has some information about the report, such as the title, the date, and the page number. In Production Reporting, the page can be subdivided into three areas.

- **Heading**—Generally contains the report title and date
- **Body**—Contains report data
- **Footing**—Generally contains the page number

In the following diagram, the heading, body, and footing have independent line numbers. You can print in each page area using line numbers that are relative to the top corner of that area without being concerned about the size of the other areas. In other words, you can print to the first line of the body using line number 1, independent of the heading size.



Adding Headings and Footings

Program ex2a.sqr expands *Program ex1a.sqr* the program described in Chapter 2, "A Simple Production Reporting Program" by adding a page heading and footing.

Program ex2a.sqr

```
begin-program
   print 'Hello, World.' (1,1)
end-program
```

```
begin-heading 1
   print 'Tutorial Report' (1) center
end-heading
begin-footing 1
   ! print "Page n of m" in the footing
   page-number (1,1) 'Page '
   last-page   () ' of '
end-footing
```

Output for *Program ex2a.sqr*

```
Tutorial Report
Hello, World.
Page 1 of 1
```

Page Heading

The HEADING section defines page heading. The section starts with BEGIN-HEADING and ends with END-HEADING. BEGIN-HEADING is followed by the number of lines reserved for the heading. In our example, the heading is one line and consists of the text “Tutorial Report.” The CENTER argument centers the text on the line.

Page Footing

The FOOTING section defines the page footing. The section starts with BEGIN-FOOTING and ends with END-FOOTING. BEGIN-FOOTING is followed by the number 1, which indicates the footing takes one line. This line consists of the text “Page 1 of 1.”

Any space reserved for the heading and footing is taken away from the body. With one line in the heading and footing, the maximum possible size of the report body is reduced by two lines. Line 1 of the body is the first line after the heading.

Comments

The first line in the FOOTING is a comment. Comments are preceded by an exclamation mark, and they extend from the exclamation mark to the end of the line. To print an exclamation mark in report text, type it twice to tell Production Reporting not to take it as the beginning of a comment. For example:

```
print 'Hello, World!!' (1,1)
```

Page Numbering

PAGE-NUMBER prints the text “Page” and the current page number. LAST-PAGE prints the number of the last page, preceded by the word “of,” which is bracketed by spaces. In our example, Production Reporting prints “Page 1 of 1” because there is only one page.

Print Position

Note the parentheses in the `PAGE-NUMBER` and `LAST-PAGE` commands. Numbers in these parentheses give the position for printing. A position in Production Reporting is expressed as three numbers in parentheses—(*line*,*column*,*width*)—where *line* is the line number, *column* is the column (character position), and *width* is the width of the text.

In many cases, a position consists only of the line and column numbers. The width is normally omitted because it defaults to the width of the printed text. If you omit the line and column numbers, the print position defaults to the current position, the position following the last item printed. In the example, `LAST-PAGE` has the position "()" so the current position is the position following the page number.

Print position is a point within the area of the page, or more precisely, within the heading, body, or footing. The position (1,1) in the heading does not share the position (1,1) in the body. Line 1 of the body is the first line below the heading. In the program, the heading has only one line, so line 1 of the body is the second line of the page. Similarly, line 1 of the footing is at the bottom of the page. It is the first line following the body.

Order of Execution

`PRINT` places text in memory, not on paper. Production Reporting prepares a page in memory before printing it to paper, performing the body first, then the `HEADING` and `FOOTING` sections. In this case, "Hello, World" is executed first, then "Tutorial Report" and "Page 1 of 1."

4 Selecting Data

In This Chapter

Sample Code

Following is the sample code used in this chapter. An explanation follows.

Program ex3a.sqr

```
begin-program
   do list_customers
end-program
begin-heading 4
   print 'Customer Listing' (1) center
   print 'Name' (3,1)
   print 'City' (,32)
   print 'State' (,49)
   print 'Phone' (,55)
end-heading
begin-footing 1
   ! Print "Page n of m" in the footing
   page-number (1,1) 'Page '
   last-page   () ' of '
end-footing
begin-procedure list_customers
begin-select
name (,1)
city (,32)
state (,49)
phone (,55)
   position (+1)  ! Advance to the next line
from customers
end-select
end-procedure ! list_customers
```

Output for *Program ex3a.sqr*

```
Customer Listing
Name                            City             State  Phone
Gregory Stonehaven              Everretsville    OH     2165553109
Alfred E Newman & Company       New York         NY     2125552311
```

```
Eliot Richards                 Queens          NY    2125554285
Isaiah J Schwartz and Company  Zanesville      OH    5185559813
Harold Alexander Fink          Davenport       IN    3015553645
Harriet Bailey                 Mamaroneck      NY    9145550144
Clair Butterfield              Teaneck         NJ    2015559901
Quentin Fields                 Cleveland       OH    2165553341
Jerry's Junkyard Specialties   Frogline        NH    6125552877
Kate's Out of Date Dress Shop  New York        NY    2125559000
Sam Johnson                    Bell Harbor     MI    3135556732
Joe Smith and Company          Big Falls       NM    8085552124
Corks and Bottles, Inc.        New York        NY    2125550021
Harry's Landmark Diner         Miningville     IN    3175550948
```

```
Page 1 of 1
```

The `PROGRAM` section consists of one `DO` command, which invokes the procedure *list_customers*.

```
begin-program
   do list_customers
end-program
```

In Production Reporting, a procedure is a group of commands performed in sequence, like a procedure (or subroutine) in other programming languages. A `DO` command invokes a procedure.

We recommend that you break your program logic into procedures and keep the `PROGRAM` section small. It should normally consist of a few `DO` commands for the main report components.

The `HEADING` section creates headings for the report columns. Four lines are reserved for the heading.

```
begin-heading 4
   print 'Customer Listing' (1) center
   print 'Name' (3,1)
   print 'City' (,32)
   print 'State' (,49)
   print 'Phone' (,55)
end-heading
```

The title *Customer Listing* prints on line 1. Line 2 is blank. The first column heading, *Name* is positioned at line 3 of the heading, character position 1. The rest of the column-heading commands omit the line numbers in their positions and default to the current line. Line 4 of the heading is blank.

Using SELECT Statements

Look again at the *list_customers* procedure, which starts with `BEGIN-PROCEDURE` and ends with `END-PROCEDURE`. Note the comment following `END-PROCEDURE`. It shows the procedure is being ended, which is helpful when you have a program with many procedures. (You can also omit the exclamation point: `END-PROCEDURE` *main*.)

The procedure itself contains a `SELECT` paragraph, which starts with `BEGIN-SELECT` and ends with `END-SELECT`.

The SELECT paragraph is unique. It combines an SQL SELECT statement with Production Reporting processing in a seamless way. The SQL statement is:

```
SELECT NAME, CITY, STATE, PHONE
FROM CUSTOMERS
```

SELECT Statement Syntax

In a Production Reporting SELECT paragraph, the word SELECT is omitted, and there are no commas between the column names. Instead, each column is on its own line. You can place Production Reporting commands between the column names, and these commands are executed for every record that the SELECT fetches.

Note:

Since the SELECT * FROM statement is not allowed in Production Reporting, you must name each individual column in a table.

Production Reporting distinguishes column names from Production Reporting commands in a SELECT paragraph by their indentation. Place column names at the beginning of a line, and indent Production Reporting commands at least one space. In the example under Positioning Data, POSITION is indented to prevent it from being taken as a column name. The word FROM must be the first word in a line. The rest of the SELECT statement is written after SQL syntax.

Think of SELECT paragraphs as loops. Production Reporting commands, including column printing, are executed in loops, once for each record that SELECT returns. The loop ends after the last record is returned.

Positioning Data

In the SELECT statement, you see positioning after each column name. This positioning implies a PRINT command for that column. As before, omitting the line number in the position lets it default to the current line.

```
begin-select
name (,1)
city (,32)
state (,49)
phone (,55)
   position (+1)  ! Advance to the next line
from customers
end-select
```

The implied PRINT command is an Production Reporting feature designed to save coding time. It only works inside a SELECT statement.

A POSITION command: POSITION(+1) appears after the last column. The plus sign (or minus sign) indicates relative positioning in Production Reporting. A plus sign moves the print position forward from the current position, and a minus sign moves it back. The “+1” in the sample program means one line down from the current line. This command advances the current print position to the next line.

When you indicate print positions using plus or minus signs, be sure your numbers do not specify a position outside the page boundaries.

5

Column Variables

In This Chapter

Using Column Variables in Conditions

When you select columns from the database in a `SELECT` statement, you can immediately print them using a position. For example:

```
begin-select
phone (,1)
   position (+1)
from customers
end-select
```

But what if you want to use the value of *phone* for another purpose, for example, in a condition? The following example shows you how to do this.

```
begin-program
   do list_customers
end-program
begin-procedure list_customers
begin-select
phone
   if &phone = ''
      print 'No phone' (,1)
   else
      print &phone (,1)
   end-if
   position (+1)
from customers
end-select
end-procedure ! list_customers
```

The *phone* column is an Production Reporting *column variable*. Column variables are preceded with an ampersand (&).

Unlike other program variables, column variables are read-only. You can use their existing value, but you cannot assign them a new value.

In the sample program, *&phone* is a column variable that you can use in Production Reporting commands as if it were a string, date, or numeric variable, depending on its contents. In the condition, *&phone* is compared to ' ', an empty string. If *&phone* is an empty string, the program prints “No phone” instead.

Changing Column Variable Names

Note that the column variable *&phone* inherited its name from the *phone* column. This is the default, but you can change it, as the following example demonstrates.

```
begin-select
phone &cust_phone
   if &cust_phone = ''
      print 'No phone' (,1)
   else
      print &cust_phone (,1)
end-if
   position (+1)
from customers
end-select
```

One reason to change the name of a column variable is that you may want to use a selected column in an expression that has no name. For example:

```
begin-select
count(name) &cust_cnt (,1)
   if &cust_cnt < 100
      print 'Less than 100 customers'
   end-if
   position (+1)
from customers
group by city, state
end-select
```

In this example, the expression `count(name)` is selected. In the program, you store this expression in the column variable *&cust_cnt* and refer to it afterwards by that name.

6

Break Logic

In This Chapter

About Breaks

A *break* is a change in the value of a column or variable. For example, records with the same value for *state* logically belong to a group. When a break occurs, a new group begins.

Reasons to use break logic in a report include:

- Adding white space to reports
- Avoiding printing redundant data
- Performing conditional processing on variables that change
- Printing subtotals

For example, you may want to prepare a sales report with records grouped by product, region, or salesperson. Using break logic, you can print column headings, count records, subtotal columns, and perform additional processing on the count or subtotal.

To see how breaks work, write a program similar to the one in Chapter 4, "Selecting Data" and then add break logic to it. The break logic makes the grouping more apparent.

Here is the program without break logic.

Program ex5a.sqr

```
begin-program
   do list_customers
end-program
```

```
begin-heading 2
   print 'State' (1,1)
   print 'City'  (1,7)
   print 'Name'  (1,24)
   print 'Phone' (1,55)
end-heading
begin-procedure list_customers
begin-select
state (,1)
city  (,7)
name  (,24)
phone (,55)
   position (+1)  ! Advance to the next line
from customers
order by state, city, name
end-select
end-procedure ! list_customers
```

Output for *Program ex5a.sqr*

```
State City             Name                          Phone
IN    Davenport        Harold Alexander Fink         3015553645
IN    Miningville      Harry's Landmark Diner        3175550948
MI    Bell Harbor      Sam Johnson                   3135556732
NH    Frogline         Jerry's Junkyard Specialties  6125552877
NJ    Teaneck          Clair Butterfield             2015559901
NM    Big Falls        Joe Smith and Company         8085552124
NY    Mamaroneck       Harriet Bailey                9145550144
NY    New York         Alfred E Newman & Company     2125552311
NY    New York         Corks and Bottles, Inc.       2125550021
NY    New York         Kate's Out of Date Dress Shop 2125559000
NY    Queens           Eliot Richards                2125554285
OH    Cleveland        Quentin Fields                2165553341
OH    Everretsville    Gregory Stonehaven            2165553109
OH    Zanesville       Isaiah J Schwartz and Company 5185559813
```

When you sort the output by *state*, *city*, and *name* (note the ORDER BY clause in BEGIN-SELECT), the records are grouped by *state*. To make the grouping more apparent, you can add a break.

Using ON-BREAK

In the program shown here, using ON-BREAK starts a new group each time the value of *state* changes, and prints *state* only when its value changes. ON-BREAK works for both implicit and explicit PRINT commands. In *Program ex5a.sqr*, *state*, *city*, *name*, and *phone* are implicitly printed as part of the SELECT statement.

Program ex5b.sqr is identical to *Program ex5a.sqr* with the exception of the line that prints the *state* column. This line appears in bold. With break processing, the state abbreviation is printed only once for each group.

Program ex5b.sqr

```
begin-program
   do list_customers
```

```
end-program
begin-heading 2
   print 'State' (1,1)
   print 'City'  (1,7)
   print 'Name'  (1,24)
   print 'Phone' (1,55)
end-heading
begin-procedure list_customers
begin-select
state (,1) on-break
city  (,7)
name  (,24)
phone (,55)
   position (+1)  ! Advance to the next line
from customers
order by state, city, name
end-select
end-procedure ! list_customers
```

Output for *Program ex5b.sqr*

```
State City             Name                           Phone
IN    Davenport        Harold Alexander Fink          3015553645
      Miningville      Harry's Landmark Diner         3175550948
MI    Bell Harbor      Sam Johnson                    3135556732
NH    Frogline         Jerry's Junkyard Specialties   6125552877
NJ    Teaneck          Clair Butterfield              2015559901
NM    Big Falls        Joe Smith and Company          8085552124
NY    Mamaroneck       Harriet Bailey                 9145550144
      New York         Alfred E Newman & Company      2125552311
      New York         Corks and Bottles, Inc.        2125550021
      New York         Kate's Out of Date Dress Shop  2125559000
      Queens           Eliot Richards                 2125554285
OH    Cleveland        Quentin Fields                 2165553341
      Everretsville    Gregory Stonehaven             2165553109
      Zanesville       Isaiah J Schwartz and Company  5185559813
```

Skipping Lines Between Groups

You can further enhance the visual effect of break processing by inserting one or more lines between groups. To do so, use the SKIPLINES qualifier with ON-BREAK. Here is the *list_customers* procedure from *Program ex5b.sqr*, with the modified line shown in bold.

```
begin-select
state (,1) on-break skiplines=1
city  (,7)
name  (,24)
phone (,55)
   position (+1)  ! Advance to the next line
from customers
order by state, city, name
end-select
```

Output for modified *Program ex5b.sqr*

```
State City             Name                           Phone
IN    Davenport        Harold Alexander Fink          3015553645
      Miningville      Harry's Landmark Diner         3175550948
MI    Bell Harbor      Sam Johnson                    3135556732
NH    Frogline         Jerry's Junkyard Specialties   6125552877
NJ    Teaneck          Clair Butterfield              2015559901
NM    Big Falls        Joe Smith and Company          8085552124
NY    Mamaroneck       Harriet Bailey                 9145550144
      New York         Alfred E Newman & Company      2125552311
      New York         Corks and Bottles, Inc.        2125550021
      New York         Kate's Out of Date Dress Shop  2125559000
      Queens           Eliot Richards                 2125554285
OH    Cleveland        Quentin Fields                 2165553341
      Everretsville    Gregory Stonehaven             2165553109
      Zanesville       Isaiah J Schwartz and Company  5185559813
```

Arranging Multiple Break Columns

As you can see in the previous example, you can also have multiple customers within a city. You can apply the same break concept to the *city* column to make this grouping of customers more apparent. Add another ON-BREAK to the program so that *city* prints only when its value changes.

Multiple breaks must be arranged in a hierarchy. In the sample program, the breaks concern geographical units, so it is logical to arrange them according to size—first *state*, then *city*. This sort of arrangement is called *nesting*.

The LEVEL keyword ensures that breaks are properly nested by numbering breaks by level and by specifying that columns print in order of increasing break levels, from left to right. Number breaks in the same order in which they are sorted in the ORDER BY clause. See "Understanding Event Order" on page 38. LEVEL controls the order in which you call break procedures. See "Setting Break Procedures with BEFORE and AFTER" on page 38.

The next example is identical to *Program ex5a.sqr* with the exception of the two lines that print the *state* and *city* columns. These two lines are shown in bold.

Program ex5c.sqr

```
begin-program
   do list_customers
end-program
begin-heading 2
   print 'State' (1,1)
   print 'City'  (1,7)
   print 'Name'  (1,24)
   print 'Phone' (1,55)
end-heading
begin-procedure list_customers
begin-select
state (,1) on-break level=1
city  (,7) on-break level=2
name  (,24)
phone (,55)
```

```
   position (+1)  ! Advance to the next line
from customers
order by state, city, name
end-select
end-procedure ! list_customers
```

Output for *Program ex5c.sqr*

```
State City           Name                           Phone
IN    Davenport      Harold Alexander Fink          3015553645
      Miningville    Harry's Landmark Diner         3175550948
MI    Bell Harbor    Sam Johnson                    3135556732
NH    Frogline       Jerry's Junkyard Specialties   6125552877
NJ    Teaneck        Clair Butterfield              2015559901
NM    Big Falls      Joe Smith and Company          8085552124
NY    Mamaroneck     Harriet Bailey                 9145550144
      New York       Alfred E Newman & Company      2125552311
                     Corks and Bottles, Inc.        2125550021
                     Kate's Out of Date Dress Shop  2125559000
      Queens         Eliot Richards                 2125554285
OH    Cleveland      Quentin Fields                 2165553341
      Everretsville  Gregory Stonehaven             2165553109
      Zanesville     Isaiah J Schwartz and Company  5185559813
```

Note that three customers are in New York, so the city name for the second and third customers is left blank.

Break Processing with Enhancements

Break logic enhances reports by controlling page breaks or calculating counts and totals for the ON-BREAK column. *Program ex5d.sqr* illustrates these techniques. The program selects the customer's name, address, and telephone number from the database and preforms break processing on the *state* column.

Program ex5d.sqr

```
begin-program
  do list_customers
end-program
begin-heading 4
  print 'Customers Listed by State' (1) center
  print $current-date    (1,1) Edit 'DD-Mon-YYYY'
  print 'State' (3,1)
  print 'Customer Name, Address and Phone Number' (,11)
  print '-' (4,1,9) fill
  print '-' (4,11,40) fill
end-heading
begin-footing 2
  ! print "Page n of m"
  page-number (1,1) 'Page '
  last-page   () ' of '
end-footing
begin-procedure state_tot
  print '   Total Customers for State:  ' (+1,1)
  print #state_total () edit 999,999
  position (+3,1)                      ! Leave 2 blank lines.
  let #cust_total = #cust_total + #state_total
  let #state_total = 0
    next-listing need=4
end-procedure ! state_tot
begin-procedure list_customers
let #state_total = 0
let #cust_total = 0
begin-select
! The 'state' field will only be printed when it
! changes. The procedure 'state_tot' will also be
! executed only when the value of 'state' changes.
state       (,1)  on-break print=change/top-page after=state_tot
name        (,11)
addr1       (+1,11)   ! continue on second line
addr2       (+1,11)   ! continue on third line
city        (+1,11)   ! continue on fourth line
phone            (,+2) edit (xxx)bxxx-xxxx ! Edit for easy reading.
  ! Skip 1 line between listings.
  ! Since each listing takes 4 lines, we specify 'need=4' to
  ! prevent a customer's data from being broken across two pages.
  next-listing skiplines=1 need=4
  let #state_total = #state_total + 1
from customers
order by state, name
end-select
if #cust_total > 0
  print '   Total Customers: ' (+3,1)
  print #cust_total () edit 999,999  ! Total customers printed.
else
  print 'No customers.' (1,1)
end-if
end-procedure ! list_customers
```

Output for *Program ex5d.sqr*

```
29-Apr-2001              Customers Listed by State
State     Customer Name, Address and Phone Number
--------- ----------------------------------------
IN        Harold Alexander Fink
          32077 Cedar Street
          West End
          Davenport  (301) 555-3645
          Harry's Landmark Diner
          17043 Silverfish Road
          South Park
          Miningville  (317) 555-0948
   Total Customers for State:        2
MI        Sam Johnson
          37 Cleaver Street
          Sandy Acres
          Bell Harbor  (313) 555-6732
   Total Customers for State:        1
NH        Jerry's Junkyard Specialties
          Crazy Lakes Cottages
          Rural Delivery #27
          Frogline  (612) 555-2877
   Total Customers for State:        1
NJ        Clair Butterfield
          371 Youngstown Blvd
          Quit Woods
          Teaneck  (201) 555-9901
   Total Customers for State:        1
NM        Joe Smith and Company
          1711 Sunset Blvd
          East River
          Big Falls  (808) 555-2124
   Total Customers for State:        1
NY        Alfred E Newman & Company
          2837 East Third Street
          Greenwich Village
          New York  (212) 555-2311
Page 1 of 2
29-Apr-2001              Customers Listed by State
State     Customer Name, Address and Phone Number
--------- ----------------------------------------
NY        Corks and Bottles, Inc.
          167 East Blvd.
          Jamaica
          New York  (212) 555-0021
          Eliot Richards
          2134 Partridge Ave
          Jamaica
          Queens  (212) 555-4285
          Harriet Bailey
          47 Season Street
          Bellevue Park
          Mamaroneck  (914) 555-0144
          Kate's Out of Date Dress Shop
          2100 Park Ave
          East Side City
          New York  (212) 555-9000
   Total Customers for State:        5
OH        Gregory Stonehaven
          Middlebrook Road
          Grey Quarter
          Everretsville  (216) 555-3109
          Isaiah J Schwartz and Company
          37211 Columbia Blvd
          Sweet Acres
          Zanesville  (518) 555-9813
          Quentin Fields
          37021 Cedar Road
          Beachwood
          Cleveland  (216) 555-3341
   Total Customers for State:        3
     Total Customers:      14
Page 2 of 2
```

Take a close look at the code. The data prints using a `select` paragraph in the *list_customers* procedure. The state and the customer name print on the first line. The customer's address and phone number print on the next three lines.

The program also uses the argument AFTER=STATE_TOT. This argument calls the *state_tot* procedure after each change in the value of *state*. Processing order is explained in "Setting Break Procedures with BEFORE and AFTER" on page 38.

Handling Page Breaks

If a page break occurs within a group, you may want to reprint headings and the value of the break column at the top of the new page.

To control the printing of the value, use PRINT=CHANGE/TOP-PAGE. With this qualifier, the value of ON-BREAK prints when it changes and after every page break. In this example, the value of state prints not only when it changes, but whenever the report starts a new page.

To format records, use NEXT-LISTING. This command serves two purposes. The SKIPLINES=1 argument skips one line between records, then renumbers the current line as line 1. The NEED=4 argument prevents a listing from splitting over two pages by specifying the minimum number of lines needed to write a new listing on the current page. In this case, if fewer than four lines are left on a page, Production Reporting starts a new page.

Printing Dates

In the HEADING section, the reserved variable *$current-date* prints the date and the time. This variable is initialized with the date and time of the client machine at the start of program execution.Production Reporting provides predefined, or reserved, variables for a variety of uses. For a complete listing of reserved variables, see Volume 2 of the *Production Reporting Developer's Guide.*

In this example, the complete command is:

```
PRINT $current-date (1,1) EDIT 'DD/MM/YYYY'
```

This prints the date and time at position 1,1 of the heading. The EDIT argument specifies an *edit mask,* or format, for printing the date. Production Reporting provides a large variety of edit masks for use in formatting numbers, dates, and strings. See Volume 2 of the *Production Reporting Developer's Guide.*

Note that the PRINT command for the report title precedes the command for the *$current-date* reserved variable, even though the date is on the left and the title is on the right. Production Reporting assembles a page in memory before printing, so the order of these commands does not matter as long as you use the correct print position qualifiers.

The last two commands in the HEADING section print a string of hyphens under the column headings. The FILL option in the PRINT command fills the specified width with a pattern. This is a good way to print a line.

In the FOOTING section, we print the "Page *n* of *m*" as we did in earlier examples.

Obtaining Totals

Program ex5d.sqr prints two totals—a subtotal of customers in each state and a grand total of all customers. These calculations are performed with two numeric variables, one for the subtotals and one for the grand totals. Their names are *#state_total* and *#cust_total*, respectively.

Production Reporting has a small set of variable types. The most common types are numeric variables and string variables. All numeric variables are preceded with a pound sign (#) and all string variables are preceded with a dollar sign ($). An additional Production Reporting variable type is the date variable (see Chapter 24, "Working with Dates.").

In Production Reporting, numeric and string variables are implicitly defined by their first use. All numeric variables start out as zero and all string variables start out as null, so there is normally no need to initialize them. String variables vary in length. Assigning a new value to a string variable automatically adjusts its length.

At the beginning of the *list_customers* procedure, *#state_total* and *#cust_total* are set to zero. This initialization is optional and is done for clarity only. The variable *#state_total* is incremented by 1 for every row selected.

When the value of *state* changes, the program calls the *state_tot* procedure and prints the value of *#state_total*. The edit mask, `EDIT 999,999`, formats the number.

This procedure also employs the `LET` command. `LET` is the assignment command in Production Reporting, for building complex expressions. Here, `LET` adds the value of *#state_total* to *#cust_total*. At the end of the procedure, *#state_total* is reset to zero.

The *list_customers* procedure incorporates if-then-else logic. The condition starts with `IF` followed by an expression. If the expression evaluates to true or to a number other than zero, the subsequent commands execute. Otherwise, if there is an `ELSE` part to the `IF`, those commands execute. `IF` commands end with an `END-IF`.

In *ex5d.sqr*, the value of *#cust_total* is examined. If it is greater than zero, the query returned rows of data, and the program prints the string *Total Customers:* and the value of *#cust_total*.

If *#cust_total* equals zero, the query did not return any data. In this case, the program prints the string *No customers*.

Hyphens and Underscores

Many Production Reporting commands, such as `BEGIN-PROGRAM` and `BEGIN-SELECT`, use a hyphen, whereas procedure and variable names use an underscore.

Procedure and variable names can contain either a hyphen or underscore, but we strongly recommend you use an underscore. Using underscores in procedure and variable names helps you distinguish them from Production Reporting commands. It also prevents confusion when variable names and numbers are mixed in an expression, where hyphens could be mistaken for minus signs.

Setting Break Procedures with BEFORE and AFTER

When you print variables with ON-BREAK, you can automatically call procedures before and after each break in a column. The BEFORE and AFTER qualifiers give you this capability. For example:

```
begin-select
state (,1) on-break before=state_heading after=state_tot
```

BEFORE calls the state_heading procedure to print headings before each group of records of the same state. Similarly, AFTER calls the state_tot procedure to print totals after each group of records.

All BEFORE procedures are invoked before each break, including the first group before the SELECT is processed. Similarly, all AFTER procedures are invoked after each break, including the last group upon completion of the SELECT.

Understanding Event Order

Use the LEVEL qualifier of ON-BREAK to define a hierarchy of break columns. In *ex5c.sqr*, state was defined as LEVEL=1 and city as LEVEL=2.

When a break occurs at one level, it also forces breaks on variables with higher LEVEL qualifiers. In the sample program, a break on state also means a break on city.

A break on a variable can trigger many other events. The value can be printed, lines skipped, procedures automatically called, and the old value saved. It is important to know the order of events, particularly where multiple ON-BREAK columns exist.

The following SELECT statement has breaks on three levels.

```
begin-select
state  (,1)   on-break level=1           after=state_tot   skiplines=2
city (,7)   on-break level=2        after=city_tot  skiplines=1
zip   (,45)  on-break level=3        after=zip_tot
from customers
order by state, city, zip
end-select
```

The breaks are processed as follows:

1. When *zip* breaks, the *zip_tot* procedure executes.
2. When *city* breaks, first the *zip_tot* procedure executes, then the *city_tot* procedure executes, and one line is skipped (SKIPLINES=1). Both *city* and *zip* print in the next record.
3. When *state* breaks, the *zip_tot*, *city_tot*, and *state_tot* procedures are processed in that order. One line is skipped after the *city_tot* procedure executes, and two lines are skipped after the *state_tot* procedure executes. All three columns—*state*, *city*, and *zip*—print in the next record.

Program ex5e.sqr demonstrates the order of events in break processing. It has three ON-BREAK columns, each with a LEVEL argument and a BEFORE and AFTER procedure. BEFORE and AFTER print strings to indicate the processing order.

Program ex5e.sqr

```
begin-setup
        declare-Layout
        default
        end-declare
end-setup
begin-program
 do main
end-program
begin-procedure a
print 'AFTER Procedure for state LEVEL 1' (+1,40)
end-procedure
begin-procedure b
print 'AFTER Procedure city LEVEL 2' (+1,40)
end-procedure
begin-procedure c
print 'AFTER Procedure zip LEVEL 3' (+1,40)
end-procedure
begin-procedure aa
print 'BEFORE Procedure state LEVEL 1' (+1,40)
end-procedure
begin-procedure bb
print 'BEFORE Procedure city LEVEL 2' (+1,40)
end-procedure
begin-procedure cc
print 'BEFORE Procedure zip LEVEL 3' (+1,40)
end-procedure
begin-procedure main local
begin-select
        add 1 to #count
        print 'Retrieved row #' (+1,40)
        print #count (,+10)Edit 9999
        position (+1)
state   (3,1) On-Break Level=1 after=a before=aa
city    (3,10) On-Break Level=2 after=b before=bb
zip     (3,25) On-Break Level=3 after=c before=cc Edit xxxxx
 next-listing  Need=10
from customers
order by state,city,zip
end-select
end-procedure
begin-heading 3
 print $current-date (1,1) edit 'DD-MM-YYYY'
 page-number (1,60) 'Page '
 last-page () ' of '
print 'STATE'  (3,1)
 print 'CITY'   (3,10)
 print 'ZIP'    (3,25)
 print 'Break Processing sequence' (3,40)
end-heading
```

Output for *Program ex5e.sqr*

```
15-10-2002                                                 Page 1 of 3
STATE    CITY           ZIP       Break Processing sequence
                                  BEFORE Procedure state LEVEL 1
```

```
DE       Dover          20652     BEFORE Procedure city LEVEL 2
                                  BEFORE Procedure zip LEVEL 3
                                  Retrieved row #                1
                                  Retrieved row #                2
IN       Davenport      62130
                                  AFTER Procedure zip LEVEL 3
                                  AFTER Procedure city LEVEL 2
                                  AFTER Procedure for state LEVEL 1
                                  BEFORE Procedure state LEVEL 1
                                  BEFORE Procedure city LEVEL 2
                                  BEFORE Procedure zip LEVEL 3
                                  Retrieved row #                3
        Fort Wayne     40622
                                  AFTER Procedure zip LEVEL 3
                                  AFTER Procedure city LEVEL 2
                                  BEFORE Procedure city LEVEL 2
                                  BEFORE Procedure zip LEVEL 3
                                  Retrieved row #               4
Miningville    40622
                                  AFTER Procedure zip LEVEL 3
                                  AFTER Procedure city LEVEL 2
                                  BEFORE Procedure city LEVEL 2
                                  BEFORE Procedure zip LEVEL 3
                                  Retrieved row #               5
MI       Bell Harbor    40674
                                  AFTER Procedure zip LEVEL 3
                                  AFTER Procedure city LEVEL 2
                                  AFTER Procedure for state LEVEL 1
                                  BEFORE Procedure state LEVEL 1
                                  BEFORE Procedure city LEVEL 2
                                  BEFORE Procedure zip LEVEL 3
                                  Retrieved row #               6
NH       Frogline       04821
                                  AFTER Procedure zip LEVEL 3
                                  AFTER Procedure city LEVEL 2
                                  AFTER Procedure for state LEVEL 1
                                  BEFORE Procedure state LEVEL 1
                                  BEFORE Procedure city LEVEL 2
                                  BEFORE Procedure zip LEVEL 3
                                  Retrieved row #               7
NJ       Teaneck        00355
                                  AFTER Procedure zip LEVEL 3
                                  AFTER Procedure city LEVEL 2
                                  AFTER Procedure for state LEVEL 1
                                  BEFORE Procedure state LEVEL 1
                                  BEFORE Procedure city LEVEL 2
                                  BEFORE Procedure zip LEVEL 3
                                  Retrieved row #               8
15-10-2002                                             Page 2 of 3
STATE    CITY           ZIP       Break Processing sequence
NM       Big Falls      87893
                                  AFTER Procedure zip LEVEL 3
                                  AFTER Procedure city LEVEL 2
                                  AFTER Procedure for state LEVEL 1
                                  BEFORE Procedure state LEVEL 1
                                  BEFORE Procedure city LEVEL 2
                                  BEFORE Procedure zip LEVEL 3
```

```
                                      Retrieved row #               9
NY       Mamaroneck     10833
                                      AFTER Procedure zip LEVEL 3
                                      AFTER Procedure city LEVEL 2
                                      AFTER Procedure for state LEVEL 1
                                      BEFORE Procedure state LEVEL 1
                                      BEFORE Procedure city LEVEL 2
                                      BEFORE Procedure zip LEVEL 3
                                      Retrieved row #               10
         New York       10002
                                      AFTER Procedure zip LEVEL 3
                                      AFTER Procedure city LEVEL 2
                                      BEFORE Procedure city LEVEL 2
                                      BEFORE Procedure zip LEVEL 3
                                      Retrieved row #               11
                        10134
                                      AFTER Procedure zip LEVEL 3
                                      BEFORE Procedure zip LEVEL 3
                                      Retrieved row #               12
                        10204
                                      AFTER Procedure zip LEVEL 3
                                      BEFORE Procedure zip LEVEL 3
                                      Retrieved row #               13
         Queens         10213
                                      AFTER Procedure zip LEVEL 3
                                      AFTER Procedure city LEVEL 2
                                      BEFORE Procedure city LEVEL 2
                                      BEFORE Procedure zip LEVEL 3
                                      Retrieved row #               14
OH       Cleveland      44121
                                      AFTER Procedure zip LEVEL 3
                                      AFTER Procedure city LEVEL 2
                                      AFTER Procedure for state LEVEL 1
                                      BEFORE Procedure state LEVEL 1
                                      BEFORE Procedure city LEVEL 2
                                      BEFORE Procedure zip LEVEL 3
                                      Retrieved row #               15
15-10-2002                                                  Page 3 of 3
STATE    CITY           ZIP           Break Processing sequence
         Everretsville  40233
                                      AFTER Procedure zip LEVEL 3
                                      AFTER Procedure city LEVEL 2
                                      BEFORE Procedure city LEVEL 2
                                      BEFORE Procedure zip LEVEL 3
                                      Retrieved row #               16
         Zanesville     44900
                                      AFTER Procedure zip LEVEL 3
                                      AFTER Procedure city LEVEL 2
                                      BEFORE Procedure city LEVEL 2
                                      BEFORE Procedure zip LEVEL 3
                                      Retrieved row #              17
PA       Pittsburgh     90672
                                      AFTER Procedure zip LEVEL 3
                                      AFTER Procedure city LEVEL 2
                                      AFTER Procedure for state LEVEL 1
                                      BEFORE Procedure state LEVEL 1
BEFORE Procedure city LEVEL 2
BEFORE Procedure zip LEVEL 3
AFTER Procedure zip LEVEL 3
AFTER Procedure city LEVEL 2
AFTER Procedure for state LEVEL 1
```

These steps explain the order of processing in detail.

1. Process BEFORE procedures in ascending order by LEVEL before retrieving the first row of the query.

   If no data is selected, BEFORE procedures do not execute.

2. Select the first row of data.
3. Select subsequent rows of data.

   Processing of the SELECT command continues. When a break occurs on any column, it triggers breaks on columns at the same or higher levels.

4. Process AFTER procedures in descending order from the highest level to the level of the current ON-BREAK column.
5. Set SAVE variables with the value of the previous ON-BREAK column. (See "Saving Values When Breaks Occur" on page 43.)
6. Process BEFORE procedures in ascending order from the current level to the highest level.
7. If SKIPLINES was specified, advance the current line position.
8. Print the value of the new group (unless PRINT=NEVER is specified).
9. Process AFTER procedures.

   After the SELECT is complete, if any rows were selected, AFTER procedures are processed in descending order by LEVEL.

Controlling Page Breaks with Multiple ON-BREAK Columns

Where multiple columns have ON-BREAK, page breaks call for careful planning. While it may be acceptable to have a page break within a group, you probably would not want to have one within a record.

To prevent page breaks within a record:

- Place ON-BREAK columns ahead of other columns in the SELECT statement.
- Place lower-level ON-BREAK columns ahead of higher-level ON-BREAK columns in the SELECT statement.
- Use the same line positions for all ON-BREAK columns.
- Avoid using WRAP and ON-BREAK together on one column.

Saving Values When Breaks Occur

In *ex5d.sqr*, `state_tot` prints the total number of customers per state. Because it is called with the `AFTER` argument, this procedure executes only after the value of the `ON-BREAK` column, state, changes.

Sometimes, however, you may want to print the previous value of the `ON-BREAK` column in the `AFTER` procedure. For example, you may want to print the state name along with the totals for each state. Simply printing the value of state does not work because its value changes by the time `AFTER` is called.

The answer is to save the previous break value in a string variable. To do this, use the `SAVE` qualifier of `ON-BREAK`. For example:

```
begin-select
state (,1) on-break after=state_tot save=$old_state
```

You can then print the value of $old_*state* in the `state_tot` procedure.

Using ON-BREAK on Hidden Columns

In some reports, you may want to use the features of break processing without printing `ON-BREAK`. For example, you may want to incorporate `ON-BREAK` into a subheading. This format might make your report more readable. It is also useful when you want to leave room on the page for additional columns.

To create such a report, use `PRINT=NEVER` to "hide" the break variable and print it in a heading procedure called by `BEFORE`.

Program ex5f.sqr is based on *Program ex5b.sqr*. The key lines are shown in bold.

Program ex5f.sqr

```
begin-program
   do list_customers
end-program
begin-procedure list_customers
begin-select
state () on-break before=state_heading print=never level=1
city  (,1) on-break level=2
name  (,18)
phone (,49)
   position (+1) ! Advance to the next line
from customers
order by state, city, name
end-select
end-procedure ! list_customers
begin-procedure state_heading
   print 'State: ' (+1,1) bold    ! Advance a line and print 'State:'
   print &state (,8) bold      ! Print the state column here
   print 'City' (+1,1) bold    ! Advance a line and print 'City'
   print 'Name' (,18) bold
   print 'Phone' (,49) bold
   print '-' (+1,1,58) fill
   position (+1)            ! Advance to the next line
end-procedure ! state_heading
```

Note that this program has no HEADING section. Instead, a procedure prints column headings for each state rather than at the top of each page.

Note that you can reference the &*state* variable throughout the program, even though the *state* column did not print as part of the break.

The following line from the SELECT statement defines the break processing for *state*. The BEFORE qualifier specifies that the *state_heading* procedure is called automatically before each change in *state*. In this program, this break is set to LEVEL=1.

```
state () on-break before=state_heading print=never level=1
```

PRINT=NEVER hides the *state* column and specifies that it does not print as part of the SELECT statement. Instead, it prints in the *state_heading* procedure. In this procedure, the *state* column is referred to as the column variable &*state*.

The *city* column is assigned a LEVEL=2 break.

Output for *Program ex5f.sqr*

```
State: IN
City             Name                          Phone
----------------------------------------------------------
Davenport        Harold Alexander Fink         3015553645
Miningville      Harry's Landmark Diner        3175550948
State: MI
City             Name                          Phone
----------------------------------------------------------
Bell Harbor      Sam Johnson                   3135556732
State: NH
City             Name                          Phone
----------------------------------------------------------
Frogline         Jerry's Junkyard Specialties  6125552877
State: NJ
City             Name                          Phone
----------------------------------------------------------
Teaneck          Clair Butterfield             2015559901
State: NM
City             Name                          Phone
----------------------------------------------------------
Big Falls        Joe Smith and Company         8085552124
State: NY
City             Name                          Phone
----------------------------------------------------------
Mamaroneck       Harriet Bailey                9145550144
New York         Alfred E Newman & Company     2125552311
                 Corks and Bottles, Inc.       2125550021
                 Kate's Out of Date Dress Shop 2125559000
Queens           Eliot Richards                2125554285
State: OH
City             Name                          Phone
----------------------------------------------------------
Cleveland        Quentin Fields                2165553341
```

```
Everretsville    Gregory Stonehaven            2165553109
Zanesville       Isaiah J Schwartz and Company  5185559813
```

Restrictions and Limitations of ON-BREAK

You cannot use `ON-BREAK` with Production Reporting numeric variables. To perform break processing on a numeric variable, move its value to a string variable and set `ON-BREAK` on that. For example:

```
begin-select
amount_received &amount
  move &amount to $amount $$9,999.99
  print $amount (+1,1) on-break
from cash_receipts
order by amount_received
end-select
```

7

SETUP Section

In This Chapter

About the Setup Section

The SETUP section holds all *declarations*. Declarations define certain report characteristics and the source and attributes of various report components, such as charts and images. The SETUP section is evaluated when your program is compiled. The SETUP section is not required in a program, but it is very useful.

Creating the SETUP Section

If present, the SETUP section is typically placed at the top of the program before the PROGRAM section. It begins with BEGIN-SETUP and ends with END-SETUP.

Commands in the SETUP section are processed at compile time, before program execution. For more information about the commands in Table 1, see Volume 2 of the *Production Reporting Developer's Guide*.

Table 1 Commands Available in the SETUP Section

Command	Comments
ASK	Allowed only in SETUP section
BEGIN-SQL	Can also appear in a procedure Executed when a run-time file (SQT) is loaded
CREATE-ARRAY	Can also appear in a procedure
DECLARE-CHART	
DECLARE-COLOR-MAP	
DECLARE-CONNECTION	DDO only

Command	Comments
DECLARE-IMAGE	
DECLARE-LAYOUT	
DECLARE-PRINTER	
DECLARE-PROCEDURE	
DECLARE-REPORT	
DECLARE-TOC	
DECLARE-VARIABLE	Can also appear in a local procedure
LOAD-LOOKUP	Can also appear in a procedure
USE	Sybase only

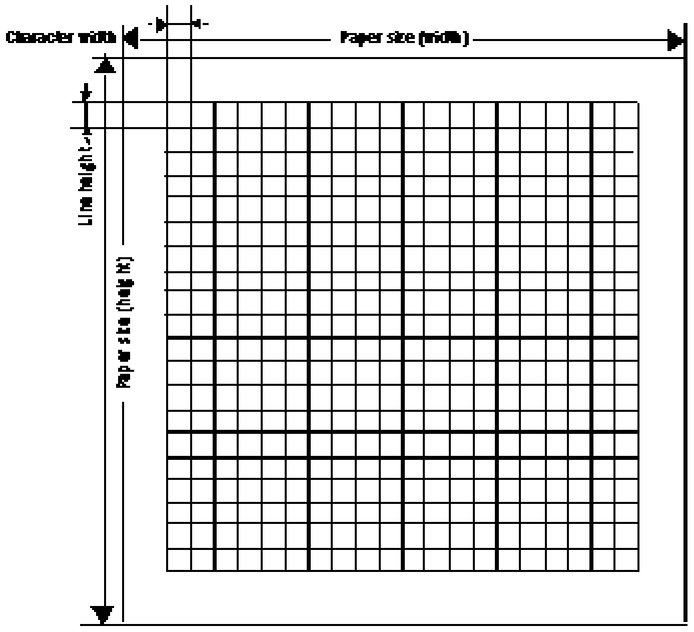
Using DECLARE-LAYOUT

DECLARE-LAYOUT is commonly used in the SETUP section to set the page layout and define paper size and margins.

In the following SETUP section, DECLARE-LAYOUT sets the paper size to 8 1/2 by 11 inches, with all margins at 1 inch.

```
begin-setup
   ! Declare the default layout for this report
   declare-layout default
      paper-size=(8.5,11)
      left-margin=1    right-margin=1
      top-margin=1     bottom-margin=1
   end-declare
end-setup
```

In Production Reporting, data is positioned using line and character position coordinates. Think of the page as a grid where each cell holds one character. With such a grid, in a position qualifier consisting of (*line*,*column*,*width*), *column* and *width* are numbers that denote characters and spaces.



The diagram shows how the main attributes of `DECLARE-LAYOUT` affect the structure of the page. `PAPER-SIZE` defines the page dimensions, including margins. `TOP-MARGIN`, `LEFT-MARGIN`, `BOTTOM-MARGIN`, and `RIGHT-MARGIN` define the margins. In Production Reporting, you cannot print in the margins.

In the sample code, the left margin uses 10 spaces and the top margin uses 6 lines. The page width accommodates 65 characters (without the margins) and 54 lines.

The default mapping of characters and lines to inches is 10 CPI (characters per inch) and 6 LPI (lines per inch). This indicates each character cell is 1/10 inch wide and 1/6 inch high. These settings are used when a program does not contain a `DECLARE-LAYOUT` command.

Overriding Default Settings

You can override the default settings by using the `LINE-HEIGHT` and `CHAR-WIDTH` arguments in `DECLARE-LAYOUT`. These arguments adjust the dimensions of the grid, which implies a change in the meaning of column and line. If a `DECLARE-LAYOUT` paragraph includes `LINE-HEIGHT=1` and `CHAR-WIDTH=1`, the cells in the grid measure 1 point by 1 point (1 point = 1/72 inch or approx. 0.35 mm). In this case, column is a dimension given in points. The length of a string, however, is still given in characters.

You can also use the `MAX-LINES` and `MAX-COLUMNS` arguments in `DECLARE-LAYOUT` to specify the number of lines on the page and the number of characters to fit across the page. Production Reporting calculates line height and character width based on these settings and the size of the page and margins.

Specify coordinates in terms of lines and character positions. The first line from the top is 1 and the first column (from the left) is 1. There is no coordinate 0.

Declaring Page Orientation

DECLARE-LAYOUT allows you to declare the page orientation. This does not affect how Production Reporting uses position coordinates. Line and character positions are not transposed when page orientation is switched. The only effect of ORIENTATION in DECLARE-LAYOUT is to switch the printer to the specified orientation, portrait or landscape. The default mode is portrait.

Part II

Production Reporting Reports

In Production Reporting Reports:

8

Master/Detail Reports

In This Chapter

About Master/Detail Reports

Master/Detail reports show hierarchical information. The information is normally retrieved from multiple tables that have a one-to-many relationship, such as *customers* and *orders*. The customer information is the "master" and the orders are the "detail."

In many cases, you can obtain such information with one Production Reporting SELECT statement. In such a program, the data from the master table is joined with data from the detail table. You can implement break logic as described in Chapter 6, "Break Logic," to group the detail records for each master record.

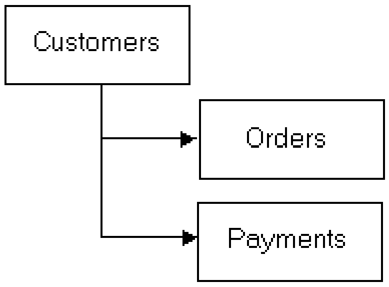
Master/Detail reports have one major disadvantage—if a master record has no associated detail records, it is not displayed. If you must show all master records, whether they have detail records or not, do not use this type of report.

Creating Master/Detail Reports

You can create master/detail reports with one SELECT that retrieves records from the master table, followed by SELECT statements that retrieve the detail records associated with each master record.

In the code example in this chapter, one BEGIN-SELECT returns customer names. For each customer, two additional BEGIN-SELECT commands are executed, one to retrieve order information and another to retrieve payment information.

The following diagram depicts the BEGIN-SELECT structure in this example.



When one query returns master information and another query returns detail information, the detail query is *nested* within the master query.

In our sample program, one query returns customer names and two nested queries return detail information. The nested queries are invoked once for each customer, each one retrieving records that correspond to the current customer. A bind variable correlates the subqueries in the `WHERE` clause. This variable correlates the customer number (*cust_num*) with the current customer record. (See Chapter 17, "Dynamic SQL and Error Checking," for information on bind variables.)

Program ex7a.sqr

```
begin-program
  do main
end-program
begin-procedure main
begin-select
  Print 'Customer Information' (,1)
  Print '-'                    (+1,1,45) Fill
name    (+1,1,25)
city    (,+1,16)
state   (,+1,2)
cust_num
  do cash_receipts(&cust_num)
  do orders(&cust_num)
  position (+2,1)
from customers
end-select
end-procedure ! main
begin-procedure cash_receipts (#cust_num)
  let #any = 0
begin-select
  if not #any
     print 'Cash Received' (+2,10)
     print '-------------' (+1,10)
     let #any = 1
  end-if
date_received    (+1,10,20) edit 'DD-MON-YY'
amount_received  (,+1,13) Edit $$$$,$$0.99
from cash_receipts a
where a.cust_num = #cust_num
end-select
end-procedure ! cash_receipts
begin-procedure orders (#cust_num)
  let #any = 0
```

```
begin-select
  if not #any
    print 'Orders Booked' (+2,10)
    print '-------------' (+1,10)
    let #any = 1
  end-if
a.order_num
order_date                (+1,10,20) Edit 'DD-MON-YY'
description               (,+1,20)
c.price * b.quantity      (,+1,13) Edit $$$$,$$0.99
from  orders a, ordlines b, products c
where a.order_num = b.order_num
  and b.product_code = c.product_code
  and a.cust_num = #cust_num
end-select
end-procedure ! orders
begin-heading 3
 print $current-date (1,1) Edit 'DD-MON-YYYY'
 page-number (1,69) 'Page '
end-heading
```

Correlating Subqueries

Program ex7a.sqr consists of three procedures, *main*, *cash_receipts*, and *orders*, which correspond to the three queries. The procedure *main* is the master. It retrieves the customer names. For each customer, *cash_receipts* lists existing cash receipts, and *orders* lists existing customer orders.

The procedures take the variable *cust_num* as an argument. (See Chapter 18, "Procedures, Argument Passing, and Local Variables .") As you can see, *cash_receipts* and *orders* are called many times, once for each customer. Each time, the procedures perform the same query with another value for the *cust_num* variable in the `WHERE` clause.

Note the use of the `IF` command and the numeric variable #any in these procedures. When the `BEGIN-SELECT` command returns no records, Production Reporting does not execute the following `PRINT` commands. Thus, the headings for these procedures are only displayed for those customers who have records in the detail tables.

The procedure orders demonstrates the use of an expression in the `BEGIN-SELECT`. The expression is `c.price * b.quantity`.

Finally, note that the format given to the dollar amount with the argument `EDIT "$$$$,$$0.99`." This format uses a "floating-to-the-right" money symbol. If fewer digits exist than the six that we allowed here, the dollar sign floats to the right and stays close to the number. (See Chapter 21, "Working with Comma Separated Files (CSV).")

Output for *Program ex7a.sqr*

```
30-JAN-2003                                             Page 1
Customer Information
---------------------------------------------
Joe Smith and Company     Big Falls       NM
         Cash Received
         -------------
         01-JAN-2001              $519.96
         Orders Booked
         -------------
         18-MAR-2001         Widgets                  $55.08
         18-MAR-2001         Curtain rods            $480.96
         18-MAR-2002         Ginger snaps              $7.38
         18-MAR-2002         Modeling clay         $4,136.40
         18-MAR-2002         Hookup wire              $71.82
         27-DEC-2001         Hammers                 $222.50
Customer Information
---------------------------------------------
Corks and Bottles, Inc.   New York         NY
         Cash Received
         -------------
         03-JAN-2001            $1,398.60
         Orders Booked
         -------------
         03-MAR-2001         Thingamajigs          $1,554.00
         18-JUN-2002         Hop scotch kits       $7,764.75
         18-JUN-2002         Wire rings           $15,898.32
         18-JUN-2002         Ginger snaps             $73.80
         22-JAN-2001         Automobile Tires        $372.32
         22-JAN-2001         All Leather Football    $311.08
         22-JAN-2001         Air Deodorizer           $88.38
         22-JAN-2001         3 Ring Binder            $15.00
         22-JAN-2001         300 lb. Weight Set      $481.76
Customer Information
---------------------------------------------
Harry's Landmark Diner    Miningville      IN
         Cash Received
         -------------
         07-JAN-2001            $7,964.17
         12-JAN-2001            $8,629.11
         Orders Booked
         -------------
         01-JAN-2001         Widgets                  $18.36
         01-JAN-2001         Thingamajigs            $220.15
         01-JAN-2001         Curtain rods             $53.44
         01-JAN-2001         Hanging plants           $36.79
         01-JAN-2001         Thimble                   $9.03
         01-JAN-2001         New car               $8,974.87
         01-JAN-2001         Canisters             $3,980.25
         01-JAN-2001         Hop scotch kits       $1,725.50
         01-JAN-2001         Wire rings            $1,987.29
         01-JAN-2001         Ginger snaps              $3.69
         01-JAN-2001         Modeling clay           $172.35
         01-JAN-2001         Hookup wire              $63.84
         02-FEB-2002         Ginger snaps              $3.69
30-JAN-2003                                            Page 2
Customer Information
---------------------------------------------
Jerry's Junkyard Specialt Frogline         NH
         Cash Received
         -------------
         22-JAN-2001           $80,980.65
         Orders Booked
         -------------
```

```
        02-FEB-2001         Curtain rods             $213.76
        02-FEB-2001         New car               $80,773.83
        02-MAY-2002         Thingamajigs             $194.25
        02-MAY-2002         Thimble                   $38.70
        02-MAY-2002         Modeling clay          $4,136.40
        26-NOV-2002         300 lb. Weight Set     $1,324.84
Customer Information
---------------------------------------------
Kate's Out of Date Dress  New York         NY
        Cash Received
        -------------
        11-JAN-2001             $724.71
        Orders Booked
        -------------
        02-FEB-2001         Hanging plants           $735.80
        02-FEB-2001         Thimble                   $25.80
        19-FEB-2002         New car              $143,597.92
        19-FEB-2002         Modeling clay            $172.35
        01-DEC-2000         Widgets                    $4.59
        01-DEC-2000         Thingamajigs              $12.95
        01-DEC-2000         Curtain rods              $26.72
        01-DEC-2000         Hanging plants            $36.79
        01-DEC-2000         Thimble                    $1.29
        01-DEC-2000         New car                $8,974.87
        01-DEC-2000         Whirlybobs                $34.17
        01-DEC-2000         Canisters              $1,326.75
        01-DEC-2000         Hop scotch kits          $862.75
        01-DEC-2000         Wire rings             $1,987.29
        01-DEC-2000         Ginger snaps               $3.69
        01-DEC-2000         Modeling clay             $34.47
        01-DEC-2000         Hookup wire                $3.99
        01-DEC-2000         Binford 4000 Power D      $33.99
        01-DEC-2000         Binford Chain Saw        $173.59
        01-DEC-2000         Shawnee Cross Bow        $219.74
        01-DEC-2000         Big Wheel Bicycle         $77.82
        01-DEC-2000         Office Partitions         $29.95
        01-DEC-2000         Light Bulbs                $2.99
        01-DEC-2000         Automobile Tires          $46.54
        01-DEC-2000         Baseball Cards             $0.35
        01-DEC-2000         All Leather Football      $77.77
        01-DEC-2000         Buckeyes                   $1.40
        01-DEC-2000         Hammers                    $8.9
        01-DEC-2000         Spark Plugs                $3.30
        01-DEC-2000         Air Deodorizer             $9.82
        01-DEC-2000         3 Ring Binder              $0.75
        01-DEC-2000         Laser Printer            $174.65
        01-DEC-2000         White Board               $15.80
        01-DEC-2000         Air Conditioner          $324.76
        01-DEC-2000         300 lb. Weight Set       $120.44
        01-DEC-2000         Reading Light             $23.55
30-JAN-2003                                            Page 3
Customer Information
---------------------------------------------
Sam Johnson               Bell Harbor      MI
        Cash Received
        -------------
        13-FEB-2001             $136.75
```

```
         Orders Booked
         -------------
         19-FEB-2001         Ginger snaps              $36.90
         19-FEB-2001         Hookup wire               $59.85
         19-MAY-2002         Hop scotch kits        $6,902.00
         19-MAY-2002         Ginger snaps              $44.28
         19-MAY-2002         Modeling clay            $344.70
         19-MAY-2002         Hookup wire               $59.85
         19-JUN-2002         Binford 4000 Power D     $339.90
         19-JUN-2002         Binford Chain Saw        $867.95
         19-JUN-2002         Shawnee Cross Bow        $439.48
         01-NOV-2000         Big Wheel Bicycle        $389.10
         01-NOV-2000         Light Bulbs            $1,659.45
Customer Information
---------------------------------------------
Harriet Bailey           Mamaroneck       NY
         Cash Received
         -------------
         14-JAN-2001           $1,386.14
         Orders Booked
         -------------
         18-JUN-2001         Thingamajigs           $1,554.00
         18-JUN-2001         Curtain rods              $53.44
         18-JUN-2001         Hanging plants            $36.79
         03-JUN-2002         Wire rings            $39,745.80
         03-JUN-2002         Ginger snaps              $36.90
         03-JUN-2002         Hookup wire               $59.85
         25-AUG-2001         New car                $8,974.87
         25-AUG-2001         Binford 4000 Power D      $67.98
         26-SEP-2002         Hanging plants         $4,083.69
         16-FEB-2001         Baseball Cards             $7.00
         16-FEB-2001         All Leather Football     $622.16
         16-FEB-2001         Air Conditioner          $649.52
         16-FEB-2001         300 lb. Weight Set       $481.76
30-JAN-2003                                             Page 4
Customer Information
---------------------------------------------
Clair Butterfield        Teaneck          NJ
         Cash Received
         -------------
         15-JAN-2001           $3,812.79
         Orders Booked
         -------------
         03-JUN-2001         Modeling clay          $4,136.40
         02-MAY-2002         Canisters              $3,980.25
         02-MAY-2002         Wire rings            $13,911.03
         02-MAY-2002         Modeling clay             $34.47
         20-JAN-2001         New car                $8,974.87
         20-JAN-2001         Binford 4000 Power D      $67.98
         22-JUN-2002         Canisters             $45,109.50
Customer Information
---------------------------------------------
Quentin Fields           Cleveland        OH
         Cash Received
         -------------
         17-JAN-2001           $8,994.48
         Orders Booked
         -------------
         01-APR-2001         Widgets                 $68.85
         01-APR-2001         Thimble                 $38.70
         01-APR-2001         New car              $8,974.87
         01-JAN-2002         Hop scotch kits      $3,451.00
         20-JAN-2002         Widgets              $5,159.16
         20-JAN-2002         Thingamajigs        $32,983.65
         20-JAN-2002         Thimble                  $1.29
         07-MAR-2001         Baseball Cards           $3.50
         07-MAR-2001         3 Ring Binder            $7.50
         07-MAR-2001         Air Conditioner        $649.52
         07-MAR-2001         300 lb. Weight Set     $602.20
Customer Information
---------------------------------------------
Eliot Richards           Queens          NY
         Cash Received
         -------------
         18-JAN-2001           $6,221.39
         Orders Booked
         -------------
         02-MAY-2001         Whirlybobs             $239.19
         02-MAY-2001         Canisters            $3,980.25
         02-FEB-2002         Widgets                  $4.59
         02-FEB-2002         New car             $26,924.61
         02-FEB-2002         Modeling clay           $68.94
         02-FEB-2002         Hookup wire              $3.99
         29-AUG-2002         Big Wheel Bicycle      $155.64
         29-AUG-2002         Office Partitions      $209.65
         29-AUG-2002         Light Bulbs             $29.90
         29-AUG-2002         Automobile Tires       $744.64
         30-MAR-2002         White Board          $1,106.00
30-JAN-2003                                             Page 5
         30-MAR-2002         Air Conditioner        $324.76
         07-JUL-2001         New car              $8,974.87
         07-JUL-2001         Hop scotch kits        $862.75
Customer Information
---------------------------------------------
Isaiah J Schwartz and Com Zanesville     OH
         Cash Received
         -------------
         01-MAR-2001          $26,143.27
         Orders Booked
         -------------
         02-MAY-2001         Hop scotch kits      $6,902.00
         02-MAY-2001         Wire rings          $19,872.90
         03-MAR-2002         Thingamajigs            $90.65
         03-MAR-2002         Curtain rods            $26.72
         03-MAR-2002         Thimble                 $21.93
         26-OCT-2002         Hop scotch kits      $6,039.25
         25-NOV-2001         Binford 4000 Power D   $407.88
         25-NOV-2001         Binford Chain Saw    $2,603.85
Customer Information
---------------------------------------------
Harold Alexander Fink     Davenport       IN
         Cash Received
         -------------
         07-MAY-2001              $593.70
         Orders Booked
         -------------
         19-MAY-2001         Widgets                    $55.08
         19-MAY-2001         Ginger snaps               $44.28
         19-MAY-2001         Modeling clay             $517.05
         01-APR-2002         Thingamajigs            $1,554.00
         01-APR-2002         Curtain rods               $53.44
         01-APR-2002         Hanging plants             $36.79
         06-OCT-2002         Hanging plants            $551.85
         06-OCT-2002         Wire rings             $11,923.74
         06-OCT-2002         Modeling clay              $68.94
         06-OCT-2002         All Leather Football      $233.31
         11-NOV-2002         Whirlybobs                $580.89
         11-NOV-2002         Canisters              $18,574.50
         11-NOV-2002         Hookup wire                $19.95
Customer Information
---------------------------------------------
Gregory Stonehaven        Everretsville    OH
         Orders Booked
         -------------
         07-MAR-2002         Office Partitions       $1,707.15
         07-MAR-2002         White Board               $395.00
         30-AUG-2002         Wire rings             $19,872.90
         30-AUG-2002         Hookup wire                $19.95
30-JAN-2003                                                Page 6
Customer Information
---------------------------------------------
Alfred E Newman & Company New York         NY
         Orders Booked
         -------------
         30-JUL-2002         Air Deodorizer             $98.20
         30-JUL-2002         Laser Printer             $523.95
         06-OCT-2001         Light Bulbs             $3,289.00
Customer Information
---------------------------------------------
Hammerhead Hardware       Fort Wayne       IN
         Orders Booked
         -------------
         02-APR-2002         Hammers                   $222.50
         02-APR-2002         Spark Plugs               $264.00
         02-APR-2002         Air Conditioner           $324.76
         30-SEP-2001         Ginger snaps              $369.00
Customer Information
---------------------------------------------
Lights R Us               Pittsburgh       PA
         Orders Booked
         -------------
         22-APR-2002         All Leather Football      $777.70
         22-APR-2002         Buckeyes                  $210.00
         22-APR-2002         300 lb. Weight Set        $240.88
Customer Information
---------------------------------------------
Office Building Contracto Dover            De
         Orders Booked
         -------------
         29-DEC-2002         Baseball Cards             $70.00
         29-DEC-2002         3 Ring Binder              $37.50
```

9

Cross-Tabular Reports

In This Chapter

About Cross-Tabular Reports

Cross-tabular reports are matrix- or spreadsheet-like reports useful for presenting summary numeric data. Cross-tabular reports vary in format. The following example shows sales revenue summarized by product by sales channel.

```
           Revenue by product by sales channel
Product    Direct Sales Resellers  Mail Order   Total
---------- ------------ --------- ----------- -------
A                $2,100    $1,209          $0  $3,309
B                  $120      $311        $519    $950
C                    $2        $0        $924    $926
---------- ------------ --------- ----------- -------
Total            $2,222    $1,520      $1,443  $5,185
```

This report is based on many sales records. The three middle columns correspond to sales channel categories. Each row corresponds to a product. The records fall into nine groups: three products sold through three sales channels. Some groups have no sales (such as mail order for Product A).

Each category can be a discrete value of some database column or a set of values. For example, *Resellers* can be domestic resellers plus international distributors.

A category can also represent a range, as demonstrated here.

```
              Orders by product by order Size
Product
Category     Less than 10 10 to 100 More than 100   Total
----------- ------------ --------- ------------- -------
Durable              200       120             0     320
Nondurable           122       311           924    1876
----------- ------------ --------- ------------- -------
Total                322       431          1443    2196
```

In this example, the rows correspond to nondescript categories. Products are classified as durable or nondurable. The columns represent ranges of order size.

For each record selected, the program must determine the range to which it belongs and add 1 to the count for that category. The numbers in the cells are counts, but they could be sums, averages, or any other expression.

Cross-tabular reports become more complex when the column number is not predefined and when more columns exist than can fit across the page.

Arrays

In many cases, the program must process all records before it can begin printing the data. During processing, the program needs to keep the data in some buffer where it can accumulate the numbers. This can be done in an Production Reporting array.

An array is a unit of storage that consists of rows and columns and exists in memory. An array is similar to a database table, but it exists only in memory.

Program ex8a.sqr specifies an array called *order_qty* to hold the sum of the quantity of orders in a given month. This example could be programmed without an array, however, using one can be beneficial. Data retrieved once and stored in an array can be presented in many ways without additional database queries. The data can even be presented in a chart, as shown later in Chapter 14, “Business Charts.”

This example demonstrates a “three-dimensional array.” This type of array has fields (columns) and rows, and it also has repeating fields (the *third dimension*). In the *order_qty* array, the first field is the product description. The second field is the order quantity of each month. Three months exist in the example; therefore, this field repeats three times.

Production Reporting references arrays in expressions such as *array_name.field(sub1[,sub2])*. *Sub1* is the first subscript, the row number. The row count starts with zero. The second subscript (*sub2*) is specified when the field repeats. Repeating fields are also numbered starting with zero. The subscript can be a literal or an Production Reporting numeric variable.

Program ex8a.sqr

```
#define max_products 100
begin-setup
  create-array
     name=order_qty       size={max_products}
     field=product:char   field=month_qty:number:3
end-setup
begin-program
  do select_data
  do print_array
end-program
begin-procedure print_array
  let #entry_cnt = #i
  let #i = 0
  while #i <= #entry_cnt
      let $product  = order_qty.product(#i)
      let #jan      = order_qty.month_qty(#i,0)
      let #feb      = order_qty.month_qty(#i,1)
      let #mar      = order_qty.month_qty(#i,2)
      let #prod_tot = #jan + #feb + #mar
      print $product  (,1,30)
      print #jan      (,32,9) edit 9,999,999
      print #feb      (,42,9) edit 9,999,999
      print #mar      (,52,9) edit 9,999,999
      print #prod_tot (,62,9) edit 9,999,999
      position (+1)
      let #jan_total = #jan_total + #jan
      let #feb_total = #feb_total + #feb
      let #mar_total = #mar_total + #mar
      let #i = #i + 1
  end-while
  let #grand_total = #jan_total + #feb_total + #mar_total
  print 'Totals'     (+2,1)
  print #jan_total   (,32,9) edit 9,999,999
  print #feb_total   (,42,9) edit 9,999,999
  print #mar_total   (,52,9) edit 9,999,999
  print #grand_total (,62,9) edit 9,999,999
end-procedure print_array
begin-procedure select_data
begin-select
order_date
! The quantity for this order
quantity
! the product for this order
description
  if #i = 0 and order_qty.product(#i) = ''
      let order_qty.product(#i) = &description
  end-if
  if order_qty.product(#i) != &description
      let #i = #i + 1
      if #i >= {max_products}
        display 'Error: There are more than {max_products} products'
         stop
      end-if
      let order_qty.product(#i) = &description
  end-if
  let #j = to_number(datetostr(&order_date,'MM')) - 1
  if #j < 3
      let order_qty.month_qty(#i,#j) =
               order_qty.month_qty(#i,#j) + &quantity
  end-if
from  orders a, ordlines b, products c
where a.order_num = b.order_num
and   b.product_code = c.product_code
order by description
end-select
end-procedure ! select_data
begin-heading 4
  print $current-date (1,1)
  print 'Order Quantity by Product by Month' (1,18)
  page-number (1,64) 'Page '
  print 'Product'   (3,1)
  print '  January' (,32)
  print ' February' (,42)
  print '    March' (,52)
  print '    Total' (,62)
  print '-'         (4,1,70) Fill
end-heading
```

Output for *Program ex8a.sqr*

```
11-JUN-01                    Order Quantity by Product by Month        Page
1
Product                     January   February     March     Total
---------------------------------------------------------------
Canisters                         3          0         0         3
Curtain rods                      2          8        18        28
Ginger snaps                      1         10         0        11
Hanging plants                    1         20         0        21
Hookup wire                      16         15         0        31
Hop scotch kits                   2          0         0         2
Modeling clay                     5          0         0         5
New car                           1          9         0        10
Thimble                           7         20         0        27
Thingamajigs                     17          0       120       137
Widgets                           4          0        12        16
Wire rings                        1          0         0         1
Totals                           60         82       150       292
```

Creating Arrays

You must define the size of an array when you create it. The program creates the array *order_qty* with a size of 100.

`#DEFINE MAX_PRODUCTS 100` defines the constant *max_products* as a substitution variable. The sample program uses this constant to define the size of the array. It is good practice to use `#DEFINE` because it displays the limit at the top of the program source.

The `SETUP` section creates the array using the `CREATE-ARRAY` command. All Production Reporting arrays are created before program execution. Their size must be known at compile time. If you do not know the exact number of rows, over-allocate and specify an upper bound. In the example, the array has 100 rows, even though the program only uses 12 rows to process the sample data.

The preceding program has two procedures: *select_data* and *print_array*. *Select_data* performs the database query, as its name suggests. While the database records are processed, nothing prints, and the data accumulates in the array. When processing completes, *print_array* loops through the array and prints the data and adds the month totals and prints them at the bottom.

The report summarizes the product order quantities for each month, which are the records ordered by the product description. The procedure then fills the array one product at a time. For each record selected, the procedure checks to see if it is a new product; if it is, the array is incremented by row subscript #i. The procedure also adds the quantity to the corresponding entry in the array based on the month.

To obtain the month in this program, use the `datetostr` function as follows:

```
let #j = to_number(datetostr(&order_date, 'MM')) - 1
```

This converts the *order_date* column into a string. (The 'MM' edit mask specifies that only the month part be converted.) The resulting string is then converted to a number; if it is less than 3, it represents January, February, or March, and is added to the array.

Grouping by Category

Program ex8b.sqr is a cross-tabular report that groups the products by price range. This grouping cannot be done using an SQL GROUP BY clause. To process the records in order of price category, the program would have to sort the table by price. The example shows how to do it without sorting the data.

The program uses the EVALUATE command to determine the price category and assign the array subscript *#i* to 0, 1, or 2. Then it adds the order quantity to the array cell that corresponds to the price category (row) and the month (column).

Program ex8b.sqr

```
#define max_categories 3
begin-setup
  create-array
     name=order_qty       size={max_categories}
     field=category:char  field=month_qty:number:3
end-setup
begin-program
  do select_data
  do print_array
end-program
begin-procedure print_array
  let #i = 0
  while #i < {max_categories}
      let $category = order_qty.category(#i)
      let #jan      = order_qty.month_qty(#i,0)
      let #feb      = order_qty.month_qty(#i,1)
      let #mar      = order_qty.month_qty(#i,2)
      let #category_tot = #jan + #feb + #mar
      print $category     (,1,31)
      print #jan          (,32,9) edit 9,999,999
      print #feb          (,42,9) edit 9,999,999
      print #mar          (,52,9) edit 9,999,999
      print #category_tot (,62,9) edit 9,999,999
      position (+1)
      let #jan_total = #jan_total + #jan
      let #feb_total = #feb_total + #feb
      let #mar_total = #mar_total + #mar
      let #i = #i + 1
  end-while
  let #grand_total = #jan_total + #feb_total + #mar_total
  print 'Totals'      (+2,1)
  print #jan_total    (,32,9) edit 9,999,999
  print #feb_total    (,42,9) edit 9,999,999
  print #mar_total    (,52,9) edit 9,999,999
  print #grand_total (,62,9) edit 9,999,999
end-procedure print_array
```

```
begin-procedure select_data
  let order_qty.category(0) = '$0-$4.99'
  let order_qty.category(1) = '$5.00-$100.00'
  let order_qty.category(2) = 'Over $100'
begin-select
order_date
! the price / price category for the order
c.price &price
  move &price to #price_num
  evaluate #price_num
  when < 5.0
     let #i = 0
     break
  when <= 100.0
     let #i = 1
     break
  when-other
     let #i = 2
     break
  end-evaluate
! The quantity for this order
quantity
  let #j = to_number(datetostr(&order_date,'MM')) - 1
  if #j < 3
      let order_qty.month_qty(#i,#j) =
               order_qty.month_qty(#i,#j) + &quantity
  end-if
from  orders a, ordlines b, products c
where a.order_num = b.order_num
and   b.product_code = c.product_code
end-select
end-procedure ! select_data
begin-heading 5
  print $current-date (1,1)
  page-number (1,64) 'Page '
  print 'Order Quantity by Product Price Category by Month' (2,11)
  print 'Product Price Category'   (4,1)
  print '  January' (,32)
  print ' February' (,42)
  print '    March' (,52)
  print '    Total' (,62)
  print '-'         (5,1,70) Fill
end-heading
```

Output for *Program ex8b.sqr*

```
11-JUN-01
Page 1
          Order Quantity by Product Price Category by Month
Product Price Category        January  February   March     Total
----------------------------------------------------------------------
$0-$4.99                         28       45       12        85
$5.00-$100.00                    25       28      138       191
Over $100                         7        9        0        16
Totals                          60        82       150       292
```

Using Multiple Arrays

Using arrays to buffer data offers several advantages. In the last example, it eliminated the need to sort the data. Another advantage is that you can combine the two sample reports. With one pass on the data, you can fill the two arrays and then print the two parts of the report.

Program ex8c.sqr performs the work done by the first two programs. The SETUP section specifies two arrays—one to summarize monthly orders by product, and another to summarize monthly orders by price range.

Program ex8c.sqr

```
#define max_categories 3
#define max_products 100
begin-setup
  create-array
     name=order_qty       size={max_products}
     field=product:char   field=month_qty:number:3
  create-array
     name=order_qty2      size={max_categories}
     field=category:char  field=month_qty:number:3
end-setup
begin-program
  do select_data
  do print_array
  print '-' (+2,1,70) fill
  position (+1)
  do print_array2
end-program
begin-procedure print_array
  let #entry_cnt = #i
  let #i = 0
  while #i <= #entry_cnt
      let $product  = order_qty.product(#i)
      let #jan      = order_qty.month_qty(#i,0)
      let #feb      = order_qty.month_qty(#i,1)
      let #mar      = order_qty.month_qty(#i,2)
      let #prod_tot = #jan + #feb + #mar
      print $product  (,1,30)
      print #jan      (,32,9) edit 9,999,999
      print #feb      (,42,9) edit 9,999,999
      print #mar      (,52,9) edit 9,999,999
      print #prod_tot (,62,9) edit 9,999,999
      position (+1)
      let #i = #i + 1
  end-while
end-procedure ! print_array
begin-procedure print_array2
  let #i = 0
  while #i < {max_categories}
      let $category = order_qty2.category(#i)
      let #jan      = order_qty2.month_qty(#i,0)
      let #feb      = order_qty2.month_qty(#i,1)
      let #mar      = order_qty2.month_qty(#i,2)
      let #category_tot = #jan + #feb + #mar
      print $category     (,1,31)
      print #jan          (,32,9) edit 9,999,999
      print #feb          (,42,9) edit 9,999,999
      print #mar          (,52,9) edit 9,999,999
      print #category_tot (,62,9) edit 9,999,999
      position (+1)
      let #jan_total = #jan_total + #jan
      let #feb_total = #feb_total + #feb
      let #mar_total = #mar_total + #mar
      let #i = #i + 1
  end-while
  let #grand_total = #jan_total + #feb_total + #mar_total
  print 'Totals'     (+2,1)
  print #jan_total   (,32,9) edit 9,999,999
  print #feb_total   (,42,9) edit 9,999,999
  print #mar_total   (,52,9) edit 9,999,999
  print #grand_total (,62,9) edit 9,999,999
end-procedure ! print_array2
begin-procedure select_data
  let order_qty2.category(0)='$0-$4.99'
  let order_qty2.category(1)='$5.00-$100.00'
  let order_qty2.category(2)='Over $100'
begin-select
order_date
! the price / price category for the order
c.price &price
  move &price to #price_num
  evaluate #price_num
  when < 5.0
     let #x = 0
     break
  when <= 100.0
     let #x = 1
     break
  when-other
     let #x = 2
     break
  end-evaluate
! The quantity for this order
quantity
  let #j = to_number(datetostr(&order_date,'MM')) - 1
  if #j < 3
      let order_qty2.month_qty(#x,#j) =
               order_qty2.month_qty(#x,#j) + &quantity
  end-if
! the product for this order
description
  if #i = 0 and order_qty.product(#i) = ''
      let order_qty.product(#i) = &description
  end-if
  if order_qty.product(#i) != &description
      let #i = #i + 1
      if #i >= {max_products}
         display 'Error: There are more than {max_products} products'
         stop
      end-if
      let order_qty.product(#i) = &description
  end-if
  if #j < 3
      let order_qty.month_qty(#i,#j) =
                order_qty.month_qty(#i,#j) + &quantity
  end-if
from  orders a, ordlines b, products c
where a.order_num = b.order_num
and   b.product_code = c.product_code
order by description
end-select
end-procedure ! select_data
begin-heading 5
print $current-date (1,1)
   page-number (1,64) 'Page '
   print 'Order Quantity by Product and Price Category by Month' (2,10)
   print 'Product / Price Category'   (4,1)
   print '  January' (,32)
   print ' February' (,42)
   print '    March' (,52)
   print '    Total' (,62)
   print '-'         (5,1,70) Fill
end-heading
```

Output for *Program ex8c.sqr*

```
11-JUN-01                                                    Page 1
         Order Quantity by Product and Price Category by Month
Product / Price Category         January  February     March     Total
----------------------------------------------------------------------
Canisters                              3         0         0         3
Curtain rods                           2         8        18        28
Ginger snaps                           1        10         0        11
Hanging plants                         1        20         0        21
Hookup wire                           16        15         0        31
Hop scotch kits                        2         0         0         2
Modeling clay                          5         0         0         5
New car                                1         9         0        10
Thimble                                7        20         0        27
Thingamajigs                          17         0       120       137
Widgets                                4         0        12        16
Wire rings                             1         0         0         1
----------------------------------------------------------------------
$0-$4.99                              28        45        12        85
$5.00-$100.00                         25        28       138       191
Over $100                              7         9         0        16
```

Production Reporting arrays are also advantageous in programs that produce charts. With the data for the chart in the array, presenting this cross-tab as a bar chart is easy. (See Chapter 14, "Business Charts.")

10

Printing Mailing Labels

In This Chapter

Defining Columns and Rows

To print labels in multiple columns, use the `COLUMNS`, `NEXT-COLUMN`, and `NEXT-LISTING` commands.

Program ex9a.sqr prints mailing labels in a format of three columns by ten rows. It also counts the number of labels printed and prints that number on the last sheet of the report.

Program ex9a.sqr

```
#define MAX_LABEL_LINES        10
#define LINES_BETWEEN_LABELS   3
begin-setup
  declare-layout default
    paper-size=(10,11)   left-margin=0.33
  end-declare
end-setup
begin-program
  do mailing_labels
end-program
begin-procedure mailing_labels
  let #label_count = 0
  let #label_lines = 0
  columns 1 29 57  ! enable columns
  alter-printer font=5 point-size=10
begin-select
name       (1,1,30)
addr1      (2,1,30)
city
state
zip
  move &zip to $zip XXXXX-XXXX
  let $last_line = &city || ', ' || &state || ' ' || $zip
  print $last_line (3,1,30)
  next-column at-end=newline
  add 1 to #label_count
  if #current-column = 1
    add 1 to #label_lines
    if #label_lines = {MAX_LABEL_LINES}
      new-page
      let #label_lines = 0
    else
      next-listing no-advance skiplines={LINES_BETWEEN_LABELS}
    end-if
  end-if
from customers
end-select
  use-column 0  ! disable columns
  new-page
  print 'Labels printed on ' (,1)
  print $current-date ()
  print 'Total labels printed = ' (+1,1)
  print #label_count () edit 9,999,999
end-procedure ! mailing_labels
```

`COLUMNS 1 29 57` defines the starting position for three columns. The first column starts at character position 1, the second at character position 29, and the third at character position 57.

The program writes the first address into the first column, the second address into the second, the third address into the third. The fourth address is written into the second row of the first column, just below the first label. When ten lines of labels are complete, a new page starts. After the last page of labels are printed, the program prints a summary page showing the number of labels printed.

To print the last line of the label, the *city*, *state*, and *zip* columns are moved to string variables. The command

```
LET $last_line = &city || ', ' || &state || ' ' || $zip
```

combines the city, state, and zip code, plus appropriate punctuation and spacing, into a string, which it stores in the variable *$last_line*. In this way, city, state, and zip code are printed without unnecessary gaps.

The program defines two counters, *#label_count* and *#label_lines*. The first counter, *#label_count*, counts the total number of labels and prints it on the summary page. The second counter, *#label_lines*, counts the number of rows of labels that were printed. When the program has printed the number of lines defined by `{MAX_LABEL_LINES}`, it starts a new page and resets the *#label_lines* counter.

After each row of labels, `NEXT-LISTING` redefines the print position for the next row of labels as line 1. `NEXT-LISTING` skips the specified number of lines (`SKIPLINES`) from the last line printed (`NO-ADVANCE`) and sets the new position as line 1.

`ALTER-PRINTER` changes the font in which the report is printed. (See Chapter 15, “Changing Fonts.”)

The sample program prints the labels in 10-point Times Roman, which is a proportionally spaced font. In Windows, you can use proportionally spaced fonts with any printer that supports fonts or graphics. On other platforms, Production Reporting directly supports HP LaserJet printers and PostScript printers.

Printing and printer support are explained in greater detail in Chapter 30, “Printing Issues.” For information on using proportionally spaced fonts, see Chapter 15, “Changing Fonts.”

In the sample program, DECLARE-LAYOUT defines a page width of 10 inches. This width accommodates the printing of the third column, which contains 30 characters and begins at character position 57. Production Reporting assumes a default character grid of 10 characters per inch, which would cause the third column to print beyond the paper edge if this report used the default font. The 10-point Times Roman used here, however, condenses the text so that it fits on the page. The page width is set at 10 inches to prevent Production Reporting from treating the third-column print position as an error.

Running Programs

Proportionally-spaced fonts require different techniques for running the program and viewing the output. For UNIX platforms, specify the printer type with the -PRINTER:xx flag. For HP LaserJet printers, enter -PRINTER:HP on the command line. For PostScript printers, enter -PRINTER:PS .

For example:

```
sqr ex9a username/password  -printer:hp
```

Use -KEEP to produce output in the SQR Portable File format (SPF) and print it using Production Reporting Print. You still must use -PRINTER:xx when printing. SQR Portable File format is covered in greater detail in Chapter 30, “Printing Issues.”

With SQR Production Reporting Studio , neither -PRINTER:xx nor -KEEP is required. The output automatically appears in the Viewer window after the report is run. Here is a portion of the output.

Output for *Program ex9a.sqr*

```
Gregory Stonehaven              Alfred E Newman & Company       Eliot Richards
Middlebrook Road                2837 East Third Street          2134 Partridge Ave
Everretsville, OH 40233-1000    New York, NY 10002-1001         Queens, NY 10213-1002


Isaiah J Schwartz and Company   Harold Alexander Fink           Harriet Bailey
37211 Columbia Blvd             32077 Cedar Street              47 Season Street
Zanesville, OH 44900-1300       Davenport, IN 62130-1025        Mamaroneck, NY 10833-1660


Clair Butterfield               Quentin Fields                  Jerry's Junkyard Specialties
371 Youngstown Blvd             37021 Cedar Road                Crazy Lakes Cottages
Teaneck, NJ 00355-4530          Cleveland, OH 44121-9475        Frogline, NH 04821-9876


Kate's Out of Date Dress Shop   Sam Johnson                     Joe Smith and Company
2100 Park Ave                   37 Cleaver Street               1711 Sunset Blvd
New York, NY 10134-2030         Bell Harbor, MI 40674-3900      Big Falls, NM 87893-7070


Corks and Bottles, Inc.         Harry's Landmark Diner          Hammerhead Hardware
167 East Blvd.                  17043 Silverfish Road           1942 Finnishing Road
New York, NY 10204-1234         Miningville, IN 40622-4321      Fort Wayne, IN 40622-4321


Lights R Us                     Office Building Contractors
5454 Enlighten Way              3981 Office Park Blvd
Pittsburgh, PA 90672-4168       Dover, De 20652-9747
```

The report produces the output in three columns corresponding to the dimensions of a sheet of mailing label stock. In the preceding example, the report prints the labels left to right, filling each row of labels before moving down the page.

You can also print the labels from the top down, filling each column before moving to the next column of labels. The code is shown here. The differences between this code and the previous one are shown in bold. The output is not printed here, but you can run the file and view it using the same procedure you used for the previous example.

Modified Program ex9a.sqr

```
#define MAX_LABEL_LINES        10
#define LINES_BETWEEN_LABELS   3
begin-setup
  declare-layout default
    paper-size=(10,11)   left-margin=0.33
  end-declare
end-setup
begin-program
  do mailing_labels
end-program
begin-procedure mailing_labels
  let #Label_Count = 0
  let #Label_Lines = 0
  columns 1 29 57  ! enable columns
alter-printer font=5 point-size=10
begin-select
name       (0,1,30)
addr1      (+1,1,30)
```

```
city
state
zip
  move &zip to $zip xxxxx-xxxx
  let $last_line = &city || ', ' || &state || ' ' || $zip
  print $last_line (+1,1,30)
  add 1 to #label_count
  add 1 to #label_lines
  if #label_lines = {MAX_LABEL_LINES}
    next-column goto-top=1 at-end=newpage
    let #label_lines = 0
  else
    position (+1)
    position (+{LINES_BETWEEN_LABELS})
  end-if
from customers
end-select
  use-column 0  ! disable columns
  new-page
  print 'Labels printed on ' (,1)
  print $current-date ()
  print 'Total labels printed = ' (+1,1)
  print #label_count () edit 9,999,999
end-procedure ! mailing_labels
```

11 Creating Form Letters

In This Chapter

Using DOCUMENT Sections

The DOCUMENT sections starts with BEGIN-DOCUMENT and ends with END-DOCUMENT. Between these commands are the form letter and the variables. Production Reporting inserts variable values when the document prints. To leave blank lines in a letter, explicitly mark them with a.*b*.

Using Document Markers

Document markers are placeholders in a DOCUMENT section where you can print data after the DOCUMENT section. Document markers are denoted with a name preceded by the at sign (@).

Production Reporting prints variable contents in the position where it is placed in the DOCUMENT section. For example, in the program that follows, the customer's name is printed on the first line.

Using document markers provides flexibility in positioning variable contents. The sample program uses a document marker to position the city, state, and zip code because the city name varies in length and thus affects the position of the state name and zip code.

A Simple Form Letter Program

Program ex 10a.sqr demonstrates the use of document markers in a simple form letter. First, Production Reporting performs the *main* procedure and the SELECT statement. Next, it performs the *write_letter* procedure and the DOCUMENT section. POSITION sets the position to the appropriate line, which is given by *@city_state_zip*. The program prints the city, then continues printing the other elements to the current position. The state name, and zip code automatically print in the correct positions with appropriate punctuation.

Program ex 10a.sqr

```
begin-program
  do main
end-program
begin-procedure main
begin-select
name
addr1
addr2
city
state
zip
  do write_letter
from customers
order by name
end-select
end-procedure ! main
begin-procedure write_letter
begin-document (1,1)
&name
&addr1
&addr2
@city_state_zip
.b
.b
                                                $current-date
Dear Sir or Madam:
.b
     Thank you for your recent purchases from ACME Inc. We would like to
tell you about our limited-time offer. During this month, our entire
inventory is marked down by 25%. Yes, you can buy your favorite merchandise
and save too.
     To place an order simply dial 800-555-ACME. Delivery is free too, so
don't wait.
.b
.b
                                Sincerely,
                                Clark Axelotle
                                ACME Inc.
end-document
position ()  @city_state_zip
print &city  ()
print ', '   ()
print &state ()
print ' '    ()
print &zip   () edit xxxxx-xxxx
new-page
end-procedure ! write_letter
```

Output for *Program ex 10a.sqr*

```
Alfred E Newman & Company
2837 East Third Street
Greenwich Village
New York, NY 10002-1001
                                                10-MAY-2001
```

```
Dear Sir or Madam:
     Thank you for your recent purchases from ACME Inc. We would like to
tell you about our limited-time offer. During this month, our entire
inventory is marked down by 25%. Yes, you can buy your favorite merchandise
and save too. To place an order simply dial 800-555-ACME. Delivery is free
too, so don't wait.
                                Sincerely,
                                Clark Axelotle
                                ACME Inc.
```

Note:

For another example of a form letter, see Chapter 13, “Using Graphics.”

12 Exporting Data to Other Applications

This chapter describes how to create tab-delimited files for exporting data to many applications.

Program ex11a.sqr creates a file that you can load into a document such as a spreadsheet. The tabs create columns in a spreadsheet or word processing document that correspond to the columns in a database table.

Program ex11a.sqr

```
begin-setup
 ! No margins, wide enough for the widest record
 ! and no page breaks
 declare-layout default
   left-margin=0    top-margin=0
   max_columns=160  formfeed=no
 end-declare
end-setup
begin-program
 do main
end-program
begin-procedure main
encode '<009>' into $sep  ! Separator character is TAB
let $cust_num = 'Customer Number'
let $name = 'Customer Name'
let $addr1 = 'Address Line 1'
let $addr2 = 'Address Line 2'
let $city = 'City'
let $state = 'State'
let $zip = 'Zip Code'
let $phone = 'Phone Number'
let $tot = 'Total'
string $cust_num $name $addr1 $addr2
       $city $state $zip $phone $tot by $sep into $col_hds
print $col_hds (1,1)
new-page
begin-select
cust_num
name
addr1
addr2
city
state
zip
phone
tot
  string &cust_num &name &addr1 &addr2
```

```
         &city &state &zip &phone &tot by $sep into $db_cols
  print $db_cols ()
  new-page
from customers
end-select
end-procedure ! main
```

ENCODE stores the ASCII code for the tab character in the variable $sep. The code (9) is enclosed in angle brackets to indicate that it is a non-display character.Production Reporting treats it as a character code and sets the variable accordingly. ENCODE is a useful way to place non-alpha and non-numeric characters into variables.

LET creates variables for text strings used as column headings in export files. STRING combines these variables in $col_hds, with each heading separated by a tab.

In the SELECT paragraph, STRING combines the records (named as column variables) in $db_cols, with each record separated by a tab.

NEW-PAGE causes a new line and carriage return at the end of each record, with the line number reset to 1. The page is not ejected because of the FORMFEED=NO argument in DECLARE-LAYOUT. (Remember, this report is meant to be exported, not printed.)

You can load the output file ex11a.lis into a spreadsheet or other application.

Part III

Fonts and Graphics

In Fonts and Graphics:

13 Using Graphics

In This Chapter

A Simple Tabular Report

Program ex12a.sqr produces a simple tabular report similar to the one in Chapter 4, “Selecting Data.”

Program ex12a.sqr

```
begin-setup
 declare-layout default
 end-declare
end-setup
begin-program
 do main
end-program
begin-procedure main
begin-select
name  (,1,30)
city  (,+1,16)
state (,+1,5)
tot   (,+1,11) edit 99999999.99
 next-listing no-advance need=1
 let #grand_total = #grand_total + &tot
from customers
end-select
print '-' (,55,11) fill
print 'Grand Total' (+1,40)
print #grand_total (,55,11) edit 99999999.99
end-procedure ! main
begin-heading 5
 print $current-date (1,1) Edit 'DD-MON-YYYY'
 page-number (1,60) 'Page '
 print 'Name'   (3,1)
 print 'City'   (,32)
 print 'State'  (,49)
 print 'Total'  (,61)
 print '-'      (4,1,65) fill
end-heading
```

The DECLARE-LAYOUT command in the SETUP section specifies the default layout without defining any options. The purpose of specifying the default layout is to use its margin settings, which are defined as 1/2 inch. Without DECLARE-LAYOUT, the report would have no margins.

The FILL option in the PRINT command produces dashed lines, which is a simple way to draw lines for a report printed on a line printer. On a graphical printer, however, it is possible to draw solid lines. (See "Adding Graphics" on page 86.)

Output for *Program ex12a.sqr*

```
    06-JUN-01                                                     Page 1

     Name                           City              State     Total
     -------------------------------------------------------------------

     Gregory Stonehaven             Everretsville     OH        39.00
     Alfred E Newman & Company      New York          NY        42.00
     Eliot Richards                 Queens            NY        30.00
     Isaiah J Schwartz and Company  Zanesville        OH        33.00
     Harold Alexander Fink          Davenport         IN        36.00
     Harriet Bailey                 Mamaroneck        NY        21.00
     Clair Butterfield              Teaneck           NJ        24.00
     Quentin Fields                 Cleveland         OH        27.00
     Jerry's Junkyard Specialties   Frogline          NH        12.00
     Kate's Out of Date Dress Shop  New York          NY        15.00
     Sam Johnson                    Bell Harbor       MI        18.00
     Joe Smith and Company          Big Falls         NM         3.00
     Corks and Bottles, Inc.        New York          NY         6.00
     Harry's Landmark Diner         Miningville       IN         9.00
                                                             ---------
                                            Grand Total        315.00
```

Adding Graphics

Program ex12b.sqr includes graphical features—a logo, solid lines, and a change of font in the heading.

Program ex12b.sqr

```
begin-setup
 declare-layout default
 end-declare
end-setup
begin-program
 do main
end-program
begin-procedure main
begin-select
name  (,1,30)
city  (,+1,16)
```

```
state (,+1,5)
tot   (,+1,11) edit 99999999.99
 next-listing no-advance need=1
 let #grand_total = #grand_total + &tot
from customers
end-select
graphic (,55,12) horz-line 20
print 'Grand Total' (+2,40)
print #grand_total (,55,11) Edit 99999999.99
end-procedure ! main
begin-heading 11
 print $current-date (1,1)
 page-number (1,60) 'Page '
 alter-printer point-size=14 font=4 ! switch font
 print 'Name'   (9,1) bold
 print 'City'   (,32) bold
 print 'State'  (,49) bold
 print 'Total'  (,61) bold
 alter-printer point-size=12 font=3 ! restore font
 graphic (9,1,66) horz-line 20
 print-image (1,23)
   type=bmp-file
   image-size=(21,5)
   source='acmelogo.bmp'
end-heading
```

GRAPHIC draws solid lines with the HORZ-LINE argument. Lines are positioned using a Production Reporting position specifier. The third number in the position specifier is the length of the line in characters. (The width of character cells is determined by the CHAR-WIDTH or MAX-COLUMNS arguments of DECLARE-LAYOUT. See Chapter 7, "SETUP Section.")

The HORZ-LINE argument of the GRAPHIC HORZ-LINE command is the thickness of the line, specified in decipoints (there are 720 decipoints per inch). For example, the command

```
graphic (10,1,66) horz-line 20
```

specifies a horizontal line below line 10 in the report starting with position 1 (the left side of the report) and stretching for 66 character positions (at 10 characters per inch this is 6.6 inches). The thickness of the line is 20 decipoints, which is 1/36 of an inch (about 0.7 millimeters).

You can also use the GRAPHIC command to draw vertical lines, boxes, and shaded boxes. See *sqrlaser.sqr*, in the SAMPLES subdirectory, for an example.

ALTER-PRINTER command changes the heading font. When used a second time, it restores the normal font for the rest of the report. FONT selects a font (typeface) supported by the printer. The font is specified by number, but the number is printer-specific. On PostScript printers, for example, font 3 is Courier, font 4 is Helvetica, and font 5 is Times Roman. See DECLARE-PRINTER in Volume 2 of the *Production Reporting Developer's Guide*.

POINT-SIZE specifies type size in points. You can use whole numbers or fractions (for example, POINT-SIZE=10.5). The following command changes the font to 14-point Helvetica:

```
alter-printer point-size=14 font=4 ! switch font
```

PRINT-IMAGE inserts the logo. PRINT-IMAGE is followed by a print position corresponding to the top left corner of the image (line 1, column 19 in our example). The TYPE option specifies the image file type. In our example, the image is stored in Windows bitmap format (.bmp). The size of the image is specified in terms of columns (width) and lines (height). In the example, the image is 30 characters wide (3 inches) and 7 lines high (1 1/6 inches).

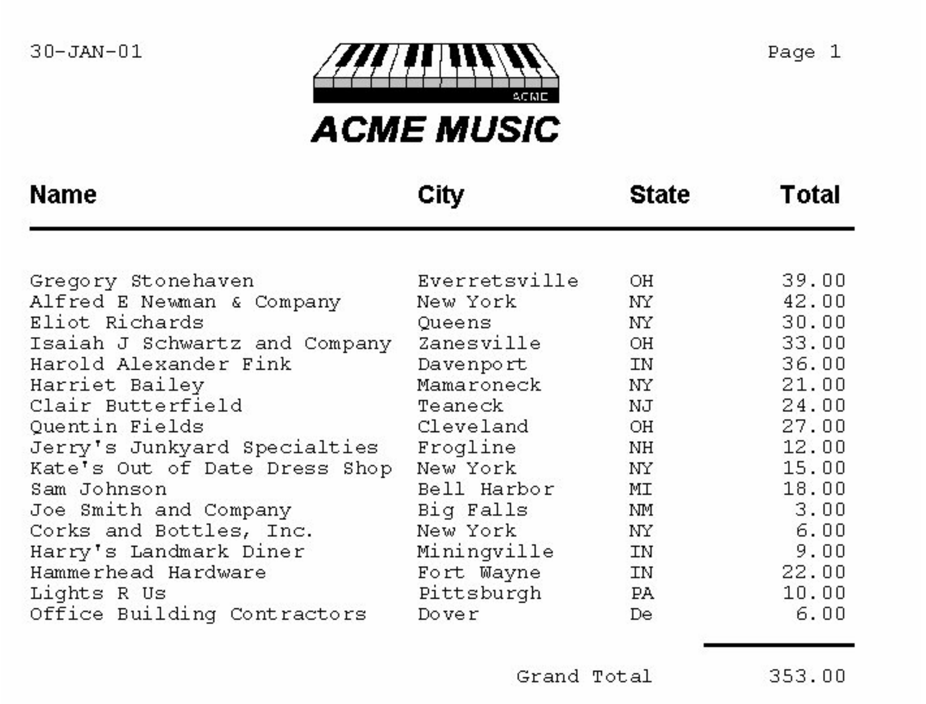
In Production Reporting, images are stored in external files. The format of the image must match that of the printer you are using. These formats are:

- Windows—BMP file images
- PostScript printer or viewer—EPS file
- HP LaserJet—HPGL file images
- HTML output—GIF, JPEG, and PNG formats
- Portable Document Format—BMP, GIF, JPEG, and PNG formats

For more information on these format options, see Chapter 30, "Printing Issues."

The `SOURCE` option specifies the file name of the image file. In our example, the file is *Acmelogo.bmp*. The file is assumed to reside in the current directory or in the directory where Production Reporting is installed, but the file can reside in any directory as long a full pathname for the image file is specified.

Output for *Program ex12b.sqr*



```
30-JAN-01                                                  Page 1

                          ACME MUSIC

Name                          City            State       Total
_______________________________________________________________

Gregory Stonehaven            Everretsville   OH          39.00
Alfred E Newman & Company     New York        NY          42.00
Eliot Richards                Queens          NY          30.00
Isaiah J Schwartz and Company Zanesville      OH          33.00
Harold Alexander Fink         Davenport       IN          36.00
Harriet Bailey                Mamaroneck      NY          21.00
Clair Butterfield             Teaneck         NJ          24.00
Quentin Fields                Cleveland       OH          27.00
Jerry's Junkyard Specialties  Frogline        NH          12.00
Kate's Out of Date Dress Shop New York        NY          15.00
Sam Johnson                   Bell Harbor     MI          18.00
Joe Smith and Company         Big Falls       NM           3.00
Corks and Bottles, Inc.       New York        NY           6.00
Harry's Landmark Diner        Miningville     IN           9.00
Hammerhead Hardware           Fort Wayne      IN          22.00
Lights R Us                   Pittsburgh      PA          10.00
Office Building Contractors   Dover           De           6.00
                                                       ________

                                      Grand Total        353.00
```

The output file now contains graphic language commands. Production Reporting can produce output for HP LaserJet printers in a file format that uses the HP PCL language or output for PostScript printers in a file format that uses the PostScript language. Production Reporting can also produce printer-independent output files in a special format called SQR Portable Format (SPF).

Production Reporting can create a printer-specific output file (LIS) or create the output in portable format (SPF). When you create an SPF file, the name of the image file is copied into it, and the image is processed at print time, when printer-specific output is generated. When SPF files are used, changes in image file contents are the next time you print or view the report. You can create printer-specific output by using Production Reporting Execute to directly generate an LIS file or by using Production Reporting Print to generate an LIS file from an SPF file. (See Chapter 30, “Printing Issues.”)

Sharing Images Among Reports

You can place logos and other images in reports using only PRINT-IMAGE. If several programs share an image definition, use DECLARE-IMAGE.

Program ex12c.sqr prints a simple form letter. It prints a logo using DECLARE-IMAGE and PRINT-IMAGE, and it prints a signature using PRINT-IMAGE.

Because the image is shared among several reports, DECLARE-IMAGE is contained in the file acme.inc. This file declares an image with *acme-logo* as the name. It specifies the logo used in the last sample program. The declaration includes the type and source file for the image. When the image is printed, do not respecify these attributes.

File acme.inc

```
declare-image acme_logo
       type=bmp-file
       image-size=(30,7)
       source='acmelogo.bmp'
end-declare
```

Multiple programs can share the declaration and include the file acme.inc. Change an attribute, such as the source, in one place only. The image size is specified and provides the default.

To change the size of an image in a report, use the IMAGE-SIZE argument in PRINT-IMAGE to overrides the image size specified in DECLARE-IMAGE.

Program ex12c.sqr

```
begin-setup
#include 'acme.inc'
end-setup
begin-program
  do main
end-program
begin-procedure main
begin-select
name
addr1
addr2
city
state
zip
phone
  do write_letter
from customers
order by name
```

```
end-select
end-procedure ! main
begin-procedure write_letter
move &city to $csz
concat ', ' with $csz
concat &state with $csz
concat ' ' with $csz
move &zip to $zip xxxxx-xxxx
concat $zip with $csz
move &phone to $phone_no (xxx)bxxx-xxxx   ! Edit phone number.
begin-document (1,1,0)
&name                                           @logo
&addr1
&addr2
$csz
.b
.b
.b
                                               $current-date
Dear &name
.b
     Thank you for your inquiry regarding Encore, Maestro!!, our
revolutionary teaching system for piano and organ. If you've always wanted
to play an instrument but felt you could never master one, Encore,
Maestro!! is made for you.
.b
     Now anyone who can hum a tune can play one too. Encore, Maestro!!
begins with a step-by-step approach to some of America's favorite songs.
You'll learn the correct keyboarding while hearing the sounds you make
through the headphones provided with the Encore, Maestro!! system. From
there, you'll advance to intricate compositions with dazzling melodic runs.
Encore, Maestro!! can even teach you to improvise your own solos.
.b
     Whether you like classical, jazz, pop, or blues, Encore, Maestro!! is
the music teacher for you.
.b
     A local representative will be calling you at $phone_no
to set up an in-house demonstration, so get ready to play your favorite
tunes!!
.b
                                Sincerely,
                                @signature
.b
.b
                                Clark Axelotle
end-document
position () @logo
print-image acme-logo ()
  image-size=(16,4)
position () @signature
print-image ()
  type=bmp-file
  image-size=(12,3)
  source='clark.bmp'
new-page
end-procedure ! write_letter
```

#INCLUDE, which is performed at compile time, pulls in text from another file. In this program, #INCLUDE 'acme.inc' includes the code from the file acme.inc.

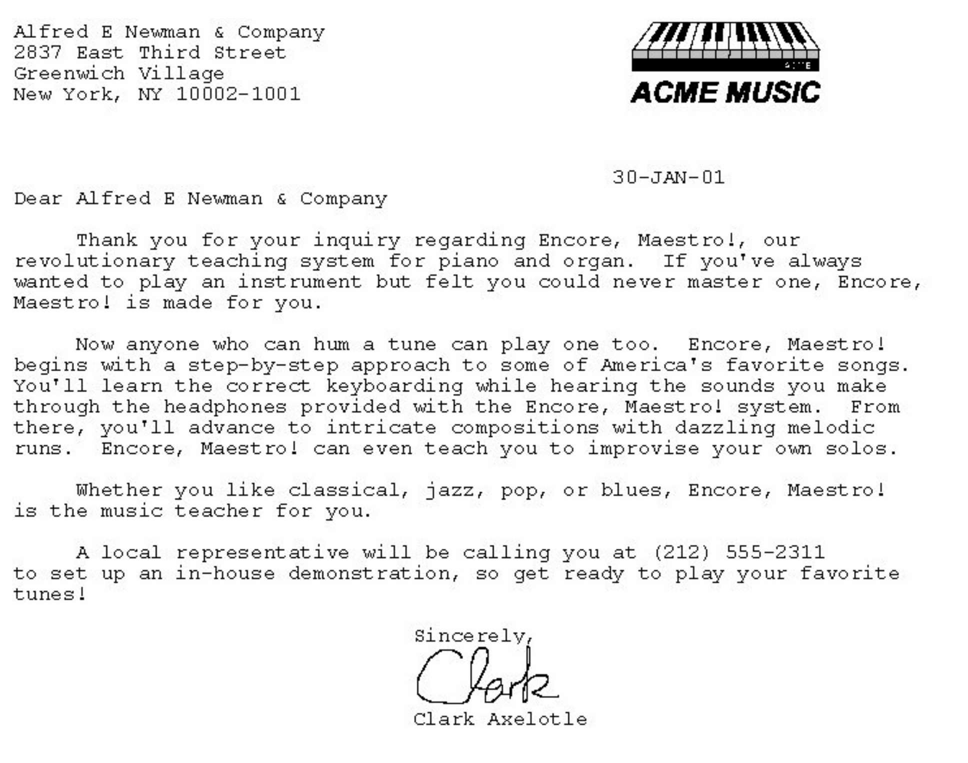
The DOCUMENT section begins with BEGIN-DOCUMENT and ends with END-DOCUMENT. It uses variables and document markers to print inside the letter. The program uses variables for the name and address, the date, and the phone number. It uses document markers for the logo and signature.

Document markers are placeholders in the letter. The program uses the document markers *@logo* and *@signature* in a POSITION command before printing each image. The document markers make it unnecessary to specify the position of these items in PRINT-IMAGE. Instead, you simply print to the current position.

The date is prepared with the reserved variable *$current-date*. It is printed directly in the DOCUMENT section without issuing a PRINT command.

CONCAT put together the city, state, and zip code. In the DOCUMENT section, variables retain their predefined size. A column variable, for example, will remain the width of the column as defined in the database. You can print the date and phone number directly, however, because they fall at the end of a line, without any following text.

Output for *Program ex12c.sqr*



```
Alfred E Newman & Company
2837 East Third Street
Greenwich Village
New York, NY 10002-1001                                    ACME MUSIC


                                                 30-JAN-01
Dear Alfred E Newman & Company

     Thank you for your inquiry regarding Encore, Maestro!, our
revolutionary teaching system for piano and organ.  If you've always
wanted to play an instrument but felt you could never master one, Encore,
Maestro! is made for you.

     Now anyone who can hum a tune can play one too.  Encore, Maestro!
begins with a step-by-step approach to some of America's favorite songs.
You'll learn the correct keyboarding while hearing the sounds you make
through the headphones provided with the Encore, Maestro! system.  From
there, you'll advance to intricate compositions with dazzling melodic
runs.  Encore, Maestro! can even teach you to improvise your own solos.

     Whether you like classical, jazz, pop, or blues, Encore, Maestro!
is the music teacher for you.

     A local representative will be calling you at (212) 555-2311
to set up an in-house demonstration, so get ready to play your favorite
tunes!

                                Sincerely,
                                Clark
                                Clark Axelotle
```

Printing Bar Codes

Production Reporting supports a wide variety of bar code types, which you can include in your Production Reporting report.

Use PRINT-BAR-CODE to create a bar code. Specify the position of the bar code in an ordinary position qualifier. In separate arguments, specify the bar code type, height, text to encode, caption, and optional check sum. For example:

```
print-bar-code (1,1)
  type=1
  height=0.5
  text='01234567890'
  caption='0 12345 67890'
```

PRINT-BAR-CODE arguments can be variables or literals. This example produces the following bar code.

012345 67890

Note:

For further information, see PRINT-BAR-CODE in Volume 2 of the *Production Reporting Developer's Guide.*

Using Color

This section describes CREATE-COLOR-PALETTE, DECLARE-COLOR-MAP, ALTER-COLOR-MAP, GET-COLOR, and SET-COLOR. Use these commands to manipulate the colors of reports and chart elements. For detailed information on these commands see Volume 2 of the *Production Reporting Developer's Guide.*

Production Reporting comes with sixteen defined colors. Use DECLARE-COLOR-MAP (in the BEGIN-SETUP section) to define individual colors and CREATE-COLOR-PALETTE to define chart data series colors.

You can define an umlimited number of colors. For example:

```
begin-setup
declare-color-map
light_blue = (193, 222, 229)
pink = (221,79,200)
end-setup
```

Changing Color Specifications

The following commands change the Production Reporting default colors or the colors defined by DECLARE-COLOR-MAP:

- ALTER-COLOR-MAP
- GET-COLOR

- `SET-COLOR`

ALTER-COLOR-MAP

`ALTER-COLOR-MAP` dynamically alters a defined color; it does not define a new color. `ALTER-COLOR-MAP` is allowed wherever `PRINT` is allowed.

GET-COLOR

`GET-COLOR` retrieves the current color. `GET-COLOR` is allowed wherever `PRINT` is allowed.

If the requested color setting does not map to a defined name, the name is returned as *RGBredgreenblue*, where each component is a three digit number. For example, *RGB127133033*. You can use this format wherever you use a color name. The color name 'none' is returned if no color is associated with the specified area.

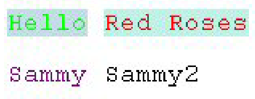
SET-COLOR

`SET-COLOR` is allowed wherever `PRINT` is allowed. If the specified color name is not defined, Production Reporting uses the settings for the color name 'default'. Use the color name 'none' to turn off the color for a specified area.

Sample Program

```
begin-setup
    declare-color-map
    light_blue = (193,222,229)
    end-declare
end-setup
begin-program
    print 'Hello' ()
        foreground=('green')
        background=('light_blue')
    alter-color-map name = 'light_blue' value = (193,240,230)
    print 'Red Roses' (,7)
        foreground=('red')
        background=('light_blue')
        get-color print-text-foreground=($print-foreground)
        set-color print-text-foreground=('purple')
    print 'Sammy' (+2,1)
        set-color print-text-foreground=($print-foreground)
    print 'Sammy2' (,7)
end-program
```

Figure 1 Output for Sample Program



```
Hello Red Roses

Sammy Sammy2
```

`ALTER-COLOR-MAP` changed the previously specified light blue background to be slightly darker. `SET-COLOR` and `GET-COLOR` changed the text color.

14

Business Charts

In This Chapter

About Business Charts

Production Reporting provides two commands for creating business charts, `DECLARE-CHART` and `PRINT-CHART`, and a rich set of chart types—line, pie, bar, stacked bar, 100% bar, overlapped bar, floating bar, histogram, area, stacked area, 100% area, *xy*-scatter plot, combination, bubble, and high-low-close.

You can customize many chart attributes by turning on three-dimensional effects or setting titles and legends. You can also move charts from one hardware platform to another.

Like cross-tabular reports (see Chapter 9, “Cross-Tabular Reports”), business charts can be prepared using data held in an array. If you have a cross-tabular report, just one more step creates a chart using the data collected in the array.

Creating Charts

Program ex13a.sqr builds on *Program ex8a.sqr* (see Chapter 9, “Cross-Tabular Reports”). *Program ex8a.sqr* combined two reports in one program. *Program ex13a.sqr* produces two charts corresponding to the two cross-tabs.

Here is the code. The bold lines are the changed or added lines.

Program ex13a.sqr

```
#define max_categories 3
#define max_products 100
begin-setup
  create-array
     name=order_qty       size={max_products}
     field=product:char   field=month_qty:number:3
  create-array
     name=order_qty2      size={max_categories}
     field=category:char  field=month_qty:number:3
  declare-chart orders-stacked-bar
    chart-size=(70,30)
    title='Order Quantity'
    legend-title='Month'
    type=stacked-bar
  end-declare ! orders-stacked-bar
end-setup
begin-program
  do select_data
  do print_array
  print '-' (+2,1,70) fill
  position (+1)
  do print_array2
  new-page
  let $done = 'YES' ! Don't need heading any more
  do print_the_charts
end-program
begin-procedure print_array
  let #entry_cnt = #i
  let #i = 0
  let #prod_tot = 0
  while #i <= #entry_cnt

      let $product  = order_qty.product(#i)
      let #jan      = order_qty.month_qty(#i,0)
      let #feb      = order_qty.month_qty(#i,1)
      let #mar      = order_qty.month_qty(#i,2)
        let #prod_tot = #jan + #feb + #mar
     if #prod_tot > 0

      print $product  (,1,30)
      print #jan      (,32,9) edit 9,999,999
      print #feb      (,42,9) edit 9,999,999
      print #mar      (,52,9) edit 9,999,999
      print #prod_tot (,62,9) edit 9,999,999
      position (+1)
   end-if
      let #i = #i + 1
  end-while
end-procedure ! print_array
begin-procedure print_array2
  let #i = 0
  while #i < {max_categories}

   !if #category_tot > 0
      let $category = order_qty2.category(#i)
      let #jan      = order_qty2.month_qty(#i,0)
      let #feb      = order_qty2.month_qty(#i,1)
      let #mar      = order_qty2.month_qty(#i,2)
      let #category_tot = #jan + #feb + #mar
      print $category     (,1,31)
      print #jan          (,32,9) edit 9,999,999
      print #feb          (,42,9) edit 9,999,999
      print #mar          (,52,9) edit 9,999,999
    !end-if
      print #category_tot (,62,9) edit 9,999,999
      position (+1)
      let #jan_total = #jan_total + #jan
      let #feb_total = #feb_total + #feb
      let #mar_total = #mar_total + #mar
      let #i = #i + 1
  end-while
  let #grand_total = #jan_total + #feb_total + #mar_total
  print 'Totals'     (+2,1)
  print #jan_total   (,32,9) edit 9,999,999
  print #feb_total   (,42,9) edit 9,999,999
  print #mar_total   (,52,9) edit 9,999,999
  print #grand_total (,62,9) edit 9,999,999
end-procedure ! print_array2
begin-procedure select_data
  let order_qty2.category(0)='$0-$4.99'
  let order_qty2.category(1)='$5.00-$100.00'
  let order_qty2.category(2)='Over $100'
begin-select
order_date
! the price / price category for the order
c.price &price
  move &price to #price_num
  evaluate #price_num
  when < 5.0
     let #x = 0
     break
  when <= 100.0
     let #x = 1
     break
  when-other
     let #x = 2
     break
  end-evaluate
! The quantity for this order
quantity
  if &quantity < 100
  let #j = to_number(datetostr(&order_date,'MM')) - 1
  if #j < 3
      let order_qty2.month_qty(#x,#j) =
               order_qty2.month_qty(#x,#j) + &quantity
  end-if
  end-if
! the product for this order
description
   if &description  = 'Thingamajigs'
        goto next
    else
        if &description = 'Widgets'
```

```
            goto next
    else
    if #i = 0 and order_qty.product(#i) = ''
      let order_qty.product(#i) = &description
    end-if
    if order_qty.product(#i) != &description
      let #i = #i + 1
      if #i >= {max_products}
         display 'Error: There are more than {max_products} products'
         stop
      end-if
      let order_qty.product(#i) = &description
    end-if
    if #j < 3
      let order_qty.month_qty(#i,#j) =
               order_qty.month_qty(#i,#j) + &quantity
    next:
    end-if
    end-if
    end-if
    from  orders a, ordlines b, products c
where a.order_num = b.order_num
and   b.product_code = c.product_code
order by description
end-select
end-procedure ! select_data
begin-heading 5
  if not ($done = 'YES')
   print $current-date (1,1)
   page-number (1,64) 'Page '
   print 'Order Quantity by Product and Price Category by Month' (2,10)
   print 'Product / Price Category'   (4,1)
   print '  January' (,32)
   print ' February' (,42)
   print '    March' (,52)
   print '    Total' (,62)
   Print '-'         (5,1,70) Fill
  end-if
end-heading
begin-procedure print_the_charts
  print-chart orders-stacked-bar (+2,1)
    item-color=('ChartBackground', (255,0,0))
    data-array=order_qty
    data-array-row-count=7
    data-array-column-count=4
    !item-color=ChartBackground 'red'
    data-array-column-labels=('Jan','Feb','Mar')

    sub-title='By Product By Month'

  new-page
  print-chart orders-stacked-bar (+2,1)
    data-array=order_qty2
    data-array-row-count=3
    data-array-column-count=4
    data-array-column-labels=('Jan','Feb','Mar')
    sub-title='By Price Category By Month'
```

```
end-procedure ! print_the_charts
```

Defining Charts

The two chart sections in *Program ex13a.sqr* are specified with DECLARE-CHART in the SETUP section and are named *orders-stacked-bar*. The chart width and height is specified by character cells. The charts are 70 characters wide, which is 7 inches on a default layout. The height (or depth) of the charts is 30 lines, which translates to 5 inches at 6 lines per inch. These dimensions define a rectangle that contains the chart. The box that surrounds the chart is drawn by default, but you can disable it using BORDER=NO.

The title is centered at the top of the chart. The text generated by LEGEND-TITLE must fit in the small legend box above the categories, so keep this description short. In general, charts look best when the text items are short. Here is the DECLARE-CHART command.

```
declare-chart orders-stacked-bar
    chart-size=(70,30)
    title='Order Quantity'
    legend-title='Month'
    type=stacked-bar
  end-declare ! orders-stacked-bar
```

Printing Charts

The PRINT-CHART commands are based on the *orders-stacked-bar* chart that was declared in the preceding section.

```
  print-chart orders-stacked-bar (+2,1)
    data-array=order_qty
    data-array-row-count=12
    data-array-column-count=4
    data-array-column-labels=('Jan','Feb','Mar')
sub-title='By Product By Month'
  new-page
  print-chart orders-stacked-bar (+2,1)
    data-array=order_qty2
    data-array-row-count=3
    data-array-column-count=4
    data-array-column-labels=('Jan','Feb','Mar')
    sub-title='By Price Category By Month'
```

The datasource is specified using DATA-ARRAY. The named array has a structure that is specified by TYPE. For a stacked-bar chart, the first field in the array gives the names of the bar categories. The rest of the fields are series of numbers. In this case, each series corresponds to a month.

The subtitle goes under the title and can be used as a second line of the title. A legend labels the series. DATA-ARRAY-COLUMN-LABELS passes these labels. DATA-ARRAY-ROW-COUNT is the number of rows (bars) to chart and DATA-ARRAY-COLUMN-COUNT is the number of fields in the array the chart uses. The array has four fields—the product (or price category) field and the series that specifies three months.

Running Programs

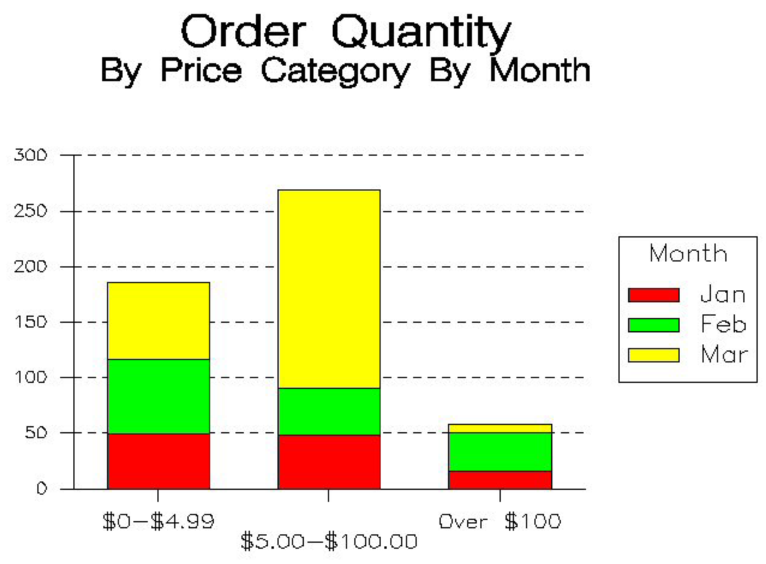
Graphical reports require slightly different techniques for running programs and viewing output. For UNIX, specify the printer type with the `-PRINTER:xx` flag. For HP LaserJet printers, enter `-PRINTER:HP` on the command line. For PostScript printers, enter `-PRINTER:PS`. For example:

```
sqr ex9a username/password  -printer:hp
```

You can also use `-KEEP` to produce output in the SQR Portable File format (SPF) and print it using Production Reporting Print. You still must use the `-PRINTER:xx` flag when printing. SQR Portable File format is covered in greater detail in Chapter 30, "Printing Issues."

With SQR Production Reporting Studio, neither `-PRINTER:xx` nor `-KEEP` is required. The output automatically appears in the Viewer or Web browser window after the report runs. A portion of the output is shown next. The charts appear on pages 2 and 3 of the report.

Output for *Program ex13a.sqr*



Passing Data to Charts

To pass data to a chart, use the first field for the bar descriptions (or lines or areas) and then use one or more additional fields with series of numbers. This procedure is common to many chart types, including line, bar, stacked-bar, 100% bar, overlapped bar, histogram, area, stacked-area, and 100% area. You can omit the first field and Production Reporting uses cardinal numbers (1, 2, 3...) for the bars. Only text fields are used for these options.

For pie charts, only one series is allowed, and you can specify which segments to "explode," or pull away, from the center of the pie. A third field in the array can have 'Y' and 'N' values that indicate whether to explode the segment. If 'Y' is the value in the first row of the array, the pie segment that corresponds to the first row is exploded. With pie charts, you cannot omit the first field with the descriptions. Pie charts cannot have more than 12 segments.

Pie charts display the numeric value next to each segment. The description is displayed in the legend. In addition, Production Reporting displays the percentage next to the value. You can disable this feature by specifying `PIE-SEGMENT-PERCENT-DISPLAY=NO`.

When data is passed to an *xy* scatter plot or a floating-bar chart, the series are paired. A pair in a floating-bar chart represents the base and height of the bars. A pair in an *xy*-scatter plot represents *x* and *y* coordinates. In an *xy*-scatter plot, the first field does not have descriptions. In a floating-bar chart, the first field may or may not have descriptions for the bars. For both types, you can have one or more pairs of series.

Changing Colors with New Graphics

Changing colors with New Graphics involves:

- Specifying Chart Data Series Colors
- Specifying Chart Item Colors

Note:

To use New Graphics, the `NewGraphics` entry in the [Default-Settings] section of the SQR.INI file must be set to `TRUE` (`NewGraphics=True`).

To use New Graphics on UNIX, an X Server is required. If your UNIX environment does not have access to an X Server (either local or remote), there is a free X Server (called VNC) available under the GNU General Public License at http://www.tightvnc.com. (Please note that the VNC Server is not officially test or certified.)

Table 2 Supported Output Types for New Graphics

Output Type	Supported?
HT-HTML	Yes
EH-Enhanced HTML	Yes
HP-Hewlett Packard	No
LP - Line Printer	No
PD-PDF	Yes
PS-Postscript	Yes
WP-Windows Printer	No
Production Reporting Viewer	No

Specifying Chart Data Series Colors

Color palettes in the New Graphics set the colors of each data point in a data series. Use `CREATE-COLOR-PALETTE` to specify the color palette in a business chart. For example:

```
Create-Color-Palette
    Name = 'Test-Palette'
    Color_1 = (100,133,238)
    Color_2 = (0, 0, 255)
    Color_3 = (0,255,0)
    Color_4 = (0,0,255)
    Color_5 = (0,0,0)
```

You can specify any number of palettes, with up to 64 colors defined in each palette. If there are more data points in the data sets than defined colors in the palette, the palette resets and continues to set the data point colors from Color_1 to Color_n.

After you define a color palette, you can use it in the `DECLARE-CHART` and `PRINT-CHART` commands. The following code demonstrates the use of a color palette:

```
Print-Chart test_Chart
    COLOR-PALETTE = 'Test-Palette'
```

Note:

For now, Production Reporting for UNIX requires a x-windows server running when NewGraphics=TRUE. This is a requirement of the New Graphics, because that is the only way at present to run Java-based graphics on a UNIX system without a graphic display, either local or remote.

For now, when using New Graphics in Windows, the color palette in Display property settings, must be set to either 65536 Colors or 256 Colors.

Specifying Chart Item Colors

You can specify chart item colors in these ways:

- Use ITEM-COLOR in DECLARE-CHART and PRINT-CHART
- Use ATTRIBUTES in DECLARE-CHART and PRINT-CHART

Use ITEM-COLOR in DECLARE-CHART and PRINT-CHART

You can specify the foreground and background colors of the individual areas within a Business Chart using `ITEM-COLOR=(rgb-value)` in the `DECLARE-CHART` and `PRINT-CHART` commands. Figure 2 is an example of a business chart.

Figure 2 A Sample Business Chart

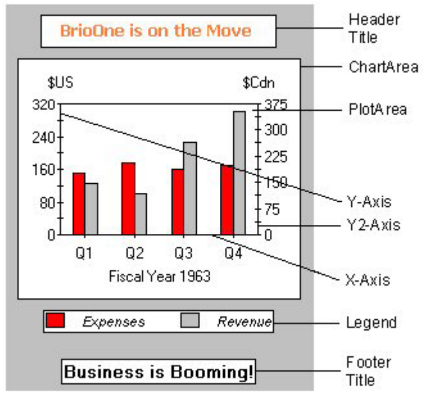


Table 3 lists chart item keywords for `ITEM-COLOR`.

Table 3 Chart ITEM-COLOR Keywords and Descriptions

Keyword	Description
ChartBackground	Background color of entire chart area
ChartForeground	Text and Line color of chart area
HeaderBackground	Area within the text box specified for the title and sub-title
HeaderForeground	Text color of the Title and sub-title
LegendBackground	Area within the box defining the legend
LegendForeground	Text and Outline color of the legend
ChartAreaBackground	Area that includes the body of the chart
ChartAreaForeground	Text and Line colors of the chart area
PlotAreaBackground	Area within the X and Y Axis of a chart

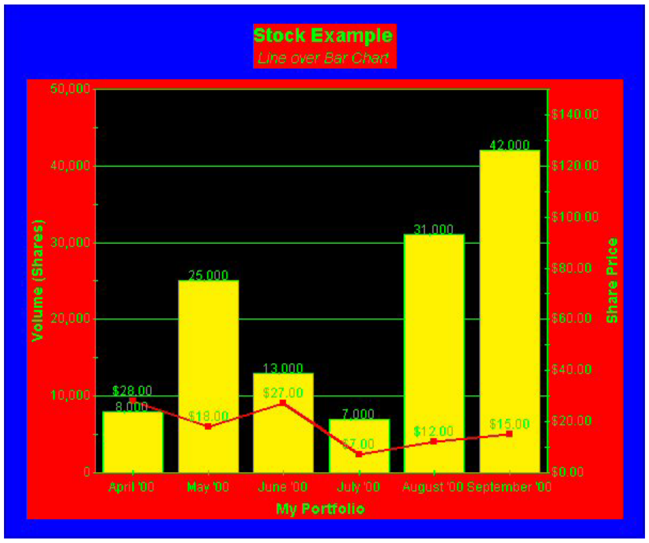
As is shown in the examples below, you can enter the syntax to define chart colors as text, numbers, or both.

```
ITEM-COLOR = ('ChartAreaBackground',($green))
ITEM-COLOR = ('HeaderBackground',(255,185,5))
ITEM-COLOR = ('ChartBackground',(#red,185,5))
```

The following syntax defines the colors that appear in the chart in Figure 3.

```
ITEM-COLOR = ('ChartForeground',($blue))
ITEM-COLOR = ('ChartAreaBackground',(#red,185,5))
ITEM-COLOR = ('PlotAreaBackground',($black))
ITEM-COLOR = ('HeaderForeground',($green))
ITEM-COLOR = ('HeaderBackground',($red))
ITEM-COLOR = ('ChartBackground',($blue))
```

Figure 3 Sample Chart Colors



Use ATTRIBUTES in DECLARE-CHART and PRINT-CHART

You can define a group of colors for an individual chart element and specify the foreground and background colors of the individual areas in a chart using the `COLOR`, `FOREGROUND`, and `BACKGROUND` declaration keywords. These declaration keywords are part of the `ATTRIBUTES` argument in the `DECLARE-CHART` and `PRINT-CHART` commands.

Note:

In Production Reporting Release 9.0, the COLOR declaration keyword in the ATTRIBUTES argument replaces the functionality provided by the ITEM-COLOR argument. As a result, values set with the COLOR declaration keyword override the values set with ITEM-COLOR.

For additional information, see “Attributes Argument” in Volume 2 of the *Production Reporting Developer's Guide*.

Creating Combination Charts

Use the Y2-Axis syntax in `PRINT-CHART` and `DECLARE-CHART` to create combination charts. For example, you could create a Line Chart over a Bar Chart or a Hi-Low Chart over a Bar Chart.

Note:

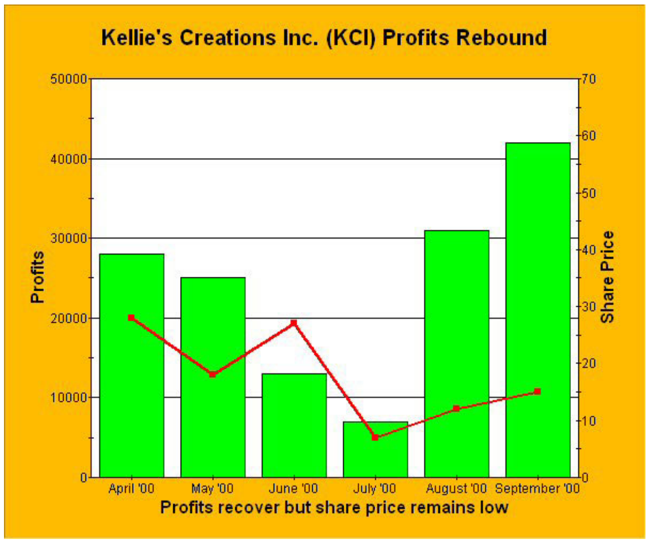
Combination charts (and the Y2-Axis syntax used to create them) require that **NewGraphics=True** in the [Default-Settings] section of the SQR.INI file.

See DECLARE-CHART and PRINT-CHART in Volume 2 of the *Production Reporting Developer's Guide* for information on the Y2-Axis syntax. See “SQR.INI” in Volume 2 of the *Production Reporting Developer's Guide* for information on the SQR.INI file.

Sample Line Chart Over Bar Chart

Figure 4 shows a Line Chart over Bar Chart created using Y2-Axis syntax. The Production Reporting code used to create the chart appears after the chart.

Figure 4 Line Chart Over Bar Chart



```
Begin-Setup
  !Dollar-Symbol <Fr>
  Declare-Printer Default-ps
    Color = yes
  End-Declare
  Declare-Report Default
    Printer-Type = ps
  End-Declare
  Declare-Layout Default
    Max-Lines = 62
    Left-Margin = 0.10
    Right-Margin = 0.25
  End-Declare
  Create-Array    ! Array of Bar Chart Values
    Name = profits
    Size = 10
    Field = date:char
    Field = profits:number
Create-Array    ! This Array is passed to the Print-Chart
    Name = quotes
    Size = 10
    Field = date:char
    Field = sharePrice:number
```

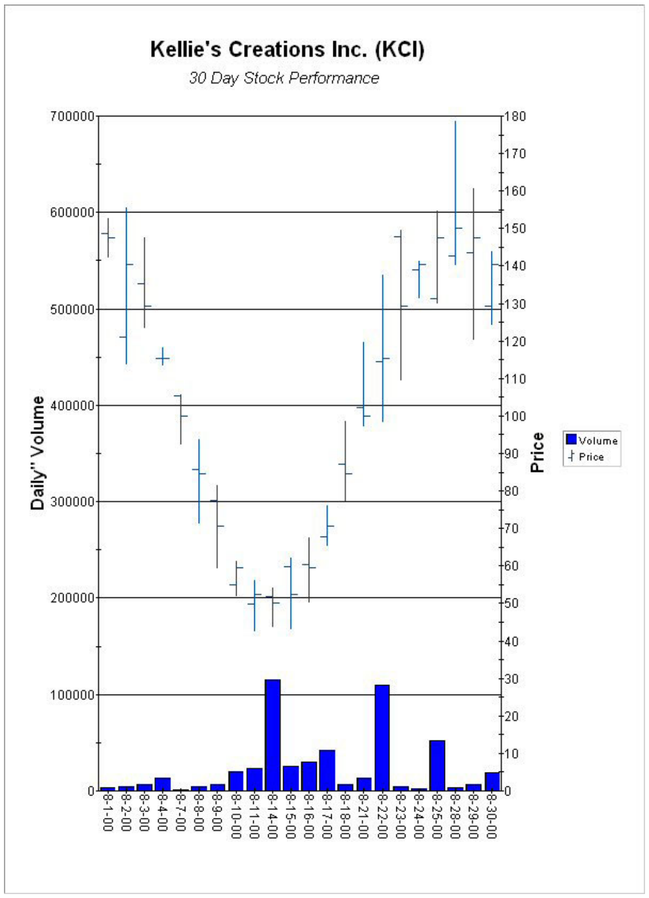
```
! Declare-Chart is not necessary to print a chart. However, it can be used
as a template and its properties can be modified in each Print-Chart that
uses this chart name.
! NOTE: Declare-Chart must physically preceede Print-Chart
  Declare-Chart stock_chart
    Chart-Size = (60,30)
    Type = line
    Y-Axis-Label = 'Profits'
  End-Declare
End-Setup
Begin-Report
  Alter-Locale
     Locale = 'US-English'
     Money-Sign-Location = Right
     Money-Sign = '$$'
  Put  'April 00' 28
  Into quotes(0) date sharePrice
  Put  'May 00' 18
  Into quotes(1) date sharePrice
  Put  'June 00' 27
  Into quotes(2) date sharePrice
  Put  'July 00' 7
  Into quotes(3) date sharePrice
  Put  'August 00' 12
  Into quotes(4) date sharePrice
  Put  'September 00' 15
  Into quotes(5) date sharePrice
  Put  'April ''00' -8000
  Into profits(0) date profits
  Put  'May ''00' 25000
  Into profits(1) date profits
  Put  'June ''00' 13000
  Into profits(2) date profits
  Put  'July ''00' 7000
  Into profits(3) date profits
  Put  'August ''00' 31000
  Into profits(4) date profits
  Put  'September ''00' 42000
  Into profits(5) date profits

  Move 1 TO #x
  Move 1 TO #y
  Move 6 TO #row
  Move 2 TO #col
 Create-Color-Palette
  Name = 'Green'
  Color_1 = (0,255,0)
  Color_2 = (0,255,0)
 Create-Color-Palette
  Name = 'Red'
  Color_1 = (255,0,0)
  Color_2 = (255,0,0)
! Print the Chart
  Print-Chart stock_chart (#y, #x)
    Title = 'Kellie''s Creations Inc. (KCI) Profits Rebound'
    Sub-Title = 'Test Program'
    Fill = color
    Y-Axis-Min-Value = -20000
    Y2-Type = line
    Y2-Data-Array-Row-Count = #row
    Y2-Data-Array-Column-Count = #col
    Y2-Data-Array = quotes
    Y2-Axis-Label = 'Share Price'
    Y2-Color-Palette = 'Red'
    Y2-Axis-Max-Value = 70
    Legend = no
    Type = bar
    Data-Array = profits
    Data-Array-Row-Count = #row
    Data-Array-Column-Count = #col
    Color-Palette = 'Green'
    X-AXIS-LABEL = 'Profits recover but share price remains low'
    Item-Color = ('ChartBackground',(255,185,5))
    Item-Color = ('HeaderBackground',(255,185,5))
    Item-Color = ('ChartAreaBackground',(255,185,5))
    X-Axis-Rotate = 8000000
    Y-Axis-Mask = '$$$$,$$MI'
    Y2-Axis-Mask = '%%%.%%'
End-Report
```

Sample Hi-Low Chart Over Bar Chart

Figure 5 shows a Hi-Low Chart over Bar Chart created using Y2-Axis syntax. The Production Reporting code used to create the chart appears after the chart.

Figure 5 Hi-Low Chart Over Bar Chart



```
Begin-Setup
  Declare-Printer Default-ps
    Color = yes
  End-Declare
  Declare-Report Default
    Printer-Type = ps
  End-Declare
  Declare-Layout Default
    Max-Lines = 62
    Left-Margin = 0.10
    Right-Margin = 0.25
  End-Declare
  Create-Array     ! Array of Bar Chart Values
    Name = volume
    Size = 30
    Field = date:char
    Field = shares:number
  Create-Array     ! This Array is passed to the Print-Chart
    Name = quotes
    Size = 30
    Field = dt:char
    Field = hi:number
    Field = lo:number
    Field = cl:number
    Field = op:number
```

```
! Declare-Chart is not necessary to print a chart. However, it can be used
as a template and its properties can be modified in each Print-Chart that
uses this chart name.
! NOTE: Declare-Chart must physically preceede Print-Chart
  Declare-Chart stock_chart
    Chart-Size = ( 60,50)
    Type = line
    Y-Axis-Label = 'Daily Volume'
    Y2-Axis-Label = 'Price'
  End-Declare
End-Setup
Begin-Report
  Put  '8-1-00' 3000
  Into volume(0) date shares
  Put  '8-2-00' 4000
  Into volume(1) date shares
  Put  '8-3-00' 7000
  Into volume(2) date shares
  Put  '8-4-00' 13000
  Into volume(3) date shares
  Put  '8-7-00' 1100
  Into volume(4) date shares
  Put  '8-8-00' 4000
  Into volume(5) date shares
  Put  '8-9-00' 7000
  Into volume(6) date shares
  Put  '8-10-00' 20000
  Into volume(7) date shares
  Put  '8-11-00' 23455
  Into volume(8) date shares
  Put  '8-14-00' 115488
  Into volume(9) date shares
  Put  '8-15-00' 25688
  Into volume(10) date shares
  Put  '8-16-00' 30000
  Into volume(11) date shares
  Put  '8-17-00' 42544
  Into volume(12) date shares
  Put  '8-18-00' 7000
  Into volume(13) date shares
  Put  '8-21-00' 13557
  Into volume(14) date shares
  Put  '8-22-00' 110100
  Into volume(15) date shares
  Put  '8-23-00' 4000
  Into volume(16) date shares
  Put  '8-24-00' 2568
  Into volume(17) date shares
  Put  '8-25-00' 51778
  Into volume(18) date shares
  Put  '8-28-00' 3000
  Into volume(19) date shares
  Put  '8-29-00' 7005
  Into volume(20) date shares
  Put  '8-30-00' 19286
  Into volume(21) date shares
  Put  '8-31-00' 5837
```

```
    Into volume(22) date shares
  ! Load the array with the Chart Data. Max rows is specified in Create-Array
    Put '8-1-00' 152.717439 142.367065 148.647876 147.552826
      Into QUOTES(0) DT HI LO CL OP
    Put '8-2-00' 155.465936 113.831755 120.981495 140.450849
      Into QUOTES(1) DT HI LO CL OP
    Put '8-3-00' 147.561490 123.530081 135.420007 129.389262
      Into QUOTES(2) DT HI LO CL OP
    Put '8-4-00' 118.330161 113.513690 115.390331 115.450849
      Into QUOTES(3) DT HI LO CL OP
    Put '8-7-00' 105.544602 92.638934 105.331129 99.999999
      Into QUOTES(4) DT HI LO CL OP
    Put '8-8-00' 93.602439 71.601649 85.824579 84.549149
      Into QUOTES(5) DT HI LO CL OP
    Put '8-9-00' 81.444453 59.592125 77.575764 70.610736
      Into QUOTES(6) DT HI LO CL OP
    Put '8-10-00' 61.358504 52.099826 55.013313 59.549149
      Into QUOTES(7) DT HI LO CL OP
    Put '8-11-00' 56.085999 42.826230 49.718137 52.447174
      Into QUOTES(8) DT HI LO CL OP
    Put '8-14-00' 54.011658 43.932310 51.848875 50.000000
      Into QUOTES(9) DT HI LO CL OP
    Put '8-15-00' 62.218576 43.322100 59.696701 52.447175
      Into QUOTES(10) DT HI LO CL OP
    Put '8-16-00' 67.582572 50.516552 60.447208 59.549152
      Into QUOTES(11) DT HI LO CL OP
    Put '8-17-00' 76.107547 65.588449 67.694709 70.610740
      Into QUOTES(12) DT HI LO CL OP
    Put '8-18-00' 98.532392 77.516249 87.257549 84.549153
      Into QUOTES(13) DT HI LO CL OP
    Put '8-21-00' 119.583732 97.471240 102.172852 100.000003
      Into QUOTES(14) DT HI LO CL OP
    Put '8-22-00' 137.581958 98.422319 114.440161 115.450853
      Into QUOTES(15) DT HI LO CL OP
    Put '8-23-00' 149.576945 109.776527 147.859431 129.389266
      Into QUOTES(16) DT HI LO CL OP
    Put '8-24-00' 141.239541 131.497523 138.871749 140.450852
      Into QUOTES(17) DT HI LO CL OP
    Put '8-25-00' 154.723551 130.154680 131.220903 147.552827
      Into QUOTES(18) DT HI LO CL OP
    Put '8-28-00' 178.681600 140.426038 142.696831 150.000000
      Into QUOTES(19) DT HI LO CL OP
    Put '8-29-00' 160.592879 120.549581 143.464461 147.552824
      Into QUOTES(20) DT HI LO CL OP
    Put '8-30-00' 143.789056 124.445614 129.321174 140.450847
      Into QUOTES(21) DT HI LO CL OP
    Put '8-31-00' 142.221173 123.263141 132.305625 129.389258
      Into QUOTES(22) DT HI LO CL OP

    Move 5 TO #x
    Move 6 TO #y
    Move 22 TO #row
    Move 2 TO #col
    Move 6 TO #col2
    Create-Color-Palette
      Name = 'Test-Palette'
      Color_1 = (0,0,0)        !Black
      Color_2 = (255, 50, 100) !Red
      Color_3 = (100,255,30)   !Green
      Color_4 = (0,0,255)      !Blue
      Color_5 = (255,255,255)  !White
   Create-Color-Palette
    Name = 'Line'
    Color_1 = (4,74,148)
    Color_2 = (4,74,148)
   Create-Color-Palette
    Name = 'Blue'
    Color_1 = (0,0,255)
    Color_2 = (0,0,255)
! Print the Chart
    Print-Chart stock_chart (#y, #x)
      Fill = color
      Type = Bar
      Title = 'Kellie''s Creations Inc. (KCI)'
      Sub-Title = '30 Day Stock Performance'
      Data-Array-Row-Count = #row
      Data-Array-Column-Count     = #col
      Data-Array-Column-Labels = ('Volume')
      Data-Array = volume
      Legend = yes
      Color-Palette = 'Blue'
      Y-Axis-Max-Value = 700000
      Y2-Type = high-low-close
      Y2-Data-Array = quotes
      Y2-Data-Array-Row-Count = #row
      Y2-Data-Array-Column-Count = #col2
      Y2-Data-Array-Column-Labels = ('Price')
      Y2-Color-Palette = 'Line'
End-Report
```

Creating Bubble Charts

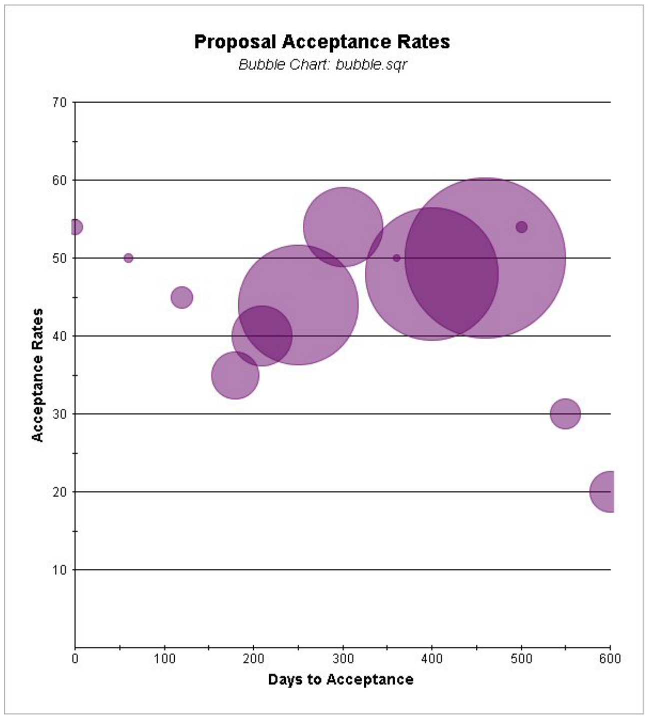
Bubble charts are variations of XY charts, where data points are replaced by bubbles. The bubbles provide a way to display a third variable in the chart. The diameter of each bubble is proportional to the value it represents.

Note:

Bubble charts require that **NewGraphics=True** in the [Default-Settings] section of the SQR.INI file. See “SQR.INI” in Volume 2 of the *Production Reporting Developer's Guide* for more information.

Figure 6 is a sample Bubble chart. The Production Reporting code used to create the chart appears after the chart.

Figure 6 Sample Bubble Chart



```
Begin-Setup
  Declare-Printer Default-ps
    Color = yes
  End-Declare
  Declare-Report Default
    Printer-Type = ps
  End-Declare
  Declare-Layout Default
    Max-Lines = 62
    Left-Margin = 0.10
    Right-Margin = 0.25
  End-Declare
  Create-Array                ! This Array is passed to the Print-Chart
    Name = adm_ratings
    Size = 20    ! Maximum of 20 rows of data
    Field = days:number:1    ! Fields in each row
    Field = rating:number:1
    Field = votes:number:1
! Declare-Chart is not necessary to print a chart. However, it can be used
! as a template and its properties can be modified in each Print-Chart
! that uses this chart name.
! NOTE: Declare-Chart must physically preceede Print-Chart
  Declare-Chart adm_ratings_chart
    Chart-Size = ( 60,40)
    Title = 'Proposal Acceptance Rates'
    Type = bubble
    X-Axis-Label = 'Days to Acceptance'
    Y-Axis-Label = 'Acceptance Rates'
    y-Axis-Min-Value = 0.01
    Legend = no
  End-Declare
End-Setup
```

```
Begin-Report
! Load the array with the Chart Data. Max rows is specified in Create-Array
  Put  0 54 10
  Into adm_ratings(0)
         days(0) rating(0) votes(0)
  Put  60 50 6
  Into adm_ratings(1)
         days(0) rating(0) votes(0)
  Put  120 45 15
  Into adm_ratings(2)
         days(0) rating(0) votes(0)
  Put  180 35 35
  Into adm_ratings(3)
         days(0) rating(0) votes(0)
  Put  210 40 44
  Into adm_ratings(4)
         days(0) rating(0) votes(0)
  Put  250 44 90
  Into adm_ratings(5)
         days(0) rating(0) votes(0)
  Put  300 54 59
  Into adm_ratings(6)
         days(0) rating(0) votes(0)
  Put  360 50 4
  Into adm_ratings(7)
         days(0) rating(0) votes(0)
  Put  400 48 99
  Into adm_ratings(8)
         days(0) rating(0) votes(0)
  Put  460 50 120
  Into adm_ratings(9)
         days(0) rating(0) votes(0)
  Put  500 54 7
  Into adm_ratings(10)
         days(0) rating(0) votes(0)
  Put  550 30 22
  Into adm_ratings(11)
         days(0) rating(0) votes(0)
  Put  600 20 30
  Into adm_ratings(12)
         days(0) rating(0) votes(0)
  Move 5 TO #x
  Move 1 TO #y
  Move 13 TO #row
  Move 3 TO #col
  Create-Color-Palette
    Name = 'Test-Palette'
    Color_1 = (100,0,100)    !Purple
    Color_2 = (255, 50, 100) !Red
    Color_3 = (100,255,30)   !Green
    Color_4 = (0,0,255)      !Blue
    Color_5 = (255,255,255)  !White
! Print the Chart
  Print-Chart adm_ratings_chart (#y, #x)
    Sub-Title = 'Bubble Chart'
    Data-Array-Row-Count = #row
    Data-Array-Column-Count = #col
    Data-Array = adm_ratings
    Color-Palette = 'Test-Palette'
    Legend = no
End-Report
```

Defining Point Labels

Use CREATE-ARRAY to specify point labels as part of a data array. Each chart has seven data points with the exception Polar charts and Radar charts.

Polar charts have two data points.

```
create-array  name=polar_data_with_labels  size=7
  field=label:char ! point label
  field=theta:number:1 ! angle
  field=radius:number:2 ! two series of point
```

Radar charts have three data points.

```
create-array  multi_series_radar_data_with_labels  size=7
  field=label:char ! point label
  field=radius1:number:1 ! series 1 Y-axis values
  field=radius2:number:1 ! series 2 Y-axis values
  field=radius3:number:1 ! series 3 Y-axis values
```

15 Changing Fonts

In This Chapter

Selecting Fonts

Use DECLARE-PRINTER and ALTER-PRINTER to select a font. DECLARE-PRINTER sets the default font for the entire report. ALTER-PRINTER changes the font anywhere in the report and the change remains in effect until the next ALTER-PRINTER.

To set a font for the entire report, use ALTER-PRINTER (which is not printer-specific) at the beginning of the program. For printer-independent reports, the attributes you set with DECLARE-PRINTER only take effect when you print your report with the printer specified with TYPE. To specify a printer at print time, use -PRINTER:xx.

Positioning Text

In Production Reporting, you position text according to a grid. The default is 10 characters per inch and 6 lines per inch, but you can give it another definition with the CHAR-WIDTH and LINE-HEIGHT parameters in DECLARE-LAYOUT.

Character grid and character size function independently. Fonts print in the size set by DECLARE-PRINTER or ALTER-PRINTER, not the size defined by the grid. A character grid is best used for positioning the first character in a string. It can only express the width of a string in terms of the number of characters it contains, not in a linear measurement, such as inches or picas.

When you use a proportionally spaced font, the number of letters printed may no longer match the number of character cells that the text fills. For example, in the sample code given in the next page, the word “Proportionally” fills only 9 cells, although it contains 14 letters.

When you print consecutive text strings, the position at the end of a string may differ from the position Production Reporting assumes according to the grid. As a result, you should concatenate consecutive pieces of text and print them as one.

For example, instead of writing code such as:

```
alter-printer font=5    ! select a proportional font
```

```
print &first_name ()     ! print first name
print ' ' ()             ! print a space
print &last_name ()      ! print the last name
alter-printer font=3     ! restore the font
```

You should write code such as:

```
alter-printer font=5     ! select a proportional font
! concatenate the name
let $full_name = &first_name || ' ' || &last_name
print $full_name ()      ! print the name
alter-printer font=3     ! restore the font
```

`WRAP` and `CENTER` in the `PRINT` command require special consideration when used with proportional fonts. They both calculate the text length based on the character count in the grid, which is not the same as its dimensional width. The use of these options with proportional fonts is explained after the output example.

Program ex22.sqr consists of a list of reminders from the *reminders* table. It is printed in a mix of fonts—Times Roman in two sizes, plus Helvetica bold.

Program ex22.sqr

```
begin-setup
 declare-layout default
   paper-size=(10,11)
 end-declare
end-setup
begin-program
 do main
end-program
begin-procedure main
! Set Times Roman as the font for the report
alter-printer font=5 point-size=12
begin-select
remind_date    (,1,20) edit 'DD-MON-YY'
reminder       (,+1) wrap 60 5
 position (+2)
from reminders
end-select
end-procedure ! main
begin-heading 7
 print $current-date    (1,1) Edit 'DD-MON-YYYY'
 page-number (1,60) 'Page '
 ! Use large font for the title
 alter-printer font=5 point-size=24
 print 'Reminder List'      (3,25)
 ! Use Helvetica for the column headings
 alter-printer font=4 point-size=12
 print 'Date' (6,1) bold
 print 'Reminder' (6,22) bold
 graphic (6,1,66) horz-line
 ! Restore the font
 alter-printer font=5 point-size=12
end-heading
```

The report uses the default layout grid of 10 characters per inch and 6 lines per inch, both for positioning the text and for setting the length of the solid line.

The font is set at the beginning of the main procedure to font 5, which is Times Roman. The point size was not set, so it remains at the default of 12. In the `HEADING` section, its size is set to 24 points to print the title.

The column headings are set to 12-point Helvetica with `ALTER-PRINTER FONT=4 POINT-SIZE=12`. The `BOLD` option in the `PRINT` command specifies that they are printed in bold.

Under the column headings, there is a solid line. Note that it is positioned at line 6, the same as the column headings. Production Reporting draws the solid line as an underline. At the end the `HEADING` section, the font is restored to Times Roman.

Output for *Program ex22.sqr*

```
 30-
JAN-2003
                                     Page 1
                             Reminder List
    ______________________________________________________________
Date             Reminder                       05-MAY-2000
Quarterly contract estimates are due
Friday.                                        Be sure to include
a  reference to possible                                  overage
by Kennedy Inc on the H247 assembly.
   05-MAY-2000                Bill wants list of new equipment planned
during next 6 months. Copy to go to Sandy Hanover in purchasing.
                 05-MAY-2000              Ask Bob to check results of last
HBI mailing.
        05-MAY-2000              Check results of Friday's MSDR test. Send
letter to Sue R.  if positive.
        06-MAY-2000              Final for 3Q budget due tomorrow. Check
that last Q's and last year's Q numbers are accurate. Ask Bob to try
new laser printer, 5 copies.
```

In Production Reporting programs, the report heading is performed after the body. A font change in the heading does not affect the font used in the body of the current page, although it changes the font used in the body of subsequent pages. Make sure to keep track of font changes. You should return fonts to their original settings in the same section in which you change them.

Positioning the title requires careful coding. The `CENTER` option of the `PRINT` command does not account for text size. Instead, position the title by estimating its length. In this case, the title should start 2 1/2 inches from the left margin. The character coordinates are (3,25), which is line 3, character position 25. Remember that the character grid used for positioning assumes 10 characters per inch. Therefore, 25 characters translates to 2 1/2 inches.

Using WRAP

The `WRAP` option in `PRINT` prints the text of the *reminder* column. This option wraps text based on a given width, which is 60 characters in the sample program.

`WRAP` works only on the basis of the width in the character grid. It does not depend on the font.

Text in Times Roman takes about 30-50 percent less room than the text in Courier (the default font, which is a fixed-size font). Thus, a column with a nominal width of 44 characters (the width of the *reminder* column) can hold as many as 66 letters when printed in Times Roman. To be conservative, specify a width of 60.

The `WRAP` argument is the maximum number of lines. Because the *reminder* column in the database is 240 characters wide, at 60 characters per line, no more than five lines are needed. This setting only specifies the maximum number of lines. Production Reporting does not use more lines than necessary.

Production Reporting calculates the maximum number of characters on a line using the page dimensions in `DECLARE-LAYOUT` (the default is 8 1/2 inches wide). In the sample program, 8 1/2 inches minus the inch used in the margins is 7 1/2 inches, or 75 characters at 10 CPI. Printing 60 characters starting from position 22 could exceed this maximum and cause an error or undesirable output. To avoid this error, define the page as wider than it actually is. This definition is given by `PAPER-SIZE=(10,11)` in `DECLARE-LAYOUT`.

16 Writing Printer-Independent Reports

In This Chapter

About Printer-Independent Reports

Printer-independent reports can be run on any printer that Production Reporting supports or distributed electronically. To create a printer-independent report, you must write a program that avoids using any characteristics that are unique to your printer. Complete printer independence may be too restrictive. However, the closer you can get to a truly printer-independent report, the better.

Guidelines for Printer-Independent Reports

Review these guidelines when creating printer-independent reports:

- Programs should not assume or require specific printers and should be free of the following commands: `GRAPHIC FONT` (use `ALTER-PRINTER` instead), `PRINTER-INIT`, `PRINTER-DEINIT`, `USE-PRINTER-TYPE` (except for using this command to select a printer at run time, as demonstrated in Modified Program ex3a.sqr), and the `CODE-PRINTER` and `CODE` arguments in `PRINT`.
- `DECLARE-PRINTER`, `PRINT-DIRECT`, the `CODE` or `CODE-PRINTER` arguments in `PRINT`, and the `SYMBOL-SET` argument in `ALTER-PRINTER` only define behavior when a specific printer is used.
- Reports should be readable on a line printers. Graphics or solid lines printed with the `GRAHPIC` command are not printed on a line printer. Test your graphical report on a line printer.
- Use only a small set of fonts. Font numbers 3, 4, 5 and their boldface versions are the same regardless printer type (except for line printers). Font 3 is Courier, font 4 is Helvetica, and font 5 is Times Roman. On some HP printers, Helvetica may not be available. This reduces the common fonts to fonts 3 and 5.
- Eps-file images can only be printed on PostScript printers. Hpgl-file images can only be printed on HP LaserJet Series 3 or higher or printers that emulate HP PCL at that level.

BMP images can only be printed using Windows. GIF and JPEG images are suitable only for HTML output. `PRINT-IMAGE` and `PRINT-CHART` may not work with older printers that use PostScript Level 1 or HP LaserJet Series II.

Specifying the Printer at Run-time

If your report is printer-neutral and does not specify a printer, you can specify the printer at run time in these ways:

- Use `-PRINTER:xx` to specify the output type:
  - `-PRINTER:LP`—Line-printer output
  - `-PRINTER:PD`—PDF output
  - `-PRINTER:PS`—PostScript output
  - `-PRINTER:HP`—HP LaserJet output
  - `-PRINTER:WP`—Windows output
  - `-PRINTER:HT`—HTML output

  In SQR Production Reporting Studio, enter these command-line flags in the Parameters field of the Run dialog box. If you are using the system shell, enter the following on the command line:

  ```
  sqr test username/password -printer:ps
  ```

  Note:

  -PRINTER:WP sends report output to the default Windows printer. To specify a non-default Windows printer, use -PRINTER:WP:{Printer Name}. {Printer Name} can be the name assigned to a printer; or, if the operating system permits it, the UNC name (i.e.\\Machine\ShareName). For example, to send output to a Windows printer named *NewPrinter*, you could use -PRINTER:WP:NewPrinter. If your printer name has spaces, enclose the entire command in double quotes.

- Use the `USE-PRINTER-TYPE` command.

  In the next example, the bold lines prompt users to select the printer type at run time.

Sample Program for Selecting a Printer at Runtime

```
begin-program
  input $p 'Printer type'    ! Prompt user for printer type
  let $p = lower($p)         ! Convert type to lowercase
  evaluate $p                ! Case statement
  when = 'hp'
  when = 'hplaserjet'        ! HP LaserJet
    use-printer-type hp
    break
  when = 'lp'
  when = 'lineprinter'       ! Line Printer
    use-printer-type lp
    break
```

```
 when = 'ps'
  when = 'postscript'        ! PostScript
    use-printer-type ps
    break
  when-other
    display 'Invalid printer type.'
    stop
  end-evaluate
  do list_customers
end-program
```

In this code, `INPUT` prompts users to enter the printer type. Because `USE-PRINTER-TYPE` does not accept variables as arguments, `EVALUATE` is used to test for the six possible values and set the printer type accordingly.

`EVALUATE` is similar to a switch statement in the C language. It compares a variable to multiple constants and executes the appropriate code.

Part IV

Advanced Production Reporting Programming

In Advanced Production Reporting Programming:

17 Dynamic SQL and Error Checking

In This Chapter

About Dynamic SQL and Error Checking

You can use Production Reporting to vary an SQL statement by using variables in SQL, by using Dynamic SQL, or by using SQL and substitution variables. You can use Production Reporting to do error checking by using `ON-ERROR` procedures, by using commands with `ON-ERROR` options, or by using the `INPUT` command.

Using Variables in SQL

The SQL language supports variables. An SQL statement containing variables is considered *static*. When Production Reporting executes this statement several times, it executes the same statement, even if the values of the variables change. Because SQL only allows variables where literals are allowed (such as in a `WHERE` clause or `INSERT` statement), the database can parse the statement before the values for the variables are given.

Program ex16a.sqr selects customers from a state that the user specifies.

Program ex16a.sqr

```
begin-program
   do list_customers_for_state
end-program
begin-procedure list_customers_for_state
input $state maxlen=2 type=char 'Enter state abbreviation'
let $state = upper($state)
begin-select
name (,1)
   position (+1)
from customers
where state = $state
end-select
```

```
end-procedure ! list_customers_for_state
```

Note the use of the *$state* variable in the `SELECT` paragraph. When you use a variable in an SQL statement in Production Reporting, the SQL statement sent to the database contains that variable. Production Reporting "binds" the variable before the SQL is executed. In many cases, the database only needs to parse the SQL statement once. The only thing that changes between executions of the `SELECT` statement is the value of the variable. This is the most common example of varying a `SELECT` statement.

In this program, `INPUT` prompts users to enter the value of *state*. `MAXLEN` and `TYPE` check the input, ensuring that users enter strings of no more than two characters. If the entry is incorrect, `INPUT` prompts again.

The program converts the contents of *$state* to uppercase, which allows users to input the state without worrying about the case. In the example, state is uppercase in the database. The example shows the `LET` command used with the Production Reporting *upper* function.

You could let the SQL convert to uppercase. You would do so by using *where state = upper ($state)* if you are using Oracle or Sybase or by using *where state = ucase($state)* if you are using another database. However, Production Reporting allows you to write database-independent code by moving the use of such SQL extensions to the Production Reporting code.

When you run this program, you must specify one of the states included in the sample data for the program to return any records. At the prompt, enter IN, MI, NH, NJ, NM, NY, or OH. If you enter NY (the state where most of the customers in the sample data reside), Production Reporting generates the following output.

Output for *Program ex16a.sqr*

```
Alfred E Newman & Company
Eliot Richards
Harriet Bailey
Kate's Out of Date Dress Shop
Corks and Bottles, Inc.
```

Dynamic SQL

In some cases, you may find the restriction against using variables where only literals are allowed too limiting. In the following example, the ordering of the records changes based on use selection. The program runs the `SELECT` twice. The first time, the first column is called *name* and the second column is called *city*, and the program sorts the records by *name* with a secondary sort by *city*. The second time, the first column is the *city* and the second is *name*, and the program sorts by *city* with a secondary sort by *name*. The first `SELECT` statement will therefore be:

```
select name, city
from customers
order by name, city
```

The second `SELECT` statement is:

```
select city, name
from customers
```

```
order by city, name
```

As you can see, the statements differ. Production Reporting constructs the statement each time before executing it. This technique is called *dynamic SQL*, and it is illustrated in *Program ex16b.sqr*. To take full advantage of the error-handling procedure, use the command-line flag -CB.

Program ex16b.sqr

```
begin-program
   let $col1 = 'name'
   let $col2 = 'city'
   let #pos = 32
   do list_customers_for_state
   position (+1)
   let $col1 = 'city'
   let $col2 = 'name'
   let #pos = 18
   do list_customers_for_state
end-program
begin-procedure give_warning
   display 'Database error occurred'
   display $sql-error
end-procedure ! give_warning
begin-procedure list_customers_for_state
   let $my_order = $col1 || ',' || $col2
begin-select on-error=give_warning
[$col1] &column1=char (,1)
[$col2] &column2=char (,#pos)
   position (+1)
from customers
order by [$my_order]
end-select
end-procedure ! list_customers_for_state
```

When you use variables in SQL statements to replace literals, make them *dynamic variables* by enclosing them in square brackets. For example, when you use the dynamic variable [*$my_order*] in the ORDER BY clause of the SELECT statement, Production Reporting places the text from the variable *$my_order* in that statement. Each time the statement is executed, if the text changes, a new statement is compiled and executed.

Other dynamic variables used are [*$col1*] and [*$col2*]. They substitute the names of the columns in the SELECT statement. The variables &*column1* and &*column2* are column variables.

You can use dynamic variables to produce reports like this one. Note that the data in the first half of the report is sorted differently than the data in the second half.

The error-handling procedure *give_warning* is discussed in "SQL and Substitution Variables" on page 128.

Output for *Program ex16b.sqr*

```
Alfred E Newman & Company      New York
Clair Butterfield              Teaneck
Corks and Bottles, Inc.        New York
```

```
Eliot Richards                Queens
Gregory Stonehaven            Everretsville
Harold Alexander Fink         Davenport
Harriet Bailey                Mamaroneck
Harry's Landmark Diner        Miningville
Isaiah J Schwartz and Company Zanesville
Jerry's Junkyard Specialties  Frogline
Joe Smith and Company         Big Falls
Kate's Out of Date Dress Shop New York
Quentin Fields                Cleveland
Sam Johnson                   Bell Harbor
Bell Harbor      Sam Johnson
Big Falls        Joe Smith and Company
Cleveland        Quentin Fields
Davenport        Harold Alexander Fink
Everretsville    Gregory Stonehaven
Frogline         Jerry's Junkyard Specialties
Mamaroneck       Harriet Bailey
Miningville      Harry's Landmark Diner
New York         Alfred E Newman & Company
New York         Corks and Bottles, Inc.
New York         Kate's Out of Date Dress Shop
Queens           Eliot Richards
Teaneck          Clair Butterfield
Zanesville       Isaiah J Schwartz and Company
```

SQL and Substitution Variables

Production Reporting uses substitution variable values to complete the SELECT statement at compile time. Because the SELECT statement is complete at compile time, Production Reporting can check its syntax before execution begins. From this point on, the value of {*my_order*} cannot change and the SQL statement is considered static.

In *Program ex16c.sqr*, the ASK command in the SETUP section prompts users at compile time. User-entered values are placed in *substitution variables*, which can substitute commands, arguments, or parts of SQL statements at compile time. This example is less common, but it demonstrates the difference between compile-time and run-time substitutions.

Program ex16c.sqr

```
begin-setup
  ask my_order 'Enter the column name to sort by (name or city)'
end-setup
begin-program
   do list_customers_for_state
end-program
begin-procedure give_warning
   display 'Database error occurred'
   display $sql-error
end-procedure ! give_warning
begin-procedure list_customers_for_state
begin-select on-error=give_warning
name (,1)
city (,32)
```

```
   position (+1)
from customers
order by {my_order}
end-select
end-procedure ! list_customers_for_state
```

In this example, ASK prompts users for the value of the substitution variable {*my_order*}, which sorts the output. If the argument is passed on the command line, there is no prompt. When you run this program, enter name, city, or both (in either order and separated by a comma). The program produces a report sorted accordingly.

You can only use ASK in the SETUP section.Production Reporting executes ASK commands at compile time before program execution begins. Therefore, ASK commands are executed before INPUT commands.

INPUT is more flexible than ASK. You can use INPUT inside loops, validate the length and type of data input, and prompt again if it is not valid. For an example, see "Using Variables in SQL" on page 125.

ASK can be more powerful. Substitution variables set in an ASK command let you modify commands that are normally quite restrictive. The following code shows this technique.

```
begin-setup
   ask hlines 'Number of lines for heading'
end-setup
begin-program
  print 'Hello, World!!' (1,1)
end-program
begin-heading {hlines}
   print 'Report Title' () center
end-heading
```

In this example, the substitution variable {*hlines*} defines the number of lines in the heading. BEGIN-HEADING normally expects a literal and does not allow a run-time variable. When a substitution variable is used, its value is modified at compile time.

For more information on ASK and INPUT, see Chapter 29, "Compiling Programs and Using Production Reporting Execute ."

SQL Error Checking

Production Reporting checks and reports database errors for SQL statements. When an Production Reporting program is compiled, Production Reporting checks the syntax of the SELECT, UPDATE, INSERT, and DELETE. SQL statements in the program. Any SQL syntax error is detected and reported at compile time, before the execution of the report begins.

With dynamic SQL, Production Reporting cannot check the syntax until run time. In that case, the content of the dynamic variable is used to construct the SQL statement, which can allow syntax errors to occur at run time. Errors could occur if the dynamic variables selected or used in WHERE or ORDER BY clauses were incorrect.

Production Reporting traps and reports run-time errors and aborts the program. To change this default behavior, use the ON-ERROR argument the BEGIN-SELECT or BEGIN-SQL paragraphs.

Note:

Production Reporting invokes ON-ERROR when it safely can. If Production Reporting can recover from a database error, users given the chance to fix the error. If Production Reporting cannot recover from a database error, it exits.

```
begin-select on-error=give_warning
[$col1] &column1=char (,1)
[$col2] &column2=char (,#pos)
   position (+1)
from customers
order by [$my_order]
end-select
```

In this example, if a database error occurs, Production Reporting invokes a procedure called *give_warning* instead of reporting the problem and aborting. Write this procedure as follows:

```
begin-procedure give_warning
   display 'Database error occurred'
   display $sql-error
end-procedure ! give_warning
```

This procedure displays the error message but does not abort program execution. Instead, execution continues at the statement immediately following the SQL or `SELECT` paragraph. *`$sql-error`* is a special Production Reporting reserved variable that contains error message text from the database and is automatically set by Production Reporting after a database error occurs.

Production Reporting has numerous reserved, or predefined, variables. For example, *`$sqr-program`* contains the name of the currently-running program, *`$username`* contains the current user name, and *`#page-count`* contains the current page number.

For a complete list of reserved variables, see “Production Reporting Reserved Variables” in Volume 2 of the *Production Reporting Developer's Guide*.

18 Procedures, Argument Passing, and Local Variables

In This Chapter

A Sample Program

The sample code in this chapter shows a procedure that spells out a number and a program for printing checks that uses this procedure. When printing checks, you normally must spell out the dollar amount.

In the sample code, *Spell.inc*, it is assumed that the checks are preprinted and that our program only prints such items as the date, name, and amount.

Procedures

Production Reporting procedures containing variables that are visible throughout the program are called *global procedures*. These procedures can also directly reference any program variable.

Procedures that take arguments, such as the *spell_number* procedure in this chapter's check-printing sample code, are *local procedures*. In Production Reporting, any procedure that takes arguments is automatically considered local.

Variables introduced in local procedures are only readable inside the *Spell.inc* procedure. This useful feature avoids name collisions. The *spell_number* procedure is an include file because other reports may also want to use it.

Local Variables

When you create library procedures that can be used in many programs, make them local. Then, if a program has a variable with the same name as a variable used in the procedure, there will not be a collision.Production Reporting treats the two variables separately.

Declare a procedure as local even if it does not take any arguments. Simply place the keyword `LOCAL` after the procedure name in `BEGIN-PROCEDURE`.

To reference a global variable from a local procedure, insert an underscore between the prefix character (#, $, or &) and the variable name. Use the same technique to reference reserved variables such as *#current-line*. These variables are global, and can be referenced from local procedures.

Production Reporting supports recursive procedure calls, but maintains only one copy of local variables. Procedures do not allocate new instances of local variables on a stack, as C or Pascal would.

Argument Passing

Procedure arguments are treated as local variables. Arguments can be either numeric, date, or text variables or strings. If an argument is preceded with a colon, its value is passed back to the calling procedure.

In the following example, *spell_number* takes two arguments. The first argument is the check amount. This argument is a number, and the program passes it to the procedure. There is no need for the procedure to pass it back.

The second argument is the result that the procedure passes back to the calling program. We precede this variable with a colon, thus the value of this argument is copied back at the end of the procedure. The colon is only used when the argument is declared in `BEGIN-PROCEDURE`.

Look at the following code. It is not a complete program, but it is the *spell_number* procedure stored in *spell.inc*. The check-printing program includes this code using an `#INCLUDE` command.

File spell.inc

```
begin-procedure spell_number(#num,:$str)
   let $str = ''
   ! break the number to it's 3-digit parts
   let #trillions  = floor(#num / 1000000000000)
   let #billions   = mod(floor(#num / 1000000000),1000)
   let #millions   = mod(floor(#num / 1000000),1000)
   let #thousands  = mod(floor(#num / 1000),1000)
   let #ones       = mod(floor(#num),1000)
   ! spell each 3-digit part
   do spell_3digit(#trillions,'trillion',$str)
   do spell_3digit(#billions,'billion',$str)
   do spell_3digit(#millions,'million',$str)
   do spell_3digit(#thousands,'thousand',$str)
   do spell_3digit(#ones,'',$str)
end-procedure ! spell_number
begin-procedure spell_3digit(#num,$part_name,:$str)
   let #hundreds = floor(#num / 100)
   let #rest     = mod(#num,100)
   if #hundreds
      do spell_digit(#hundreds,$str)
      concat 'hundred ' with $str
   end-if
   if #rest
   do spell_2digit(#rest,$str)
   end-if
   if #hundreds or #rest
      if $part_name != ''
         concat $part_name with $str
         concat ' ' with $str
      end-if
   end-if
end-procedure ! spell_3digit
begin-procedure spell_2digit(#num,:$str)
   let #tens     = floor(#num / 10)
   let #ones     = mod(#num,10)
   if #num < 20 and #num > 9
      evaluate #num
       when = 10
         concat 'ten ' with $str
         break
       when = 11
         concat 'eleven ' with $str
         break
   when = 12
         concat 'twelve ' with $str
         break
      when = 13
         concat 'thirteen ' with $str
         break
       when = 14
         concat 'fourteen ' with $str
         break
     when = 15
         concat 'fifteen ' with $str
         break
       when = 16
         concat 'sixteen ' with $str
         break
       when = 17
         concat 'seventeen ' with $str
         break
       when = 18
         concat 'eighteen ' with $str
         break
       when = 19
         concat 'nineteen ' with $str
         break
      end-evaluate
   else
      evaluate #tens
       when = 2
         concat 'twenty' with $str
         break
       when = 3
         concat 'thirty' with $str
         break
       when = 4
         concat 'forty' with $str
         break
        when = 5
         concat 'fifty' with $str
         break
       when = 6
         concat 'sixty' with $str
         break
      when = 7
         concat 'seventy' with $str
         break
       when = 8
         concat 'eighty' with $str
         break
       when = 9
         concat 'ninety' with $str
         break
       end-evaluate
     if #num > 20
          if #ones
             concat '-' with $str
          else
             concat ' ' with $str
          end-if
       end-if
       if #ones
          do spell_digit(#ones,$str)
       end-if
   end-if
end-procedure ! spell_2digit
begin-procedure spell_digit(#num,:$str)
   evaluate #num
      when = 1
         concat 'one ' with $str
         break
      when = 2
         concat 'two ' with $str
         break
      when = 3
         concat 'three ' with $str
         break
      when = 4
         concat 'four ' with $str
         break
      when = 5
         concat 'five ' with $str
         break
      when = 6
         concat 'six ' with $str
         break
      when = 7
         concat 'seven ' with $str
         break
      when = 8
         concat 'eight ' with $str
         break
      when = 9
         concat 'nine ' with $str
         break
   end-evaluate
end-procedure ! spell_digit
```

The result argument is reset in the procedure, because the program begins with an empty string and keeps concatenating the parts of the number to it. The program only supports numbers up to 999 trillion.

The number is broken into its three-digit parts: trillions, billions, millions, thousands, and ones. Another procedure spells out the three-digit numbers such as “one hundred twelve.” Note that the word *and* is inserted only between dollars and cents, but not between three-digit parts. This format is common for check printing in dollars.

Note the use of math functions such as `floor` and `mod`.Production Reporting has a rich set of functions that can be used in expressions. These functions are described under the `LET` command in Volume 2 of the *Production Reporting Developer's Guide*

The series of `EVALUATE` commands in the number-spelling procedures are used to correlate the numbers stored in the variables with the strings used to spell them out.

Program ex17a.sqr is the full program that prints the checks.

Program ex17a.sqr

```
#include 'spell.inc'
begin-setup
  declare-layout default
  end-declare
end-setup
begin-program
  do main
end-program
begin-procedure main
  alter-printer font=5 point-size=15
begin-select
name                               &name
sum(d.price * c.quantity) * 0.10   &refund
  do print_check(&refund)
from  customers a, orders b,
      ordlines c, products d
  where a.cust_num = b.cust_num
  and   b.order_num = c.order_num
  and   c.product_code = d.product_code
 group by name
 having sum(d.price * c.quantity) * 0.10 >= 0.01
end-select
end-procedure ! main
begin-procedure print_check(#amount)
  print $_current-date (3,45) edit 'DD-Mon-YYYY'
  print &_name (8,12)
  move #amount to $display_amt 9,999,990.99
  ! enclose number with asterisks for security
  let $display_amt = '**' || ltrim($display_amt,' ') || '**'
  print $display_amt (8,58)
  if #amount < 1.00
    let $spelled_amount = 'Zero dollars '
  else
    do spell_number(#amount,$spelled_amount)
    let #len = length($spelled_amount)
    ! Change the first letter to uppercase
    let $spelled_amount = upper(substr($spelled_amount,1,1))
                 || substr($spelled_amount,2,#len - 1)
    concat 'dollars ' with $spelled_amount
  end-if
  let #cents = round(mod(#amount,1) * 100, 0)
  let $cents_amount = 'and ' || edit(#cents,'00') || ' cents'
  concat $cents_amount with $spelled_amount
  print $spelled_amount (12,12)
  print 'Rebate'        (16,12)
  print ' ' (20)
  next-listing need=20
end-procedure ! print_check
```

The *main* procedure starts by setting the font to 15-point Times Roman. The SELECT paragraph is a join of several tables. (A join is created when you select data from multiple database tables in the same SELECT paragraph.) The *customers* table has the customer's name. The program joins it with the *orders* and *ordlines* tables to get the customer's order details. It also joins with the *products* table for the price.

The following expression adds all of the customer's purchases and calculates a 10 percent rebate:

```
sum(d.price * c.quantity) * 0.10
```

The statement groups the records by the customer name, one check per customer. This is done with the following clause:

```
group by name
having sum(d.price * c.quantity) * 0.10 >= 0.01
```

The HAVING clause eliminates checks for less than 1 cent.

The procedure *print_check* is a local procedure. Note the way it references the date and customer name with &_*current-date* and &_*name*, respectively.

19 Multiple Reports

This chapter explains how to create multiple reports from one program. You can create multiple reports based on common data, selecting the database records only once and creating reports simultaneously.

The alternative—writing separate programs for the reports—would require a database query for each report. Repeated queries are costly because database operations are often the most resource-consuming or time-consuming part of creating a report.

Program ex18a.sqr writes multiple reports with different layouts, heading, and footing sections. The sample program prints three reports—the labels from Chapter 10, "Printing Mailing Labels," the form letter from Chapter 11, "Creating Form Letters,"and the listing report from Chapter 4, "Selecting Data." All three reports are based on identical data.

Program ex18a.sqr

```
#define MAX_LABEL_LINES        10
#define LINES_BETWEEN_LABELS   3
begin-setup
  declare-layout labels
    paper-size=(10,11)   left-margin=0.33
  end-declare
  declare-layout form_letter
  end-declare
  declare-layout listing
  end-declare
  declare-report labels
    layout=labels
  end-declare
  declare-report form_letter
    layout=form_letter
  end-declare
  declare-report listing
    layout=listing
  end-declare
end-setup
begin-program
  do main
end-program
begin-procedure main
  do init_mailing_labels
begin-select
name
addr1
addr2
city
```

```
state
zip
  move &zip to $zip xxxxx-xxxx
phone
  do print_label
  do print_letter
  do print_listing
from customers
end-select
  do end_mailing_labels
end-procedure ! main
begin-procedure init_mailing_labels
  let #label_count = 0
  let #label_lines = 0
  use-report labels
  columns 1 29 57  ! enable columns
  alter-printer font=5 point-size=10
end-procedure ! init_mailing_labels
begin-procedure print_label
  use-report labels
  print &name   (1,1,30)
  print &addr1  (2,1,30)
  let $last_line = &city || ', ' || &state || ' ' || $zip
  print $last_line (3,1,30)
  next-column at-end=newline
  add 1 to #label_count
  if #current-column = 1
    add 1 to #label_lines
    if #label_lines = {MAX_LABEL_LINES}
      new-page
      let #label_lines = 0
    else
      next-listing no-advance skiplines={LINES_BETWEEN_LABELS}
    end-if
  end-if
end-procedure ! print_label
begin-procedure end_mailing_labels
  use-report labels
  use-column 0  ! disable columns
  new-page
  print 'Labels printed on ' (,1)
  print $current-date ()
  print 'Total labels printed = ' (+1,1)
  print #label_count () edit 9,999,999
end-procedure ! end_mailing_labels
begin-procedure print_letter
use-report form_letter
begin-document (1,1)
&name
&addr1
&addr2
@city_state_zip
.b
.b
                                                $current-date
Dear Sir or Madam:
.b
```

```
    Thank you for your recent purchases from ACME Inc. We would like to
tell you about our limited time offer. During this month, our entire
inventory is marked down by 25%. Yes, you can buy your favorite merchandise
and save too.
    To place an order simply dial 800-555-ACME. Delivery is free too, so
don't wait.
.b
.b
                                Sincerely,
                                Clark Axelotle
                                ACME Inc.
end-document
position () @city_state_zip
print &city  ()
print ', '   ()
print &state ()
print ' '    ()
move &zip to $zip xxxxx-xxxx
print $zip   ()
new-page
end-procedure ! print_letter
begin-heading 4 for-reports=(listing)
print 'Customer Listing' (1) center
   print 'Name' (3,1)
   print 'City' (,32)
   print 'State' (,49)
   print 'Phone' (,55)
end-heading
begin-footing 1 for-reports=(listing)
   ! Print "Page n of m" in the footing
   page-number (1,1) 'Page '
   last-page   () ' of '
end-footing
begin-procedure print_listing
  use-report listing
  print &name (,1)
  print &city (,32)
  print &state (,49)
  print &phone (,55)
  position (+1)
end-procedure ! print_listing
```

The SETUP section defines the layouts and the reports that use the layouts. The labels report requires a different layout from the default. The other two reports use a layout that is identical to the default layout. It would be possible to save the last layout declaration and use the form-letter layout for the listing. However, unless there is a logical reason why the two layouts should be identical, it is better to keep separate layouts. The name of the layout indicates which report uses it.

The *main* procedure performs the SELECT. It is only performed once and includes all columns for all reports. The *phone* column is only used in the listing report and the *addr2* column is only used in the form-letter report. The other columns are used in multiple reports.

For each record selected, three procedures are executed. Each procedure processes one record for its corresponding report. The *print_label* procedure prints one label, *print_letter* prints one letter, and *print_listing* prints one line into the listing report. Each procedure begins by setting the Production Reporting printing context to its corresponding report. Production Reporting sets the printing context with USE-REPORT.

You can define HEADING and FOOTING sections for each report. This example only defines the heading and footing for the listing report, because the other two reports do not use them. The FOR-REPORTS argument in BEGIN-HEADING and BEGIN-FOOTING specifies the report name. The parentheses are required. USE-REPORT is not needed in the heading or the footing. The report is implied by FOR-REPORTS.

Because this program creates output with proportional fonts, you must run it with -KEEP or -PRINTER:xx. (If you run the report with Production Reporting , you can omit -KEEP. See Chapter 10, “Printing Mailing Labels.”)

If OUTPUT-FILE_MODE=LONG in SQR.INI (which is the default), running Example produces three output files that match the output files for *Modified Program ex9a.sqr*, *Program ex 10a.sqr* and *Program ex3a.sqr* respectively. These output files have *.lis extensions. If you specify -KEEP, the output files have *.spf. (See “[Default-Settings] Section” in the chapter titled “SQR.INI” in Volume 2 of the *Production Reporting Developer's Guide* for information on setting OUTPUT-FILE-MODE to LONG.)

In SQR Production Reporting Studio, the Viewer window opens automatically after you run the programs, but only the first output file, ex18a.spf, is displayed. View the other output files by selecting File, then Open.

20 Using DML and DDL

In This Chapter

SQL Statements

Although `SELECT` may be the most common SQL statement, you can also perform other SQL commands in Production Reporting. For example:

- If the program prints documents such as checks, tickets, or invoices, update the database using a SQL UPDATE statement to indicate the document printed.
- Use Production Reporting to load data into the database, read and write external files, and construct records. Production Reporting can insert records into the database using an SQL `INSERT` statement.
- To hold intermediate results in a temporary database table, create two SQL paragraphs in your Production Reporting program (`CREATE TABLE` and `DROP TABLE`) to create this table at the beginning of the program and drop the table at the end.

Using BEGIN-SQL

All SQL statements other than a `SELECT` statement must use the `BEGIN-SQL` paragraph.

Program ex19a.sqr loads data from an external file into the database. It demonstrates two important features of Production Reporting—handling external files and performing database inserts.

Program ex19a.sqr

```
begin-setup
  begin-sql  on-error=skip ! table may already exist
    create table customers_ext (
      cust_num int not null,
      name     varchar (30),
      addr1    varchar (30),
      addr2    varchar (30),
      city     varchar (16),
      state    varchar (2),
      zip      varchar (10),
      phone    varchar (10),
      tot      int
      )
  end-sql
end-setup
begin-program
  do main
end-program
begin-procedure main
#if {sqr-database} = 'Sybase'
  begin-sql
    begin transaction
  end-sql
#endif
  encode '<009>' into $sep
  open 'ex11a.lis' as 1 for-reading record=160:vary
  read 1 into $rec:160  ! skip the first record, column headings
  while 1
    read 1 into $rec:160
    if #end-file
      break
    end-if
    unstring $rec by $sep into $cust_num $name
        $addr1 $addr2 $city $state $zip $phone $tot
    move $cust_num to #cust_num
    move $tot to #tot
    begin-sql
      insert into customers_ext (cust_num, name,
         addr1, addr2, city, state, zip, phone, tot)
      values
         (#cust_num, $name, $addr1, $addr2, $city,
          $state, $zip, $phone, #tot)
    end-sql
  end-while
#if {sqr-database} = 'Sybase'
  begin-sql
    commit transaction
  end-sql
#else
#if {sqr-database} <> 'Informix'
  begin-sql
    commit
  end-sql
#endif
#endif
  close 1
end-procedure ! main
```

The program starts by creating the table *customers_ext*. If the table exists, an error message appears. To ignore this error message, use ON-ERROR=SKIP. (See Chapter 17, "Dynamic SQL and Error Checking.")

The program reads the records from the file and inserts each record into the database by using an INSERT statement inside a BEGIN-SQL paragraph. The input file format is one record per line, with each field separated by the separator character. When the end of the file is encountered (*if #end-file*), the program branches out of the loop. Note that *#end-file* is a Production Reporting reserved variable. For a complete list of reserved variables, see Volume 2 of the *Production Reporting Developer's Guide*.

The last step is to commit the changes to the database and close the file. You do this with a SQL `COMMIT` statement inside a `BEGIN-SQL` paragraph. Alternatively, you can use the Production Reporting `COMMIT` command. For Oracle databases, we recommend you use `COMMIT`.

The code may be database-specific. If you are using Informix, for example, and your database was created with transaction logging, you must add `BEGIN WORK` and a `COMMIT WORK`, much like the Sybase example of `BEGIN TRANSACTION` and `COMMIT TRANSACTION`.

21 Working with Comma Separated Files (CSV)

In This Chapter

Declaring a Connection to a CSV Data Source

To start accessing data from the CSV Data Source, `DECLARE-CONNECTION` must be established to a registered data source.

➤ To establish a `DECLARE-CONNECTION`:

1. **Enter Declare-Connection followed by a connection_name_literal *CSV*.**
2. **Enter DSN, this is the logical data source name as recorded in the DDO Registry.**

   (User and Password are associated with CSV data sources)

```
declare-connection CSV
dsn=CSVsource
end-declare
```

Specifying a Separator Value for CSV File Generation

By default, comma separated value (CSV) files created by Production Reporting use the comma as a separator or delimiter. You can specify an alternate delimiter character with the CSVSeparator value in the [Default-Settings] section of the SQR.INI file.

Entry	Value	Description
CSVSeparator	Comma | Semicolon |Space | Tab	Specifies the character used as a delimiter when creating CSV files **Note**: If the CSVSeparator setting is missing from the SQRINI file, the default value of Comma is used

For example, the following setting causes Production Reporting to use the Tab character as the delimiter when creating CSV files:

```
[Default-Settings]

CSVSeparator=Tab
```

Note:

The SQR.INI setting to specify the CSVSeparator as a semicolon or a space is only supported with the Production Reporting Server. Using this setting in Oracle Enterprise Performance Management Workspace, Fusion Edition is not recommended and may create corrupted BQD files.

Viewing CSV Metadata

When creating queries, it is often helpful to view the structure of the CSV file that you are querying. You can browse a CSV file's *metadata* (information about the file's structure) by running the DDO Query Editor and selecting a schema in *Schema View* and viewing its selectable column list.

Creating and Executing MD Queries

You construct queries in the same manner you access relational databases. You can choose a sample script from the `SAMPLES` directory and run or modify it, or construct your own. The scripts in the `SAMPLES` directory are included when the Production Reporting DDO port is installed. You can edit this file with a text editor or create files of your own. To properly access a CSV datasource, a Data Object must be defined. The data object is declared after `BEGIN-EXECUTE` and before `BEGIN-SELECT`. For CSV queries, use the DDO `GetData` paradigm for data access.

Program Sample for Executing MD Queries

```
Begin-Execute
Connection=CSV
GetData='customer.csv' (Data Object)
Begin-Select
Customer_num        type=number   (+1,1)
Name        type=char (,11)
phone         type=char (,41)
addr_line1          type=char (+1,11)
addr_line2          type=char (+1,11)
city         type=char (+1,11)
state         type=char (+1,11)
zip         type=number (,41)
From Rowsets=(1)
End-Select
End-Execute
```

22 Retrieving BINARY Column Data

In This Chapter

Defining that a Variable or Column Supports BINARY Data

To specify that an Production Reporting variable supports BINARY data, use the following syntax in either the `BEGIN-SETUP` section or within a `LOCAL` procedure:

```
DECLARE-VARIABLE
```

`BINARY $Binary`

```
END-DECLARE
```

To specify that an Production Reporting column supports BINARY data, use the following syntax:

```
BEGIN-SELECT
```

`column_name $column_name=BINARY`

```
END-SELECT
```

Defining How to Treat BINARY Data

The SQR.INI file includes two keywords in the [Default-Settings] section that you can use to define how to treat BINARY data.

Entry	Value	Description
TreatBinaryColumnAsText	TRUE | FALSE	Defines whether to treat a BINARY column as a TEXT column The default is True
ImageCompression	0 - 9	Defines the compression level when the PRINT-IMAGE command references a BINARY variable

Entry	Value	Description
		The default is 6

Converting Between BINARY and TEXT

Use the LET command functions to convert information between BINARY and TEXT.

Function	Description
tohex	Accepts a BINARY variable and returns a string composed of uppercase hexadecimal characters that represents the data *Each byte of BINARY data consists of two hexadecimal characters* **Syntax:** *dst_var* = tohex(*source_value*) ● *source_value* = binary literal, column, variable, or expression ● *dst_var* = text variable **Example:** let $hexchars = tohex($vargraphic)
fromhex	Accepts a TEXT variable that contains a string of hexadecimal characters (case insensitive) and returns a BINARY variable *Each byte of BINARY data consists of two hexadecimal characters* **Syntax:** *dst_var* = fromhex(*source_value*) ● *source_value* = text literal, column, variable, or expression ● *dst_var* = binary variable **Example:** let $image = fromhex($hexchars)

Processing External Files

Use the LET command function to facilitate the processing of external files.

Function	Description
filesize	Accepts the name of an external file and returns the number of bytes it contains If the file size cannot be determined, a value of -1 is returned **Syntax:** *dst_var* = filesize(*source_value*) ● *source_value* = text literal, column, variable, or expression ● *dst_var* = decimal, float, or integer variable **Example:** let #size = filesize($file)

Production Reporting Commands that Support BINARY Data

The following commands support BINARY data. Any command not in this list produces an error when it references a BINARY variable, column, or literal.

Note:

CREATE-ARRAY, DECLARE-VARIABLE, and OPEN include additional syntax to support BINARY data. (The additional syntax appears in bold.) The other commands in the list do not require additional syntax to support BINARY data.

- CONCAT
- CREATE-ARRAY

```
CREATE-ARRAY NAME=array_name SIZE=nn

[EXTENT=nn]

{FIELD=name:type[:occurs]
[={init_value_txt_lit|_num_lit|_binary_lit}]}...
```

- DECLARE-VARIABLE

```
DECLARE-VARIABLE

[DEFAULT-NUMERIC={DECIMAL[(prec_lit)]|FLOAT|INTEGER}]

[DECIMAL[(prec_lit)]num_var[(prec_lit)][num_var
[(prec_lit)]]...]

[FLOAT num_var[num_var]...]

[DATE date_var[date_var]...]

[INTEGER num_var[num_var]...]

[TEXT string_var[string_var]...]

[BINARY binary_var[binary_var]...]

END-DECLARE
```

- DO
- EVALUATE
- EXECUTE (DB2, Sybase, and Oracle only)
- EXTRACT
- GET
- IF
- LET
- MOVE
- OPEN

```
OPEN {filename_lit|_var|_col} AS
{filenum_num_lit|_var|_col}
{FOR-READING|FOR-WRITING|FOR-APPEND}
{RECORD=length_num_lit|_var|_col[:FIXED|:FIXED_NOLF
|:VARY|:BINARY]}]
[STATUS=num_var]]
```

```
[ENCODING={_var|_col|ASCII|ANSI|SJIS|JEUC|EBCDIC| EBCDIK290|EBCDIK1027|
UCS-2|UTF-8|others... }]
```

- **PRINT CODE-PRINTER**
- **PRINT-DIRECT**
- **PRINT-IMAGE**

  When a BINARY variable is referenced, the contents are used as the source. When a TEXT variable is referenced, the contents refer to an external file.
- **PUT**
- **READ**
- **STRING**
- **WHILE**
- **WRITE**

23 Working with Multi-Dimensional Data Sources (OLAP)

In This Chapter

Declaring a Connection to an OLAP Server

To access data from MD Data Source, DECLARE-CONNECTION must be established to a registered data source.

➤ To establish a DECLARE-CONNECTION:

1. Enter **Declare-Connection** followed by a connection_name_literal 'OLAP'.
2. Enter **DSN**, the logical data source name as recorded in the DDO Registry.
3. Enter **user** and **password**.
4. Enter **set-member** parameter.

```
Declare-Connection OLAP
dsn=MSOLAP
user=Administrator
password=administrator
set-members=('product','all products.drink.alcoholic beverages.beer and
wine','time','2002.Q1' )
set-levels= ('product', 2)
set-generations= ('product', 5)
End-Declare
```

Viewing Cube Metadata

When you create MD queries to use with multidimensional databases, it is often helpful to view the structure of the cube you are querying.

You can browse a cube's metadata (information about the schema's structure) by running the DDO TestTool and selecting a schema.

Select the schema in *Schema View*. Explore the schema's dimensions and measures in the tree view and member panes below the list.

Creating and Executing MD Queries

You construct MD queries in the same manner that relational databases are accessed. You can choose sample scripts from the `SAMPLES` directory and run or modify it, or construct your own. The scripts in the `SAMPLES` directory are included when the Production Reporting DDO port is installed. You can edit this file with a text editor or create files of your own. To properly access an MD datasource, a Catalog Schema and Data Object must be defined after the `BEGIN-EXECUTE` and before the `BEGIN-SELECT` declaration of the schema and object. For MD queries, the DDO `GetData` paradigm is used for data access.

Measures

Measures are the numeric data of primary interest to MD users. Some common measures are sales, cost, expenditures, and production count. Measures are aggregations stored for quick retrieval by users querying cubes. Each measure is stored in a column in a fact table in the data warehouse. A measure can contain multiple columns combined in an expression. For example, the Profit measure is the difference of two numeric columns: Sales and Cost.

To select a measure column, the format is 'measure', dot, 'measure name' (measure.profit). This is regardless of the name used by the data source to declare measures.

Column Order

The order in which the dimension columns are presented determines the order of the data displayed for multiple rowset queries. The following sample program shows Profit and Store's Sales reports for the selected time period and product. If the order of the time and product dimensions are reversed, then a report would be produced for each selected time period and product. (A greatly different report).

Program Sample for Defining Column Order

```
Begin-Execute
Connection=OLAP
Schema='FoodMart'
GetData='Sales'
Begin-Select
Time             (+1,1)
Product              ( ,15)
Measures.Profit             ( ,45) edit 999999.99
"Measures.Store Sales"               ( ,60) edit 999999.99
From Rowsets=(1)
```

```
End-Select
End-Execute
```

Dimensions, Levels, and Hierarchies

Use the following arguments under the `ALTER-CONNECTION` command as you work the dimensions, levels, and hierarchies in your data.

- `SET-GENERATIONS`—Specifies the dimension hierarchy for the previously-declared dimension.
- `SET-LEVELS`—Extends the dimension hierarchy for the previous-declared dimension.
- `SET-MEMBERS`—Returns the set of members in a dimension, level, or hierarchy whose name is specified by a string.

See "`ALTER-CONNECTION`" in Volume 2 of the *Production Reporting Developer's Guide* for more information on these arguments.

24

Working with Dates

In This Chapter

Dates in Production Reporting

Production Reporting has powerful capabilities in date arithmetic, editing, and manipulation. A date can be represented as a character string or in an internal format using the Production Reporting date datatype.

The date datatype allows you to store dates in the range of January 1, 4712 b.c. to December 31, 9999 a.d. It also stores the time of day with the precision of a microsecond. The internal date representation keeps the year as a four-digit value. We strongly recommend that you keep dates with four-digit year values (and not truncate to two digits) to avoid date problems at the turn of the century.

Obtaining Date Values

Date values can be obtained in one of five ways:

- By selecting a date column from the database
- By using `INPUT` to get a date from the user
- By referencing or printing the reserved variable *`$current-date`*
- As a result of an Production Reporting date function: *dateadd*, *datediff*, *datenow*, or *strdodate*
- By declaring a date variable using the `DECLARE-VARIABLE` command

For most applications, it is not necessary to declare date variables. See "Declaring Date Variables" on page 160.

Date Arithmetic

Many applications require date calculations. To add or subtract a number of days from a given date, subtract one date from another to find a time difference, or compare dates to find if one date is later, earlier, or the same as another date.

Many databases allow you to perform date calculations in SQL, but that can be awkward if you are trying to write portable code, because the syntax varies between databases. Instead, perform those calculations in Production Reporting—your programs will be portable, because they won't rely on SQL syntax.

The *dateadd* function adds or subtracts a number of specified time units from a given date. The *datediff* function returns the difference between two specified dates in the time units you specify —years, quarters, months, weeks, days, hours, minutes, or seconds. Fractions are allowed—you can add 2.5 days to a given date. Conversion between time units is also allowed—you can add, subtract, or compare dates using days and state the difference using weeks.

The *datenow* function returns the current local date and time. In addition, Production Reporting provides a reserved date variable, *`$current-date`*, which is automatically initialized with the local date and time at the beginning of the program.

You can compare dates by using the usual operators (< , =, or >) in an expression. The *datetostr* function converts a date to a string. The *strtodate* function converts a string to a date.

The following code uses functions to add 30 days to the invoice date and compare it to the current date:

```
begin-select
order_num     (,1)
invoice_date
   if dateadd(&invoice_date,'day',30) < datenow()
      print 'Past Due Order' (,12)
   else
      print 'Current Order'  (,12)
   end-if
   position (+1)
end-select
```

In this example, `dateadd` and `datenow` are used to compare dates. `dateadd` adds 30 days to the invoice date (`&invoice_date`). The resulting date is then compared with the current date, which is returned by `datenow`. If the invoice is older than 30 days, the program prints the string "Past Due Order." If the invoice is 30 days old or less, the program prints the string "Current Order."

To subtract a given number of days from a date, use `dateadd` with a negative argument. This technique is demonstrated in the next example. In this example, the IF condition compares the invoice date with the date of 30 days before today. The condition is equivalent to that of the previous example.

```
   if &invoice_date < dateadd(datenow(),'day',-30)
```

This condition can also be written as follows using `datediff`. Note that the comparison is now a simple numeric comparison, not a date comparison:

```
   if datediff(datenow(),&invoice_date,'day') > 30
```

All three IF statements are equivalent, and they demonstrate the flexibility provided by these functions.

Here is another technique for comparing dates:

```
begin-select
order_date
   if &order_date > strtodate('3/1/2001','dd/mm/yyyy')
     print 'Current Order' ()
   else
     print 'Past Due Order' ()
   end-if
from orders
end-select
```

The `IF` statement has a date column on the left side and `strtodate` on the right side. `strtodate` returns a date type, which is compared with `&order_date`. When the order date is later than January 3, 2001, the condition is satisfied. If the date includes the time of day, the comparison is satisfied for orders of January 3, 2001 with a time of day greater than 00:00.

In the next example, the date is truncated to remove the time-of-day portion of a date:

```
if strtodate(datetostr(&order_date,'dd/mm/yyyy'),'dd/mm/yyyy') >
       strtodate('3/1/2001','dd/mm/yyyy')
```

In this example, `datetostr` converts the order date to a string that only stores the day, month, and year. `strtodate` then converts this value back into a date. With these two conversions, the time-of-day portion of the order date is omitted. Now when it is compared with January 3, 2001, only dates that are of January 4 or later satisfy the condition.

Date Formats

Production Reporting allows you to specify date constants and date values in a special format that is recognized without the use of an edit mask. This is called the literal date format. For example, you can use a value in this format in the *strtodate* function without the use of an edit mask. This format has the advantage of being independent of any database or language preference.

The literal date format is SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]. The first S in this format represents an optional minus sign. If preceded with a minus sign, the string represents a date b.c. The digits that follow represent year, month, day, hours, minutes, seconds, and microseconds. The literal date format assumes a 24-hour clock.

Note:

The literal date format assumes a 24-hour clock.

You can omit one or more time elements from the right part of the format. A default is assumed for the missing elements. Here are some examples:

```
let $a = strtodate('20010409')
let $a = strtodate('20010409152000')
```

The first `LET` statement assigns the date of April 9, 2001 to the variable *$a*. The time portion defaults to 00:00. The second `LET` statement assigns 3:20 in the afternoon of April 9, 2001 to *$a*. The respective outputs (when printed with the edit mask 'DD-MON-YYYY HH:MI AM') are:

```
09-APR-2001 12:00 AM
09-APR-2001 03:20 PM
```

You can also specify a date format with the environment variable *`SQR_DB_DATE_FORMAT`*. You can specify this as an environment variable or in the SQR.INI file. See "SQR.INI" in Volume 2 of the *Production Reporting Developer's Guide*.

String to Date Conversions

If you convert a string variable or constant to a date variable without specifying an edit mask that identifies the format of the string, Production Reporting applies a date format. This implicit conversion occurs with the following commands:

- MOVE
- The *strtodate* function
- The commands `DISPLAY`, `PRINT`, or `SHOW`, when used to output a string variable as a date.

Production Reporting attempts to apply date formats in the following order:

1. The format specified in *`SQR_DB_DATE_FORMAT`*
2. The database-dependent format
3. The literal date format SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]

Date to String Conversions

If you convert a date variable to a string without specifying an edit mask, Production Reporting applies a date format. The conversion occurs with the `datetostr` function and these commands:

- `MOVE`
- `DISPLAY`, `PRINT`, or `SHOW`, when used to output a date variable

Production Reporting attempts to apply date formats in the following order:

1. The format specified in *`SQR_DB_DATE_FORMAT`*
2. The database-dependent format

Database-dependent formats are listed in Table 52, "Default Formats by Database" in Volume 2 of the *Production Reporting Developer's Guide*.

Using Dates with the INPUT Command

The INPUT command also supports dates. A date can be loaded into a date or string variable. For string variables, use the `TYPE=DATE` qualifier. A format for the date should be specified. Here is an example:

```
input $start_date 'Enter starting date'  type=date  format='dd/mm/yyyy'
```

In this example, the user is prompted with *Enter starting date*: (the colon is automatically added). The user then types in the value, which is validated as a date using the “dd/mm/yyyy” format. The value is loaded into the variable *$start_date*.

Date Edit Masks

When you print dates, you can format them with an edit mask. For example:

```
print &order_date () edit 'Month dd, YYYY'
```

This command prints the order date in the specified format. The name of the order date month is printed followed by the day of the month, a comma, and four-digit year.Production Reporting provides a rich set of date edit masks. See “Date Edit Format Code-RR” in Volume 2 of the *Production Reporting Developer's Guide* for a complete listing.

If the value of the date value being edited is March 14, 2001 at 9:35 in the morning, the edit masks produce the following results.

Table 4 Sample Date Edit Masks

Edit Mask	Result	Description
dd/mm/yyyy	14/03/2001	
DD-MON-YYYY	14-MAR-2001	
'Month dd, YYYY'	March 14, 2001	An edit mask containing blank space must be enclosed in single quotes
MONTH-YYYY	MARCH-2001	Name of the month in uppercase followed by four-digit year
HH:MI	09:35	
'HH:MI AM'	09:35 AM	Meridian indicators An edit mask containing blank space must be enclosed in single quotes
YYYYMMDD	20010314	
DDMMYY	140301	
Mon	Mar	Abbreviated name of the month
Day	Thursday	Day of the week
DY	THU	Abbreviated name of day of the week
Q	1	Quarter

Edit Mask	Result	Description
WW	11	Week of the year
W	2	Week of the month
DDD	74	Day of the year
DD	14	Day of the month (1-31)
D	5	Day of the week (Sunday = 1)

If the edit mask contains other text, it is also printed. For example:

```
print &order_date () edit 'As of Month dd, YYYY'
```

This command prints the string “As of March 14, 2001” if the order date is March 14, 2001. Because the words “As of” are not recognized as date mask elements, they are simply printed.

A backslash forces the character that follows into the output. This technique is useful when you want to print text that would otherwise be recognized as a date mask element. For example, a mask of “The \mo\nth is month” results in the output string of “The month is march”. Without the backslashes, the output string would be “The march is march”. The second backslash is needed because “n” is a valid date edit mask element.

In some cases, combining date edit mask elements can result in ambiguity. One example is the mask 'DDDD', which could be interpreted as various combinations of 'DDD' (day of year), 'DD' (day of month), and 'D' (day of week). To resolve such ambiguity, use a vertical bar as a delimiter between format elements. For example, 'DDD' followed by 'D' can be written as 'DDD|D'.

The masks MON, MONTH, DAY, DY, AM, PM, BC, AD, and RM are case-sensitive and follow the case of the mask entered. For example, if the month is January, the mask Mon yields “Jan” and MON yields “JAN”. All other masks are case-insensitive and can be entered in either uppercase or lowercase.

In addition, national language support is provided for the following masks: MON, MONTH, DAY, DY, AM, PM, BC, and AD. See “ALTER-LOCALE” and “SQR.INI” in Volume 2 of the *Production Reporting Developer's Guide* for additional information.

Declaring Date Variables

To hold date values in your program, use date variables. Like string variables, date variables are prefixed with a dollar sign ($). You must explicitly declare date variables using DECLARE-VARIABLE.

Date variables are useful for holding results of date calculations. For example:

```
begin-setup
   declare-variable
      date $c
   end-declare
end-setup
```

```
...
let $c = strtodate('March 1, 2001 12:00','Month dd, yyyy hh:mi')
print $c () edit 'dd/mm/yyyy'
```

In this example, *$c* is declared as a date variable. Later, it is assigned the value of noon on March 1, 2001. The variable *$c* is then printed with the edit mask 'dd/mm/yyyy', which yields 01/03/2001.

Date variables can be initialized with date literals as shown in the following example:

```
begin-setup
   declare-variable
      date $c
   end-declare
end-setup
...
let $c = '20010409152000'
```

The `LET` statement assigns 3:20 in the afternoon of April 9, 2001 to *$c*.

25 National Language Support

In This Chapter

Locales

A *locale* is a set of local preferences for language, currency, and the presentation of dates and numbers. For example, one locale may use English, dollar currency, dates in “dd/mm/yy” format, numbers with commas separating the thousands, and a period for the decimal place.

A locale contains default edit masks for number, money, and date. Use these edit masks to specify the keywords `NUMBER`, `MONEY`, and `DATE`, respectively. You can specify these keywords in the `INPUT`, `MOVE`, `DISPLAY`, `SHOW`, and `PRINT` commands. Their use is discussed and demonstrated in this chapter.

A locale also contains settings for currency symbol, thousands separator, decimal separator, date separator, and time separator. A locale contains settings for N/A, AM, PM, BC, and AD in the language of the locale.

A locale contains a setting for names of the days of the week and names of the months in the language of the locale. It also contains settings for how to handle lower/upper case editing of these names.

These settings are described in detail under “`ALTER-LOCALE`” in Volume 2 of the *Production Reporting Developer's Guide*.

Available Locales

Production Reporting provides predefined locales such as US-English, UK-English, German, French, and Spanish. You can easily define additional locales or modify existing locales by editing the SQR.INI file. For more information about the SQR.INI file, see “SQR.INI” in Volume 2 of the *Production Reporting Developer's Guide*.

With the ALTER-LOCALE command, you can choose a locale—at the beginning of your program or anywhere else. You can even have parts of your program use different locales.

You can select a locale with this command:

```
alter-locale locale = 'German'
```

Default Locale

The SQR.INI file defines a default locale. Most or all of your programs can use the same locale, and specifying the default locale in the SQR.INI file makes it unnecessary to specify the locale in every program.

When you install Production Reporting, the default locale is set to the reserved locale called "System." System is not a real locale. It defines the behavior of older versions of Production Reporting, before national language support was added. The preferences in the system locale are hard-coded in the product and cannot be set or defined in the SQR.INI; however, System settings can be altered at run time using ALTER-LOCALE. The date preferences are dependent on the database you are using. Therefore, date format preferences in the system locale differ for every database you use with Production Reporting.

Sites can have different locales as the default. For example, an office in Paris might use the "French" locale, and an office in London might use the "UK-English" locale. To adapt your program to any location, use the default locale. Your program automatically respects the local preferences, which are specified in the SQR.INI file of the machine on which it is run. For example, you can print the number 5120 using the following command:

```
print #invoice_total () edit '9,999,999.99'
```

The setting of the default locale in the SQR.INI file controls the format. In London, the result might be 5,120.00, and in Paris 5.120,00. The delimiters for thousands and the decimal—the comma and the period—are switched automatically according to the preferences of the locale.

Tip:

Changing the settings of the default locale can change the behavior of existing programs. For example, if you change the default locale to *French*, programs that used to print dates in English may now print them in French. Be sure that you review and test existing programs when making a change to the default locale.

Switching Locales

You can switch from one locale to another any number of times during program execution. This technique is useful for writing reports that use multiple currencies, or reports that have different sections for different locales.

To switch to another locale, use the ALTER-LOCALE command. For example, to switch to the Spanish locale:

```
alter-locale locale = 'Spanish'
```

From this point in the program, the locale is Spanish.

Consider this example:

```
begin-procedure print_data_in_spanish
   ! Save the current locale
   let $old_locale = $sqr-locale
   ! Change the locale to "Spanish"
   alter-locale locale = 'Spanish'
   ! Print the data
   do print_data
   ! restore the locale to the previous setting
   alter-locale locale = $old_locale
end-procedure
```

In this example, the locale is switched to Spanish and later restored to the previous locale before it was switched. To do that, the locale setting before it is changed is read in the reserved variable *`$sqr-locale`* and stored in *$old_locale*. The value of *$old_locale* is then used in the `ALTER-LOCALE` command at the end of the procedure.

Modifying Locale Preferences

With `ALTER-LOCALE`, you can modify any individual preference in a locale. `ALTER-LOCALE` only affects the current program. It does not modify SQR.INI.

Here is an example of how you can modify default preferences in a locale:

```
alter-locale
    date-edit-mask  = 'Mon-DD-YYYY'
    money-edit-mask = '$$,$$$,$$9.99'
```

To restore modified locale preferences to their defaults, you can reselect the modified locale. For example, suppose that the locale was *US-English* and the date and money edit masks were modified using the preceding code. The following code resets the changed date and money edit masks:

```
alter-locale locale = 'US-English'
```

Keywords—NUMBER, MONEY, and DATE

The commands `DISPLAY`, `MOVE`, `PRINT`, and `SHOW` allow you to specify the keywords `NUMBER`, `MONEY`, and `DATE` in place of an explicit number or date edit mask. These keywords can be useful in two cases.

The first case is when you want to write programs that automatically adapt to the default locale. By using the keywords `NUMBER`, `MONEY`, and `DATE`, you tell Production Reporting to take these edit masks from the default locale settings.

The second case is when you want to specify your number, money, and date formats once at the top of your program and use these formats throughout your report. In this case, you define these formats with an `ALTER-LOCALE` command at the top of your program. Then when you use the keywords NUMBER, MONEY, and DATE later in your program, they format number, money, and date outputs with the masks defined in the ALTER-LOCALE command.

Whether you set the locale in the SQR.INI file or in your program, these keywords have the same effect. In the following example, these keywords are used with the PRINT command to produce output for the US-English and French locales:

Sample Program

```
let #num_var = 123456
let #money_var = 123456
let $date_var = strtodate('20010520152000')
! set locale to US-English
alter-locale locale = 'US-English'
print 'US-English locale' (1,1)
print 'With NUMBER keyword ' (+1,1)
print #num_var (,22) NUMBER
print 'With MONEY keyword ' (+1,1)
print #money_var (,22) MONEY
print 'With DATE keyword ' (+1,1)
print $date_var (,22) DATE
! set locale to French
ALTER-LOCALE locale = 'French'
print 'French locale' (+2,1)
print 'With NUMBER keyword ' (+1,1)
print #num_var (,22) NUMBER
print 'With MONEY keyword ' (+1,1)
print #money_var (,22) MONEY
print 'With DATE keyword ' (+1,1)
print $date_var (,22) DATE
```

The output is:

```
US-English locale
With NUMBER keyword                  123,456.00
With MONEY keyword                 $    123,456.00
With DATE keyword                 May 20, 2001
French locale
With NUMBER keyword                  123.456,00
With MONEY keyword                 123.456,00 F
With DATE keyword                 20 mai 2001
```

26 Interoperability

In This Chapter

Interoperability Diagrams

Applications can run Production Reporting programs using the Production Reporting API (application program interface). A Production Reporting program can also call an external application's API.

This interoperability is depicted in the two diagrams shown here.

Figure 7 External Application Invoking an Production Reporting Program Using the Production Reporting API

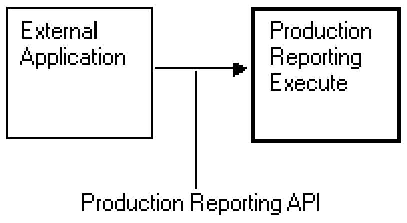
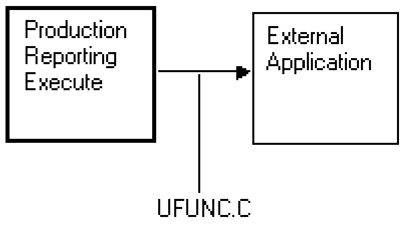


Figure 8 A Production Reporting Program Invoking an External Application using UFUNC.C



This chapter describes how to invoke an Production Reporting program from another application using the Production Reporting API. This API is provided through a DLL on Windows and through an object library on other platforms.

The chapter also explains how to invoke an external application's API by using the UFUNC.C interface.

Calling Production Reporting from Another Application

You can use the following techniques to invoke an Production Reporting program from another application:

- **Using the Production Reporting command line**—The application initiates a process for running Production Reporting. The Production Reporting command includes all necessary parameters. See Chapter 31, "Using the Production Reporting Command Line."
- **Using the Production Reporting API**—The application makes a call to the Production Reporting API. This method is covered in the next section.
- **Using Oracle's Hyperion® SQR® Production Reporting Activator**— Oracle's Hyperion® SQR® Production Reporting Activator runs on the Windows platform and supports application development environments that support ActiveX. For example, Oracle Developer/2000, VisualBasic, PowerBuilder, Delphi, and so on.

Using the Production Reporting API

This section discusses using the Production Reporting API in the following areas:

- Using the Production Reporting API on Windows
- Using the Production Reporting API on Non-windows Platforms
- API Functions for Calling Production Reporting
- Relinking Production Reporting on UNIX Platforms
- Error Values Returned by the Production Reporting API

Using the Production Reporting API on Windows

The Production Reporting API is provided on Windows through a DLL (Dynamic Link Library). You can use the Production Reporting API from any application that is capable of calling DLL functions. For C and C++ applications, a header file, SQRAPI.H, and an import library (SQR.LIB) are provided.

Production Reporting requires the DLLs listed below to run. These DLL files are located in the BINW directory.

bclw32dll	btara320dll	btbat320dll
btcel320dll	btchi320dll	btCroation320dll
btcyrillic320dll	btDevangari320dll	btgre320dll
btguj320dll	btGurmukhi320dll	btheb320dll

btice320dll	btjpn320dll	btkor320dll
btlat320dll	btmal320dll	btnordic320dll
btron320dll	btsla320dll	btsymbol320dll
bttha320dll	bttur320dll	btuc320dll
btukr320dll	btvie320dll	libsti32dll
sqrextdll	sqrdll	stimagesdll

Using the Production Reporting API on Non-windows Platforms

On platforms other than Windows, the Production Reporting API is provided as a static library (sqr.a or sqr.lib). For C and C++ applications, a header file, sqrapi.h, is provided. Be sure to include the Production Reporting API library and your database library when you link your C or C++ application. In addition, the following libraries are required:

- bcl.a
- pdf.a
- libsti.a

API Functions for Calling Production Reporting

The API functions defined for calling Production Reporting are:

Function	Description
int **sqr**(char *)	Runs an Production Reporting program Passes the address of a null terminated string containing an Production Reporting command line, including program name, connectivity information, flags, and arguments This is a synchronous call It returns when the Production Reporting program has completed This function returns zero (0) if it is successful
void **sqrcancel(**void)	Cancels a running Production Reporting program The program may not stop immediately because Production Reporting waits for any currently pending database operations to complete Because the Production Reporting function does not return until the Production Reporting program has completed, *sqrcancel* is called using another thread or some similar asynchronous method
int **sqrend**(void)	Releases memory and closes cursors Cursors can be left open to accelerate repeated execution of the same Production Reporting program Call this function after the last program execution, or optionally between Production Reporting program executions This function returns zero (0)

For the benefit of C/C++ programmers, the APIs are declared in the file SQRAPI.H. Include this header file in your source code:

```
#include 'sqrapi.h'
```

When you call Production Reporting from a program, the most recently run Production Reporting program is saved in memory. If the same Production Reporting program is run again with either the same or different arguments, the program is not scanned again and the SQL statements are not parsed again. This feature provides a significant improvement in processing time.

To force Production Reporting to release its memory and database cursors, call *sqrend*() at any time.

Although memory is automatically released when the program exits, you must call *sqrend()* before the calling program exits to ensure that Production Reporting properly cleans any database resources such as database cursors and temporary stored procedures.

Relinking Production Reporting on UNIX Platforms

To relink Production Reporting on all UNIX platforms, use the *sqrmake* and *makefile* files located in *$SQRDIR/../lib*. After you invoke *sqrmake* and optionally select the database version to link with, the Production Reporting executables are recreated.

Check which 'cc' command line gets created and invoked for Production Reporting, and adapt it to your program. Each UNIX platform and database has its own requirements. Consult your operating system and database product documentation for specific information.

You may see the following output when you relink with Sybase SDK 12.5 under HP/HP-UX 11.00:

```
cc -o {user program} {user objects} {user libraries} \
${SQRDIR}/../lib/nounilib.o ${SQRDIR}/../lib/sqr.a \
${SQRDIR}/../lib/cls.a ${SQRDIR}/../lib/lm_new.o ${SQRDIR}/../lib/liblmgr.a \
${SQRDIR}/../lib/libcrvs.a ${SQRDIR}/../lib/libsb.a ${SQRDIR}/../lib/libsti.a \
${SQRDIR}/../lib/bcl.a ${SQRDIR}/../lib/pdf.a -lcl -lrt -lpthread -ldld \
-lcres -L${SYBASE}/${SYBASE_OCS}/lib -lct -lcs -ltcl -lcomn -lintl -lcl -lm \
-lBSD -ldld -L${HYPERION}/common/JRE/HP/1.4.2/lib/PA_RISC/server \
-ljvm -Wl,-s,+s -z -Wl,-O
```

Check the make files or link scripts that are supplied with Production Reporting for details. You may want to copy and modify those to link in your program.

To call Production Reporting, call *sqr*() and pass a command line. For example, in C:

```
status = sqr("myprog sammy/baker arg1 arg2 arg3");
if (status != 0)
        ...error occurred...
```

Error Values Returned by the Production Reporting API

Table 5 Standalone and Callable Error Values Returned by the Production Reporting API

Error Code	Reason
0	Normal exit
1	Error exit
2	Cannot process SQRERRDAT
3	Command-line flag in error
4	Problem creating SQT file
5	Program did not compile
6	Problem with SQR/SQT file (open/read)
7	Problem with LIS file (create/write)
8	Problem with ERR file (create/write)
9	Problem with LOG file (create/write)
10	Problem with POSTSCRISTR file (open/read)
11	Cannot call Production Reporting recursively
12	Problem with Windows
13	Internal error occurred
14	Problem with SQRWINDLL
15	Problem with ZCF file

Note:

Error code 12 only applies to Windows.

Extending Production Reporting—UFUNC.C

The Production Reporting language can be extended by adding user functions written in standard languages such as C. This feature allows you to integrate your own code and third-party libraries into Production Reporting. For example, assume you had a library for communication over a serial line, with functions for initiating the connection and sending and receiving data. Production Reporting would allow you to call these functions from Production Reporting programs.

To extend Production Reporting in this way, you must prepare the functions, "tell" Production Reporting about them, and then link the objects (and libraries) with the Production Reporting objects and libraries to form a new Production Reporting executable. The new Production

Reporting executable recognizes the new functions as if they were standard Production Reporting functions.

For detailed information on writing custom functions using UFUNC.C, see “Writing Custom Functions” in Volume 2 of the *Production Reporting Developer's Guide*.

ufunc on the Windows Platform

On the Windows platform, *ufunc* resides in SQREXT.DLL. You can rebuild SQREXT.DLL using any language or tool, as long as the appropriate calling protocol is maintained. The source code for SQREXT.DLL is included in the shipped package (EXTUFUNC.C).

When SQR.DLL and SQRT.DLL are loaded, they look for SQREXT.DLL in the same directory and for any `DLLs` specified in the [SQR Extension] section in SQR.INI. If SQR.DLL and SQRT.DLL find SQREXT.DLL and the DLLs specified in the SQR.INI file, they make the following calls in the DLLs, passing the instance handle (of the calling module) and three function pointers:

```
void InitSQRExtension (
    HINSTANCE hInstance,
    FARPROC lpfnUFuncRegister,
    FARPROC lpfnConsole,
    FARPROC lpfnError
    );
```

Implementing New User Functions on the Windows Platform

You can implement new user functions in SQREXT.DLL or any other extension DLL. The extension DLLs must have the *InitSQRExtension()* function exported. If you choose to implement user functions in SQREXT.DLL, you should rebuild the DLL using the supplied make file, SQREXT.MAK. If new extension DLLs containing new user functions are to be used, they must be listed in the [SQR Extension] section in SQR.INI in the SYSTEM directory.

For any *ufunc*, you must register it by making the following call in *InitSQRExtension()*.

```
lpfnUFuncRegister(struct ufnns* ufunc);
```

The function pointer *lpfnUFuncRegister* is passed in from the calling module. See EXTUFUNC.C for the definition of *struct ufnns* and the sample user functions.

XML Support in Production Reporting

XML support in Production Reporting is provided through the DataDirect Connect for ODBC XML driver. This driver supports tabular and hierarchical-formatted XML documents accessed from local file systems, web servers, and web services.

The XML driver supports three main types of tabular-formatted files; namely, Microsoft Data Islands, ADO 2.5 persisted files, and DataDirect Formats. The XML driver runs in Windows environments, and includes an SQL Engine that provides ANSI SQL-92 support.

For detailed information on the ODBC XML Driver provided by DataDirect, refer to the DataDirect documentation installed with the ODBC driver. To access this documentation, open **books.pdf** found in:

```
<hyperion_home>\common\ODBC\Merant\5.2\books\odbc
```

After you open books.pdf, you can access any of the following DataDirect guides:

- Installation Guide
- User's Guide
- Reference
- Troubleshooting Guide

27 Testing and Debugging

In This Chapter

Using the Test Feature

During the development of an Production Reporting program, you frequently test it by running it and examining its output. In many cases, you are only interested in the first few pages of the report.

To speed the cycle of running and viewing a few pages, use the `-T` command-line flag. The `-T` flag lets reports finish more quickly because all `BEGIN-SELECT` and `ORDER BY` clauses are ignored. The database does not sort the data and the first set of records are selected sooner. Enter the desired number of test pages after the `-T` flag. For example, `-T6` causes the program to stop after creating six pages of output.

Note:

If your program contains break logic, the breaks can occur in unexpected locations because the ORDER BY clause is ignored.

To test a report file called *customer.sqr*, enter the following command:

```
sqr customer username/password -T3
```

The `-T3` flag specifies that the program stops running after 3 pages are produced.

When the test completes successfully, check it by displaying the output file on your screen or printing it. The default name of the output file is the same as the program file with the extension LIS. For example, if your report is named *customer.sqr*, the output file is named *customer.lis*.

If you are using SQR Production Reporting Studio, select **Limit to nn pages** in the Run dialog box.

When the development of your program is complete, run it without the `-T` flag. Your program processes all `ORDER BY` clauses and run to completion. If the program creates multiple reports, the `-T` flag restriction applies only to the first report.

Using the #DEBUG Command

When debugging a program it is useful to:

- Display data or show when a procedure or query executes by using temporary `SHOW` or `DISPLAY` commands in key places in the program.
- Isolate problem areas by temporarily skipping the parts of the program that work correctly.
- Temporarily cause additional behavior in questionable areas of the program. For example, display or modify variables that you suspect are causing a problem.

Production Reporting provides the `#DEBUG` command to help you make temporary changes to your code. You can use the `#DEBUG` command to conditionally process portions of your program.

Precede the command with `#DEBUG`, as shown here:

```
#debug display $s
```

When the `#DEBUG` precedes a command, that command is processed only if the `-DEBUG` flag is specified on the Production Reporting command line. In this example, the value of *$s* is displayed only when you run the program with `-DEBUG`.

You can achieve debug multiple commands by using up to 36 letters or digits to differentiate between them. Indicate which command is to be debugged on the `-DEBUG` flag, as shown here:

```
sqr myreport username/password -DEBUGabc
```

In this example, commands preceded by `#DEBUG`, `#DEBUGa`, `#DEBUGb`, or `#DEBUGc` are compiled when the program is executed. Commands preceded with `#DEBUGd` are not compiled because d was not specified in the `-DEBUG` command-line flag.

Using Compiler Directives for Debugging

You can conditionally compile entire sections of your program using the five compiler directives:

- #IF
- #ELSE
- #END-IF or #ENDIF
- #IFDEF
- #IFNDEF

You can use the value of a substitution variable, declared by a `#DEFINE` command, to turn on or off a set of statements, as shown here:

```
#define DEBUG_SESSION Y
#if DEBUG_SESSION = 'Y'
begin-procedure dump_array
   let #i = 0
   while #i < #counter
      ! Get data from the array
      get $state $city $name $phone from customer_array(#i)
      print $state (,1)
      print $city  (,7)
      print $name  (,24)
      print $phone (,55)
      position (+1)
      add 1 to #i
   end-while
end-procedure ! dump_array
#end-if
```

The *dump_array* procedure is only used for debugging. By defining `DEBUG_SESSION` as `Y`, the *dump_array* procedure is included in the program. Later, you can change `DEBUG_SESSION` to `N` and exclude the *dump_array* procedure from the program. The `#IF` command in this example is case-insensitive.

Common Programming Errors

The most common programming error using Production Reporting is mistyping variable names. Because Production Reporting does not require variables to be declared, it does not issue an error message when variables names are mistyped. Instead, Production Reporting considers the mistyped variable as if it is another variable.

For example:

```
let #customer_access_code = 55
print #customer_acess_code ()
```

This example will not print 55 because we mistyped the variable name. Can you see the typo? One *c* in *acess* on the `PRINT` command is missing.

Another problem relates to global versus local variables. If you refer to a global variable in a local procedure without preceding it with an underscore, Production Reporting does not issue an error message. Instead, it is a new local variable name. For example:

```
begin-procedure main
   let $area = 'North'
   do proc
end-procedure ! main
begin-procedure proc local
   print $area ()   ! Should be $_area
end-procedure
```

In this example, the local procedure *proc* prints the value of the local variable *$area* and not the global variable *$area*. It prints nothing because the local *$area* variable did not receive a value. To refer to the global variable, use *$_area*.

Such small errors are hard to detect because Production Reporting considers *#customer_acess_code* as simply another variable with a value of zero.

28 Performance and Tuning

In This Chapter

About Performance and Tuning

Performance considerations are an important aspect of application development. This chapter examines some of the issues that affect the performance of Production Reporting programs. This chapter also describes certain Production Reporting capabilities that can help you write high-performance programs

Whenever your program contains a `BEGIN-SELECT`, `BEGIN-SQL`, or `EXECUTE` command, it performs an SQL statement. Processing SQL statements typically consumes significant computing resources. Tuning SQL statements typically yields higher performance gains than tuning any other part of your program.

General tuning of SQL is outside the scope of this book. Tuning SQL is often specific to the type of database that you are using—tuning SQL statements for an ORACLE database may differ from tuning SQL statements for DB2. This chapter focuses on Production Reporting tools for simplifying SQL statements and reducing the number of SQL executions.

Simplifying a Complex SELECT

With relational database design, information is often "normalized" by storing data entities in separate tables. To display the normalized information, you must write a `SELECT` statement that joins these tables together. With many database systems, performance suffers when you join more than three or four tables in one SELECT.

With Production Reporting, you can perform multiple SELECT statements and nest them as we saw in Chapter 8, "Master/Detail Reports." In this way, you can break a large join into several simpler SELECTS. For example, a SELECT statement that joins *orders* and *products* tables can be broken into two SELECTS. The first SELECT retrieves the orders in which we are interested. For each order retrieved, a second SELECT retrieves the products that were ordered. The second SELECT is correlated to the first SELECT by having a condition such as:

```
where order_num = &order_num
```

This condition specifies that the second SELECT only retrieves products for the current order.

Similarly, if your report is based on products ordered, you can make the first SELECT retrieve the products, and make the second SELECT retrieve the orders for each product.

This method improves performance in many cases, but not all. To achieve the best performance, experiment with the alternatives.

Using LOAD-LOOKUP to Simplify Joins

Database tables often contain key columns such as a product code or customer number. To retrieve a certain piece of information, you join two or more tables that contain the same column. For example, to obtain a product description, you can join the *orders* table with the *products* table, using the *product_code* column as the key.

With LOAD-LOOKUP, you can reduce the number of tables that are joined in one SELECT. Use this command in conjunction with one or more LOOKUP commands.

The LOAD-LOOKUP command defines an array containing a set of keys and values and loads it into memory. The LOOKUP command looks up a key in the array and returns the associated value. In some programs, this technique performs better than a conventional table join.

LOAD-LOOKUP can be used in the SETUP section or in a procedure. If used in the SETUP section, it is processed only once. If used in a procedure, it is processed each time it is encountered.

LOAD-LOOKUP retrieves two fields from the database, the KEY field and the RETURN_VALUE field. Rows are ordered by KEY and stored in an array. The KEY field must be unique and contain no NULL values.

When the LOOKUP command is used, the array is searched (using a "binary" search) to find the RETURN_VALUE field corresponding to the KEY referenced in the lookup.

The following sample code illustrates LOAD-LOOKUP and LOOKUP:

```
begin-setup
  load-lookup
     name=prods
        table=products
        key=product_code
        return_value=description
end-setup
...
```

```
begin-select
order_num (+1,1)
product_code
  lookup prods &product_code $desc
  print $desc (,15)
from orderlines
end-select
```

In this example, LOAD-LOOKUP loads an array with the *product_code* and *description* columns from the *products* table. The lookup array is named *prods*. The *product_code* column is the key and the *description* column is the return value. In the SELECT paragraph, a LOOKUP on the *prods* array retrieves the *description* for each *product_code*. This technique eliminates joining the *products* table in the SELECT.

If the *ordlines* and *products* tables were simply joined in the SELECT (without LOAD-LOOKUP), the code would look like this:

```
begin-select
order_num (+1,1)
ordlines.product_code
description (,15)
from ordlines, products
where ordlines.product_code = products.product_code
end-select
```

Which is faster, a database join or LOAD-LOOKUP? It depends on your program. LOAD-LOOKUP improves performance in the following situations:

- When it is used with multiple SELECTS
- When it keeps the number of tables being joined from exceeding three or four
- When the number of entries in the LOAD-LOOKUP table is small compared to the number of rows in the SELECT, and they are used often
- When most entries in the LOAD-LOOKUP table are used

Tip:

You can concatenate columns if you want RETURN_VALUE to return multiple columns. The concatenation symbol is database-specific.

Improving SQL Performance with Dynamic SQL

Chapter 17, "Dynamic SQL and Error Checking," explained how to use dynamic SQL variables. Dynamic SQL can also be used in some situations to simplify a SQL statement and gain performance.

```
begin-select
order_num
from orders, customers
where order.customer_num = customers.customer_num
and ($state = 'CA' and order_date > $start_date
     or $state != 'CA' and ship_date > $start_date)
```

```
end-select
```

In this example, a given value of *$state*, *order_date* or *ship_date* is compared to *$start_date*. The OR operator in the condition makes such multiple comparisons possible. With most databases, an OR operator slows processing. It can cause the database to perform more work than necessary.

However, the work can be done with a simpler `SELECT`. For example, if *$state* is 'CA,' the following `SELECT` would work:

```
begin-select
order_num
from orders, customers
where order.customer_num = customers.customer_num
and order_date > $start_date
end-select
```

Dynamic SQL allows you to check the value of *$state* and create the simpler condition:

```
if $state = 'CA'
   let $datecol = 'order_date'
else
   let $datecol = 'ship_date'
end-if
begin-select
order_num
from orders, customers
where order.customer_num = customers.customer_num
and [$datecol] > $start_date
end-select
```

The substitution variable [*$datecol*] substitutes the name of the column to be compared with *$state_date*. The `SELECT` is simpler and no longer uses an `OR`. In most cases, this use of dynamic SQL improves performance.

Examining SQL Cursor Status

Because Production Reporting programs select and manipulate data from a SQL database, it is helpful to understand how Production Reporting handles SQL statements and queries.

Production Reporting programs can perform multiple SQL statements. Moreover, the same SQL statement can be executed many times.

When your program executes, a pool of SQL statement handles—called cursors—is maintained. A cursor is a storage location for one SQL statement, for example, `SELECT`, `INSERT`, or `UPDATE`. Every SQL statement uses a cursor for processing. A cursor holds the context for the execution of a SQL statement.

The cursor pool consists of 30 cursors, and its size cannot be changed. When a SQL statement is re-executed, its cursor can be immediately reused if it is still in the cursor pool. When your Production Reporting program executes more than 30 SQL statements, cursors in the pool are reassigned.

To examine how cursors are managed, use the -S command-line flag. This flag causes cursor status information to be displayed at the end of a run.

The following information is displayed for each cursor:

```
Cursor #nn:
SQL = <SQL statement>
Compiles = nn
Executes = nn
Rows = nn
```

The listing also includes the number of compiles, which vary according to the database and the complexity of the query. With Oracle, for example, a simple query is compiled only once. With Sybase, a SQL statement is compiled before it is first executed and recompiled for the purpose of validation during the Production Reporting compile phase. Therefore, you may see two compiles for a SQL statement. Later when the SQL is re-executed, if its cursor is found in the cursor pool, it can proceed without recompiling.

Avoiding Temporary Database Tables

Programs often use temporary database tables to hold intermediate results. Creating, updating, and deleting database temporary tables is a very resource-consuming task, however, and can hurt your program's performance. Production Reporting provides two alternatives to using temporary database tables.

The first alternative is to store intermediate results in an Production Reporting array. The second is to store intermediate results in a local flat file. Both techniques can bring about a significant performance gain. You can use the Production Reporting language to manipulate data stored in an array or a flat file.

These two methods are explained and demonstrated in the following sections. Methods for sorting data in Production Reporting arrays or flat files are also explained.

Using and Sorting Arrays

Chapter 9, "Cross-Tabular Reports," introduced the array as a means of holding data records during program execution.

An Production Reporting array can hold as many records as can fit in memory. During the first pass, when records are retrieved from the database, you can store them in the array. Subsequent passes on the data can be made without additional database access.

The following code retrieves records, prints them, and saves them into an array named *customer_array*:

```
create-array name=customer_array size=1000
   field=state:char   field=city:char
   field=name:char    field=phone:char
let #counter = 0
begin-select
state (,1)
city  (,7)
name  (,24)
phone (,55)
```

```
   position (+1)
   put &state &city &name &phone into customer_array(#counter)
   add 1 to #counter
from customers
end-select
```

This example creates an array named *customer_array*. The array has four fields that correspond to the four columns selected from the *customers* table, and it can hold up to 1,000 rows. If you anticipate that the *customers* table has more than 1,000 rows, use the EXTENT argument in the CREATE-ARRAY command to allow the ray to grow. (See "CLEAR-ARRAY" in Volume 2 of the *Production Reporting Developer's Guide*.)

The SELECT prints the data. The PUT command then stores the data in the array. Chapter 9, "Cross-Tabular Reports," showed how to use the LET command to assign values to array fields. The PUT command performs the same work, but with fewer lines of code. With PUT, you can assign all four fields in one command.

The *#counter* variable serves as the array subscript. It starts with zero and maintains the subscript of the next available entry. At the end of the SELECT, the value of *#counter* is the number of records in the array.

The next piece of code retrieves the data from *customer_array* and prints it:

```
let #i = 0
while #i < #counter
   get $state $city $name $phone from customer_array(#i)
   print $state (,1)
   print $city  (,7)
   print $name  (,24)
   print $phone (,55)
   position (+1)
   add 1 to #i
end-while
```

In this piece of code, *#i* goes from 0 to *#counter*-1. The fields from each record are moved into the corresponding variables *$name*, *$city*, *$state*, and *$phone*. These values are then printed.

Sorting

In many cases, intermediate results must be sorted by another field. *Program ex24a.sqr* shows how to sort *customer_array* by *name*. The program uses a well-known sorting algorithm called *QuickSort*. You can copy this code into your program, make appropriate changes, and use it to sort your array. For further information on *QuickSort*, see the book *Fundamentals of Data Structures* by Horowitz and Sahni, 1983.

Program ex24a.sqr

```
#define MAX_ROWS 1000
begin-setup
create-array name=customer_array size={MAX_ROWS}
   field=state:char     field=city:char
   field=name:char      field=phone:char
!
! Create a helper array that is used in the sort
```

```
!
create-array name=QSort size={MAX_ROWS}
      field=n:number    field=j:number
end-setup
begin-program
  do main
end-program
begin-procedure main
let #counter = 0
!
! Print customers sorted by state
!
begin-select
state (,1)
city  (,7)
name  (,24)
phone (,55)
   position (+1)
   ! Put data in the array
   put &state &city &name &phone into customer_array(#counter)
   add 1 to #counter
from customers
order by state
end-select
position (+2)
!
! Sort customer_array by name
!
let #last_row = #counter - 1
do QuickSort(0, 0, #last_row)
!
! Print customers (which are now sorted by name)
!
let #i = 0
while #i < #counter
   ! Get data from the array
   get $state $city $name $phone from customer_array(#i)
   print $state (,1)
   print $city  (,7)
   print $name  (,24)
   print $phone (,55)
   position (+1)
   add 1 to #i
end-while
end-procedure ! main
!
! QuickSort
!
! Purpose: Sort customer_array by name.
! This is a recursive function. Since Production Reporting does not
allocate
! local variables on a stack (they are all static), this
! procedure uses a helper array.
!
! #level - Recursion level (used as a subscript to the helper
! array)
! #m     - The "m" argument of the classical QuickSort
! #n     - The "n" argument of the classical QuickSort
!
begin-procedure QuickSort(#level, #m, #n)
   if #m < #n
      let #i = #m
      let #j = #n + 1
      ! Sort key is "name"
      let $key = customer_array.name(#m)
      while 1
         add 1 to #i
         while #i <= #j and customer_array.name(#i) < $key
             add 1 to #i
         end-while
         subtract 1 from #j
         while #j >= 0 and customer_array.name(#j) > $key
            subtract 1 from #j
         end-while
         if #i < #j
            do QSortSwap(#i, #j)
         else
            break
         end-if
      end-while
      do QSortSwap(#m, #j)
      add 1 to #level
      ! Save #j and #n
      let QSort.j(#level - 1) = #j
      let QSort.n(#level - 1) = #n
      subtract 1 from #j
      do QuickSort(#level, #m, #j)
      ! restore #j and #n
      let #j = QSort.j(#level - 1)
      let #n = QSort.n(#level - 1)
      add 1 to #j
      do QuickSort(#level, #j, #n)
      subtract 1 from #level
   end-if
end-procedure ! QuickSort
!
!
! QSortSwap
!
! Purpose: Swaps records #i and #j of customer_array
!
! #i     - Array subscript
! #j     - Array subscript
!
begin-procedure QSortSwap(#i, #j)
   get $state $city $name $phone from customer_array(#i)
   let customer_array.state(#i) = customer_array.state(#j)
   let customer_array.city(#i)  = customer_array.city(#j)
   let customer_array.name(#i)  = customer_array.name(#j)
   let customer_array.phone(#i) = customer_array.phone(#j)
   put $state $city $name $phone into customer_array(#j)
end-procedure ! QSortSwap
```

The QuickSort algorithm uses a recursive procedure, thus it calls itself. Production Reporting maintains only one copy of the procedure's local variables. In *QuickSort* the variables *#j* and *#n* are overwritten when *QuickSort* calls itself.

For the algorithm to work properly, the program must save the values of these two variables before making the recursive call, then restore those values when the call completes. *QuickSort* can call itself recursively many times, so the program may save many copies of *#j* and *#n*. To do this, add a *#level* variable that maintains the depth of recursion. In this example, a helper array, *Qsort*, is used to hold multiple values of *#j* and *#n*.

The *QuickSort* procedure takes three arguments. The first is the recursion level (or depth), which is *#level*, as previously described. The second and third arguments are the beginning and end of the range of rows to be sorted. Each time *QuickSort* calls itself, the range gets smaller. The main procedure starts *QuickSort* by calling it with the full range of rows.

The *QSortSwap* procedure swaps two rows in *customer_array*. Typically, rows with a lower key value are moved up.

The procedures *QuickSort* and *QSortSwap* in *ex24a.sqr* refer to *customer_array* and its fields. If you plan to use these procedures to sort an array in your applications, change these references to the applicable array and fields. The *QuickSort* procedure sorts in ascending order.

QuickSort and National Language

The *QuickSort* procedure does not support National Language Sensitive character string sort. The comparisons

```
while #i <= #j and customer_array.name(#i) < $key
and
while #j >= 0 and customer_array.name(#j) > $key
```

are simple string comparisons. They work well for US ASCII English, but they may not sort correctly with other languages. For such languages, write a National Language Sensitive character string comparison and add that to Production Reporting. Chapter 26, "Interoperability,"explains how to add functions to Production Reporting. You can modify the *QuickSort* procedure as follows.

```
while #i <= #j and NLS_STRING_COMPARE(customer_array.name(#i),$key)
while #j >= 0 and NLS_STRING_COMPARE($key,customer_array.name(#j))
```

Using and Sorting Flat Files

An alternative to an array is a flat file. You can use a flat file when the required array size exceeds available memory. As is the case with an array, you may need a sorting utility that supports NLS.

The sample code in the previous section can be rewritten to use a file instead of an array. The advantage of using a file is that the program is not constrained by the amount of memory that is available. The disadvantage of using a file is that the program will perform more I/O. However, it may still be faster than performing another SQL statement to retrieve the same data.

This program uses the UNIX sort utility to sort the file by *name*. This example can be extended to include other operating systems.

The following code is rewritten to use the file *cust.dat* instead of the array.

Program ex 24b.sqr

```
begin-program
  do main
end-program
begin-procedure main
!
! Open cust.dat
!
open 'cust.dat' as 1 for-writing record=80:vary
begin-select
state (,1)
city  (,7)
name  (,24)
phone (,55)
   position (+1)
   ! Put data in the file
   write 1 from &name:30 &state:2 &city:16 &phone:10
from customers
order by state
end-select
position (+2)
!
! Close cust.dat
close 1
! Sort cust.dat by name
!
call system using 'sort cust.dat > cust2.dat' #status
if #status <> 0
    display 'Error in sort'
    stop
end-if
!
! Print customers (which are now sorted by name)
!
open 'cust2.dat' as 1 for-reading record=80:vary
while 1  ! loop until break
   ! Get data from the file
   read 1 into $name:30 $state:2 $city:16 $phone:10
   if #end-file
      break   ! End of file reached
   end-if
   print $state (,1)
   print $city  (,7)
   print $name  (,24)
   print $phone (,55)
   position (+1)
end-while
!
! close cust2.dat
close 1
end-procedure ! main
```

The program starts by opening a file *cust*.dat.

```
open 'cust.dat' as 1 for-writing record=80:vary
```

The `OPEN` command opens the file for writing and assigns it file number 1. You can open as many as 12 files in one Production Reporting program. The file is set to support records of varying length with a maximum of 80 bytes (characters). For this example, you could also use fixed-length records.

As the program selects records from the database and prints them, it writes them to *cust*.dat.

```
write 1 from &name:30 &state:2 &city:16 &phone:10
```

The `WRITE` command writes the four columns into file number 1—the currently open *cust*.dat. It writes the name first, which makes it easier to sort the file by name. The program writes fixed-length fields—for example, `&name:30` specifies that the name column uses 30 characters. If the name is shorter, it is padded with blanks. When the program has finished writing data to the file, it closes the file using the `CLOSE` command.

The file is sorted with the UNIX sort utility.

```
call system using 'sort cust.dat > cust2.dat' #status
```

The command `sort cust.dat > cust2.dat` is sent to the UNIX system. It invokes the UNIX sort command to sort *cust.dat* and direct the output to *cust2.dat*. The completion status is saved in *#status*; a status of 0 indicates success. Because *name* is at the beginning of each record, the file is sorted by *name*.

Next, we open *cust2.dat* for reading. The command

```
read 1 into $name:30 $state:2 $city:16 $phone:10
```

reads one record from the file and places the first 30 characters in *$name*. The next two characters are placed in *$state* and so on. When the end of the file is encountered, the reserved variable *#end-file* is automatically set to 1 (true). The program checks for *#end-file* and breaks out of the loop when the end of the file is reached. Finally, the program closes the file using the `CLOSE` command.

Creating Multiple Reports in One Pass

Sometimes you must create multiple reports that are based on the same data. In many cases, these reports are similar, with only a difference in layout or summary. Typically, you can create multiple programs and even reuse code. However, if each program is executed separately, the database has to repeat the query. Such repeated processing is often unnecessary.

With Production Reporting, one program can create multiple reports simultaneously. In this method, one program creates multiple reports, making just one pass on the data, greatly reducing the amount of database processing. The multiple report feature of Production Reporting is described in Chapter 19, “Multiple Reports.”

Tuning Production Reporting Numerics

Production Reporting provides three types of numeric values:

- Machine floating point numbers

- Decimal numbers
- Integers

Machine floating point numbers are the default. They use the floating point arithmetic provided by the hardware. This method is very fast. It uses binary floating point and normally holds up to 15 digits of precision.

Some accuracy can be lost when converting decimal fractions to binary floating point numbers. To overcome this loss of accuracy, you can sometimes use the `ROUND` option of commands such as `ADD`, `SUBTRACT`, `MULTIPLY`, and `DIVIDE`. You can also use the round function of `LET` or numeric edit masks that round the results to the desired precision.

Decimal numbers provide exact math and precision of up to 38 digits. Math is performed in software. This is the most accurate method, but also the slowest.

Integers can be used for numbers that are known to be integers. Several benefits exist for using integers: They enforce the integer type by not allowing fractions, and they adhere to integer rules when dividing numbers. Integer math is also the fastest, typically faster than floating point numbers.

If you use the `DECLARE-VARIABLE` command, the -DNT command-line flag, or the `DEFAULT-NUMERIC` entry in the [Default-Settings] section of the SQR.INI file, you can choose the type of numbers that Production Reporting uses. Moreover, you can select the type for individual variables in the program with the `DECLARE-VARIABLE` command. When you choose decimal numbers, you can also specify the desired precision.

Selecting the numeric type for variables allows you to fine-tune the precision of numbers in your program. For most applications, however, this type of tuning does not yield a significant performance improvements and we recommend selecting decimal. The default is machine floating point to provide compatibility with older releases of the product.

Compiling Production Reporting Programs and Using Production Reporting Execute

Compiling your Production Reporting program can improve its performance. The compiled program is stored in a run-time (.SQT) file. You can then run it with Production Reporting Execute. Your program runs faster because it skips the compile phase. This method is explained in Chapter 29, “Compiling Programs and Using Production Reporting Execute .”

Buffering Fetched Rows

When a `BEGIN-SELECT` command is executed, records are fetched from the database server. To improve performance, they are fetched in groups rather than one at a time. The default is groups of 10 records. The records are buffered, and your program processes these records one at a time. A database fetch operation is therefore performed after every 10 records, instead of after every single record. This is a substantial performance gain. If the database server is on another computer, then network traffic is also significantly reduced.

The number of records to fetch together can be modified using the -B command-line flag or for an individual `BEGIN-SELECT` command using its -B option. In both cases, you specify the number of records to be fetched together. For example -B100 specifies that records be fetched in groups of 100. Thus, the number of database fetch operations is further reduced.

This feature is currently available with Production Reporting for ODBC and Production Reporting for the Oracle or Sybase databases.

Executing Programs on the Database Server

You can reduce network traffic and greatly improve performance by running Production Reporting programs directly on the database server machine. The Production Reporting product is available on many server platforms including Windows and UNIX.

P a r t   V

Running and Printing

In Running and Printing:

29 Compiling Programs and Using Production Reporting Execute

This chapter explains how to save and run compiled versions of Production Reporting programs.

For the user, running a Production Reporting program is a one-step process. For Production Reporting, however, there are two steps—compiling the program and executing it. When compiling a program, Production Reporting:

- Reads, interprets, and validates the program
- "preprocesses" substitution variables and certain commands—`ASK`, `#DEFINE`, `#INCLUDE`, `#IF`, and `#IFDEF`
- Validates SQL statements
- Performs the `SETUP` section

Production Reporting allows you to save the compiled version of a program and use it when you rerun a report. That way, you perform the compile step only once and skip it in subsequent runs. Note that Production Reporting does not compile the program into machine language. Production Reporting creates a ready-to-execute version of your program that is compiled and validated. This file is portable between hardware platforms.

The steps are simple. Run the Production Reporting executable (sqr) against your Production Reporting program file and include the -RS command-line flag to save the run-time file. Production Reporting creates a file with a file name extension of .sqt. You should enter something similar to:

```
sqr ex1a.sqr sammy/baker@rome -RS
```

Run the Production Reporting executable (sqr) with the -RT command-line flag to execute the. SQT file. Execution is faster because the program is compiled. An example of this is as follows:

```
sqr ex1a.sqt sammy/baker@rome -RT
```

The Production Reporting product distribution includes Production Reporting Execute (the SQRT program). Production Reporting Execute is capable of running .sqt files, but does not include the code that compiles an Production Reporting program. (This program is equivalent to running Production Reporting with -RT.) You can run the .sqt file by invoking Production Reporting Execute from the command line with sqrt. An example of running Production Reporting Execute from the command line is as follows:

```
sqrt ex1a.sqt sammy/baker@rome
```

It is important to realize that after you save the run-time (.sqt) file, Production Reporting no longer performs any compile-time steps such as executing `#IF`, `#INCLUDE`, or `ASK` commands or performing the `SETUP` section. These were performed at the time that the program was compiled and the run-time file was saved.

You must make a clear distinction between what is performed at compile time and what is performed at run time. Think of compile-time steps as defining what the report is. Commands such as `#IF` or `ASK` allow you to customize your report at compile time. For run-time customization, you should use commands such as `IF` and `INPUT`.

A list of Production Reporting features that apply at compile time and their possible run-time equivalents follows. In some cases, no equivalent exists and you have to work your way around the limitation. For example, you may have to use substitution variables with commands that require a constant and do not allow a variable. We demonstrated this solution in Chapter 16, "Writing Printer-Independent Reports," where we worked around the limitation of the `USE-PRINTER-TYPE` command, which does not accept a variable as an argument.

Table 6 Compile-time Commands and Run-time Equivalents

Compile Time	Run Time
Substitution variables	Use regular Production Reporting variables If you are substituting parts of a SQL statement, use dynamic SQL instead See Chapter 17, "Dynamic SQL and Error Checking."
ASK	**INPUT**
#DEFINE	**LET**
#IF	**IF**
INCLUDE	No equivalent
DECLARE-LAYOUT, margins	No equivalent
Number of heading or footing lines	**ALTER-REPORT**
DECLARE-CHART	**PRINT-CHART**
DECLARE-IMAGE	**PRINT-IMAGE**
DECLARE-PROCEDURE	**USE-PROCEDURE**
DECLARE-PRINTER	**ALTER-PRINTER (where possible)**
USE (Sybase only)	-DB command-line flag

Tip:

See "Production Reporting Command-line Arguments" in Volume 2 of the *Production Reporting Developer's Guide* for a list of the flags that you can use with Production Reporting Execute.

30 Printing Issues

In This Chapter

Printing in Production Reporting

This chapter discusses technical issues relevant to printing. Except on the Microsoft Windows platform, Production Reporting does not actually print the report. Production Reporting creates an output file that contains the report, but it does not print it directly. The output file can be a printer-specific file or an SQR Portable File (SPF). SQR Portable Files have a default extension of SPF or SNN (for multiple reports).

Command-Line Flags and Output Types

Table 7 summarizes Production Reporting command-line flags that produce a certain type of output and the types of output they produce.

Table 7 Command Line Flags and Output Types

Command Line Flag	Output File Extension	File Format	Suitable for
-PRINTER:EH	htm	Enhanced HTML	Intranet/Internet
-PRINTER:HP	lis	PCL	HP LaserJet printer
-PRINTER:HT	htm	HTML	Intranet/Internet
-PRINTER:LP	lis	ASCII	Line printer
-PRINTER:PD	pdf	PDF	Acrobat Reader
-PRINTER:PS	lis	PostScript	PostScript printer
-PRINTER:WP	Output goes directly to the default printer without being		Windows

Command Line Flag	Output File Extension	File Format	Suitable for
	saved to a file You can set your default printer using the Windows Control Panel		
-NOLIS	spf or snn	SQR Portable Format	Production Reporting Print and Production Reporting Viewer can print this file to different printers
-KEEP	spf or snn (in addition to the lis file that is normally created)	SQR Portable Format and the format of the lis file	Production Reporting Print and Production Reporting Viewer can print this spf file to different printers
No flag	lis	ASCII, PCL, or PostScript	Line printer, HP LaserJet, or PostScript, respectively

Note:

When no flags are specified, Production Reporting produces a line printer output unless otherwise set in the Production Reporting program with DECLARE-PRINTER, USE-PRINTER-TYPE, or the PRINTER-TYPE option of DECLARE-REPORT.

SQr portable File (spf) is a printer-independent file format that allows for the Production Reporting graphical features, including fonts, lines, boxes, shaded areas, charts, bar codes, and images.

This file format is very useful for saving the output of a report. SPF files can be distributed electronically and read with the Production Reporting Viewer. Producing SPF output also allows you to decide later where to print it. When you are ready to print an SPF file, you can do so with the Production Reporting Viewer or Production Reporting Print.

DECLARE-PRINTER Command

`DECLARE-PRINTER` specifies printer-specific settings for the printers that Production Reporting supports: line printer, PostScript, HP LaserJet, and HTML. `DECLARE-PRINTER` does not cause the report to be produced for a specific printer. To specify a specific format, use one of three methods:

- Use the `-PRINTER:xx` command-line flag. For example `-PRINTER:PS` causes Production Reporting to produce a PostScript output. If your program creates multiple reports, such as the program *ex18a.sqr* from Chapter 19, “Multiple Reports,” the `-PRINTER:xx` flag effects all reports.
- Use the `USE-PRINTER-TYPE` command in your report. You must use this command before you print anything because Production Reporting cannot switch printer type in the middle of a program. `USE-PRINTER-TYPE PS`, for example, causes Production Reporting to produce PostScript output.

- Use the `PRINTER-TYPE` option of the `DECLARE-REPORT` command. `DECLARE-REPORT` is normally used when your program generates multiple reports. See Chapter 19, "Multiple Reports."

For example:

```
declare-report labels
    layout=labels
    printer-type=ps
end-declare
```

causes Production Reporting to produce PostScript output for the labels report.

`DECLARE-PRINTER` defines settings for line printers, PostScript printers, or HP LaserJet printers. Specify the type of printer using the `type` option in `DECLARE-PRINTER` or one of the predefined printers: `DEFAULT-LP`, `DEFAULT-PD`, `DEFAULT-PS`, `DEFAULT-HP`, and `DEFAULT-HT`.

Your program may have multiple `DECLARE-PRINTER` commands if you define settings for each of the printer types. Printer settings only take effect when output is produced. When your program generates multiple reports, you can define settings for each printer for each report. To make `DECLARE-PRINTER` apply to a report, use the `FOR-REPORTS` option.

Naming the Output File

The output file normally shares the name of your program, but with another file extension.

Printers	File Extension
PDF (pd)	pdf
PostScript (PS)	lis
HP Laserjet (HP)	lis
Line Printer (LP)	lis
SQR Portable File	spf

To define another name for the output file (including a user-defined file extension), use the -F option on the command line. For example, to define *chapter1.out* as the output of the program *ex1*a*.sqr*, enter the following command:

```
sqr ex1a username/password -fchapter1.out
```

When your program creates multiple reports, name the output file by using multiple -F flags as follows:

```
sqr ex20a username/password -flabel.lis -fletter.lis -flisting.lis
```

Note that you cannot directly name SPF files. You can still use the -F command-line flag to name the file, but you cannot control the file name extension. For example:

```
sqr ex20a username/password -flabel.lis -fletter.lis -flisting.lis -nolis
```

Note:

Output file names differ if OUTPUT-FILE-MODE is set to LONG in SQR.INI. See “[Default-Settings] Section” in the chanpter titled “SQR.INI” in Volume 2 of the *Production Reporting Developer's Guide* for information.

The `-NOLIS` command-line flag causes Production Reporting to produce SPF files instead of LIS files. The file names are *label*.spf, *letter*.s01, and *listing*.s02. Production Reporting supplies file extensions such as these when your program generates multiple reports.

Operating systems require different techniques for printing the output. On platforms other than Windows, if the output is in SPF format, you first use Production Reporting Print to create the printer-specific file. For example, the following command invokes Production Reporting Print to create a PostScript file *myreport*.lis from the output file *myreport*.spf:

```
sqrp myreport.spf -printer:ps
```

Note that this is a one-way conversion—an SPF file can be turned into a LIS file, but a LIS file cannot be turned into an SPF file.

Print Commands by Operating System

Table 8 summarizes the commands and command-line options used to send report output to the printer. Consult your operating system documentation for details.

Table 8 Print Commands by Operating System

O/S	Command	Command-Line Options
UNIX	lp myreportlis lp myreportlis -d	Use -D for printer destination You can use the UNIX “at” command to schedule the printing time
UNIX BSD	lpr myreportlis	
Windows	Production Reporting prints directly You can also use Production Reporting Print or SQR Production Reporting Studio	Use Oracle's Hyperion® SQR® Production Reporting Studio, Production Reporting Viewer, or Print Setup in Production Reporting Print to choose a printer destination Use Production Reporting Print to print multiple copies You can also use the File Manager Copy command to copy the file to the printer destination (for example, lpt1)

Check with your system administrator about other procedures or commands applicable to printing output files at your site.

31 Using the Production Reporting Command Line

In This Chapter

The Production Reporting Command Line

Command-line flags can be entered on the command line to modify some aspect of program execution or output. Command-line arguments are typically answers to requests (made in the Production Reporting program by `ASK` or `INPUT` commands) for user input.

The syntax of the Production Reporting command line:

```
SQR [program] [connectivity] [flags ...] [args ...] [@file ...]
```

See “Production Reporting Command-Line Arguments” and “Production Reporting Command-line Flags” in Volume 2 of the *Production Reporting Developer's Guide* for detailed information the command-line arguments and flags that you can use with Production Reporting.

Specifying Command-Line Arguments

You can pass an almost unlimited number of command-line arguments to Production Reporting at run time. On some platforms, the operating system imposes a limit on the number of arguments or the total size of the command line. Passing arguments is especially useful in automated reports, such as those invoked by scripts or menu-driven applications.

You can pass arguments to Production Reporting on the command line, in files, or with the environment variable `SQRFLAGS`. When you pass arguments in a file, reference the file name on the command line and put one argument on each line of the file. You thus avoid any limits imposed by the operating system.

To reference a file on the command line, precede its name with the at sign (@) as shown here:

```
sqr myreport sammy/baker arg1 arg2 @file.dat
```

In this example, *arg1* and *arg2* are passed to Production Reporting, followed by the file *file*.dat. Each line in *file*.dat has an additional argument.

How Production Reporting Retrieves the Arguments

When ASK and INPUT execute, Production Reporting checks to see if you entered any arguments on the command line or if an argument file is open. If either has happened, Production Reporting uses this input instead of prompting the user. After the available arguments are used, subsequent ASK or INPUT commands prompt the user for input. If INPUT is used with the BATCH-MODE argument, Production Reporting does not prompt the user, but instead returns a status meaning "No more arguments."

Production Reporting processes all ASK commands before INPUT commands.

Note:

If you compiled your Production Reporting program into an SQT file, ASK commands will have been processed. Use INPUT instead.

Specifying Arguments and Argument Files

You can mix argument files with simple arguments, as shown here:

```
sqr rep2 sammy/baker 18 @argfile1.dat "OH" @argfile2.dat "New York"
```

This command line passes Production Reporting the number 18, the contents of *argfile1.dat*, the value OH, the contents of *argfile2.dat*, and the value "New York", in that order.

The OH argument is in quotes to ensure that Production Reporting uses uppercase OH. When a command-line argument is case-sensitive or contains spaces, it must be enclosed in quotes. Arguments stored in files do not require quotes and cannot contain them; the strings with uppercase characters and any spaces are passed to Production Reporting.

Using an Argument File

If you wanted to print the report on another printer with different characteristics, you could save values for the page sizes, printer initializations, and fonts in separate files and use a command-line argument to specify which file to use. For example, the following command line passes the value 18 to Production Reporting:

```
sqr myreport sammy/baker 18
```

An #INCLUDE command in the report file chooses file *printer18.dat* based on the command-line argument:

```
begin-setup
  ask num                     ! Printer number.
  #include 'printer{num}.dat'                        ! Contains #DEFINE
commands for
                    ! printer and paper width and length
  declare-layout report
          paper-size =({paper_width} {paper_length})
  end-declare
end-setup
```

In this example, the ASK command assigns the value 18 to the variable *num*; 18 is a compile-time argument. The #INCLUDE command then uses the value of *num* to include the file *printer18.dat*, which could include commands similar to the following:

```
! Printer18.dat-definitions for printer in Bldg 4.
#define paper_length 11
#define paper_width 8.5
#define bold_font LS12755
#define light_font LS13377
#define init HM^J73011
```

Passing Command-Line Arguments—Other Approaches

Production Reporting examines an argument file for a program name, user name, or password if none is provided on the command line. The following command line omits the program name, user name, and password:

```
sqr @argfile.dat
```

The first two lines of the argument file for this example contain the program name and user name/password:

```
myreport
sammy/baker
18
OH
...
```

If you do not want to specify the report name, user name, or password on the command line or in an argument file, use the question mark (?). Production Reporting prompts the user to supply these. For example:

```
sqr myreport ? @argfile.dat
```

In this example, the program prompts the user for the user name and password instead of taking them from the first line in the argument file.

You can use multiple question marks on the command line, as shown here:

```
sqr ? ? @argfile.dat
```

In this example, the user is prompted for the program name and user name/password.

Note:

Production Reporting for Windows does not accept the Production Reporting program name and database connectivity to be part of the argument file.

Reserved Characters

The hyphen (-) and at sign (@) characters have special meaning on the command line. The hyphen precedes an Production Reporting flag, and the at sign precedes an argument file name.

To use either of these characters as the first character of a command-line argument, double the character to indicate that it is a literal hyphen or at sign, as shown here:

```
sqr myreport ? --17 @argfile.dat @@X2H44
```

In this example, the double hyphen and double at sign are interpreted as single literal characters.

Creating an Argument File from a Report

You can create an argument file for one program from the output of another program. For example, you could print a list of account numbers to the file *acctlist*.dat, then run a second report with the following command:

```
sqr myreport sammy/baker @acctlist.dat
```

End *acctlist*.dat with a flag such as "END," as shown here:

```
123344
134455
156664
 ...
END
```

A Production Reporting program could use the numbers in *acctlist*.dat with an INPUT command, as shown here:

```
begin-procedure get_company
next:
input $account batch-mode status = #status
  if #status = 3
        goto end_proc
  end-if
begin-select
cust_num, co_name, contact, addr, city, state, zip
  do print-page                    ! Print page with
                    ! complete company data
from customers
where cust_num = $account
end-select
goto next                    ! Get next account number
end_proc:
end-procedure !get_company
```

Using Batch Mode

Production Reporting lets you run reports in batch mode in UNIX and Windows based operating systems.

You can create UNIX shell (sh, csh, ksh) scripts or Windows command (CMD or BAT) files to run Production Reporting. Include the Production Reporting command line in the file just as you would type it at the keyboard.

32 Working with HTML

In This Chapter

Production Reporting Capabilities Available with HTML

The Production Reporting language has a rich set of available features, but some of these features are not available for HTML output due to the limitations of that format.

The Production Reporting features supported under HTML include:

- Images
- Font sizing

  The Production Reporting language specifies font sizes in points. HTML specifies font sizes in a value from one to six. A point size specified in an Production Reporting program is mapped into an appropriate HTML font size.
- Font styles

  The bold and underline font styles are supported.
- Centering.

The Production Reporting features not currently supported for HTML output include:

- Font selection
- Bar codes
- Lines and boxes (Using `-PRINTER:HT`)

Producing HTML Output

Production Reporting can generate two types of HTML files – standard (HTML version 2.0) and enhanced (XHMTL version 1.1). Standard HTML is produced internally by Production Reporting. Enhanced HTML is produced by an external Java-based process.

Specifying the Output Type

Use `-PRINTER:HT` and `-PRINTER:EH` to specify the type of HTML output to produce.

- `-PRINTER:HT` is controlled by the `PrinterHT` setting in the SQR.INI file. If the `PrinterHT` setting is set to *Standard*, `-PRINTER:HT` produces version 2.0 HTML files with the report content inside of <PRE></PRE> tags. If the `PrinterHT` setting is set to *Enhanced*, `-PRINTER:HT` is mapped to `-PRINTER:EH`, and produces version 1.1 XHTML tags.

  For additional information on the `PrinterHT` setting in the SQR.INI file, see "[Default-Settings] Section" in the chapter titled "SQR.INI" in Volume 2 of the *Production Reporting Developer's Guide*.

- `-PRINTER:EH` produces reports in which content is fully formatted with version 1.1 XHTML tags.

  Following is an example of a command that uses `-PRINTER:EH`:

  ```
  sqr myreport.sqr sammy/baker@rome -PRINTER:EH
  ```

Using HTML Procedures to Produce Output

You can use HTML procedures to produce output with a full set of HTML features. See "Using HTML Procedures in a Production Reporting Program" on page 220 for detailed information.

Viewing HTML Output

When you use an Production Reporting program to generate HTML output, the output contains HTML tags. An HTML tag is a character sequence that defines how information is displayed in a Web browser.

HTML output looks something like this:

```
<HTML><HEAD><TITLE>myreport.lis</TITLE></HEAD><BODY>
```

This code is just a portion of the HTML output that Production Reporting generates. The tags it contains indicate the start and end points of HTML formatting.

For example, in the HTML code shown above, the tag <HTML> defines the output that follows as HTML output. The tags <TITLE> and </TITLE> enclose the report title—in this case, *myreport*.lis. The <BODY> tag indicates that the information following it comprises the body of the report.

Output File Types

A Production Reporting report named *myreport.sqr* creates a FRAME file (*myreport*.htm) and report output file(s). The `OUTPUT-FILE-MODE` entry in the [Default-Setting] section of the SQR.INI file controls the report output file extensions. When set to SHORT, the report output files use the form *myreport*.hzz and when set to LONG, the files use the form *myreport_zz*.htm. The value of zz ranges from 00 to 99 and reflects the report number.

The FRAME file shows a list (hypertext links) of report pages in one frame and the report text in another frame. Each report output file contains a list of pages (hypertext links) at the end of the file. If *myreport.sqr* created multiple reports, then the FRAME file contains a link to each report output file. In addition, each report output file contains links to the other report output files that were created during the program run.

Testing the Output

You can preview HTML output produced by a Production Reporting program on a local system. This is a good way to test the output before it is published on a Web site. To test a program's output, open the file in the Web browser. If your Web browser supports the HTML FRAME construct, open the FRAME file (*myreport_frm*.htm*)*; otherwise open the report output file (*myreport*.h00, *myreport_00*.htm).

Generating Enhanced HTML

As was discussed earlier, you can generate Enhanced HTML output from an Production Reporting program (see "Specifying the Output Type" on page 206). Enhanced HTML produces reports in which content is fully formatted with version 1.1 XHTML tags.

The version of HTML used is defined in the `FullHTML` parameter in the [Enhanced-HTML] section in the SQR.INI file. (See "[Enhanced-HTML] Section" in Volume 2 of the Production Reporting Developer's Guide.)

If you have existing SPF files for which you want to generate Enhanced HTML output, it is not necessary to re-run your Production Reporting program. You can invoke Production Reporting Print (sqrp) to output Enhanced HTML from SPF files by using a command similar to:

```
sqrp myreport.spf -PRINTER:EH
```

From within the Production Reporting Viewer, you can also output this high-quality HTML by selecting File, then Save as HTML. The HTML level output from the Production Reporting Viewer is also determined by the your SQR.INI file settings and shares the default value.

You can also generate Enhanced HTML files with precompiled Production Reporting program files, *.sqt* files. Run the SQT file against Production Reporting Execute with a command similar to the following:

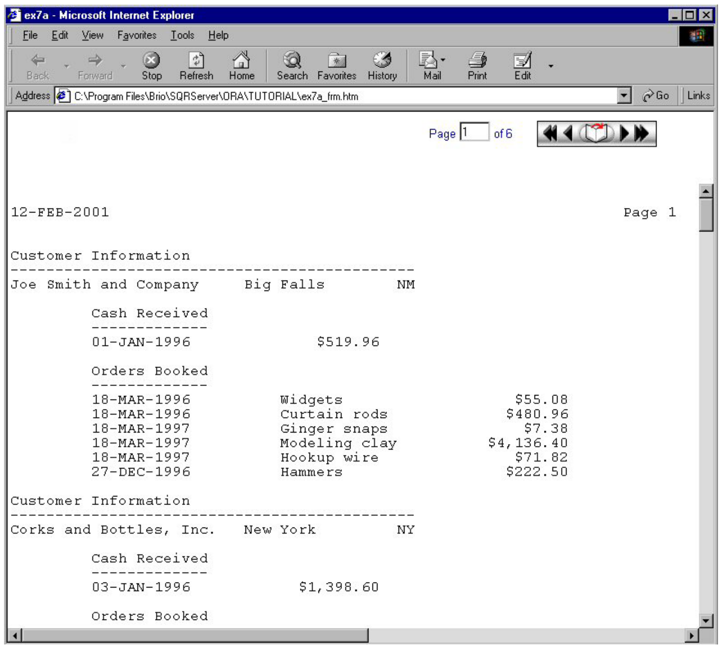
```
sqrt myreport.sqt sammy/baker@rome -PRINTER:EH
```

As is true of executing any SQT file, you can run it against Production Reporting by including the -RT flag. To generate Enhanced HTML, use the `-PRINTER:EH` flag in the command:

```
sqr myreport.sqr sammy/baker@rome -RT -PRINTER:EH
```

Chapter 8, "Master/Detail Reports," contains *Program ex7a.sqr*, which produces a simple master/detail report. By running it using Enhanced HTML, you can produce HTML output which, when viewed from a Web browser, is similar to the following example. Note that a "banner" frame is produced that contains the means to navigate through the report. You can enter a specific page number and press <Enter> on your keyboard (or click "Go!"). You can also use the navigation links to move through the pages in any order you wish—"First", "Last", "Previous", or "Next".

Figure 9 Enhanced HTML Output *Program ex7a.sqr* in a Web browser


ex7a - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address C:\Program Files\Brio\SQRServer\ORA\TUTORIAL\ex7a_frm.htm
Go Links
Page 1 of 6

```
12-FEB-2001                                                          Page 1

Customer Information
-----------------------------------------------
Joe Smith and Company     Big Falls        NM

         Cash Received
         -------------
         01-JAN-1996              $519.96

         Orders Booked
         -------------
         18-MAR-1996          Widgets                $55.08
         18-MAR-1996          Curtain rods          $480.96
         18-MAR-1997          Ginger snaps            $7.38
         18-MAR-1997          Modeling clay       $4,136.40
         18-MAR-1997          Hookup wire            $71.82
         27-DEC-1996          Hammers               $222.50

Customer Information
-----------------------------------------------
Corks and Bottles, Inc.   New York         NY

         Cash Received
         -------------
         03-JAN-1996            $1,398.60

         Orders Booked
```

Note:

When you use the -PRINTER:EH command-line flag (or -PRINTER:HT with the PrinterHT entry in the SQR.INI file set to Enhanced), you can also use additional flags such as -EH_CSV, -EH_CSV:*file*, -EH_Icons:*dir*, and -EH_Scale:{*nn*} to modify the output. These flags only work with -PRINTER:EH. -EH_CSV creates an additional output file in Comma Separated Value (CSV)format. -EH_CSV:*file* associates the CSV icon with the specified file. -EH_Icons:*dir* specifies the directory where the HTML should look for the referenced icons. -EH_Scale:{*nn*} sets the scaling factor from 50 to 200.

Setting Enhanced HTML Attributes

In certain cases, you may want additional control over the Enhanced HTML code that is generated. Production Reporting supports extensions that allow you to control the generated HTML. By using these extensions, you can specify features such as the HTML title, background color (or image), text color, and hyperlinks.

Enhanced HTML extensions also allow you to include your own HTML tags in the output. These tags are passed through to the output without change. You can use this feature to include advanced HTML capabilities such as JavaScript and <APPLET> tags.

Review the following sections for information on:

- Specifying HTML Titles
- Specifying Background Colors
- Specifying Background Images
- Specifying Hyperlinks
- Specifying Text Color
- Specifying HTML Colors
- Including Your Own HTML Tags
- Specifying Table of Contents Attributes
- Specifying Navigation Bar Attributes
- Specifying Cell Borders
- Specifying Expand/Collapse and Filter Features

Specifying HTML Titles

The HTML page title normally appears on the caption bar of the browser window and is also used when creating a bookmark for the page. It is placed between the <TITLE> </TITLE> HTML tags. To specify the title of the HTML page, use the `%%Title` extension at the beginning of your Production Reporting program by entering:

```
Print-Direct Printer=html '%%Title Monthly Sales'
```

Specifying Background Colors

To specify a background color, use the `%%Body-BgColor` extension. Enter code similar to the following at the beginning of your program:

```
Print-Direct Printer=html '%%Body-BgColor #0000FF'
```

To set the background color for the navigation bar, enter code similar to the following:

```
Print-Direct Printer=html '%%Nav-Body-BgColor #0000FF'
```

For information about specifying colors, see "Specifying HTML Colors" on page 210.

Specifying Background Images

To use a background image for the report pages that enhanced HTML generates, insert `%% Background` extension at the beginning of your program:

```
Print-Direct Printer=html '%%Background tile.gif'
```

To set the background image for the navigation bar, enter code similar to the following:

```
Print-Direct Printer=html '%%Nav-Background D:\jpegdir\house.jpg'
```

The background attribute can be any URL. If you do not specify Nav-Background while the body background is specified, then the background image specified for the body is used both in the body and in the navigation bar. If you want to prevent this and want no image to appear in the navigation bar, then you must specify in code similar to the following:

```
Print-Direct printer=html '%%Nav-Background EMPTY'
```

Specifying Hyperlinks

To specify a hyperlink in a report, use the `URL` print format command. The `URL` print format command creates a hypertext link to the specified address. For example:

```
Print "My web page" (40,10) URL="http://www.somewebhost.com/~myusername/index.htm"
```

Creates a link to the following ULR in your report:

```
http://www.somewebhost.com/~myusername/index.htm
```

When you click on the "My web page" your browser is directed to the page.

Note:

When you use the URL command, Production Reporting does not validate the address.

Specifying Text Color

The *`%%Color`* and *`%%ResetColor`* extensions change the color of text. The following code example demonstrates this capability:

```
If &Salary > 100000
Print-Direct Printer=html '%%Color #FF0000'
End-If
Print &Salary ()
If &Salary > 100000
Print-Direct Printer=html '%%ResetColor'
End-If
```

In our example, when the value of the column is over 100000, we print it in red. The *`%%Color`* extension affects all text (and number) printing from this point on. This is similar to the behavior of the `ALTER-PRINTER` command. A subsequent invocation of *`%%Color`* with a different color value sets the current color to the new color. To restore the color back to the default (normally, black) use the *`%%ResetColor`* extension.

Specifying HTML Colors

Specifying color as an RGB hexadecimal value is the only way to designate color in Production Reporting. Your browser documentation should contain a complete listing of supported colors and their hexadecimal values.

To specify color as an RGB hexadecimal value, enter a # character followed by six hexadecimal digits. The first two digits specify the intensity of the red, the next two specify the green, and the last two specify the blue. For example, green would be #00FF00.

Including Your Own HTML Tags

The Production Reporting `PRINT` command with `CODE-PRINTER=HT` provides a means for you to inject any text into the HTML output. Production Reporting makes no attempt to check the text you are printing. This text could have anything that your browser understands.

Be careful however not to try to use this hook for formatting as it is very likely that your formatting conflicts with enhanced HTML formatting. Enhanced HTML makes extensive use of HTML tables.

To gain full control over formatting, you can use the HTML procedures that are defined in *html.inc* and are documented in this chapter and in Volume 2 of the *Production Reporting Developer's Guide*.By invoking the *html_on* procedure, you instruct Enhanced HTML to perform no formatting at all. Then specify all formatting using the HTML procedures in *html.inc* or by using the Production Reporting PRINT command with `CODE-PRINTER=HT` to insert HTML codes.

When you use `PRINT` with `CODE-PRINTER=HT`, Enhanced HTML does not translate special symbols that are used in HTML tags such as ‘<', '>’, and ‘&’.

Note:

The HTML procedures defined in *html.inc* are only supported when you produce HTML 2.0 output. (Use the PRINTER:HT command-line flag and set the PrinterHT entry in the SQR.INI file to Standard.)

Specifying Table of Contents Attributes

To specify an alternate table of contents title for the table, use the *`%%TOC-Title`* extension. Enter code similar to the following in the TOC's `BEGIN-HEADING` section:

```
Print-Direct Printer=html '%%TOC-Title My Table of Contents'
```

To specify alternate colors for the TOC title's text color and background color, use the *`%%TOC-Title-Color`* extension. Enter code similar to the following in the TOC's `BEGIN-HEADING` section:

```
Print-Direct printer=html '%%TOC-TitleColor #FF00FF9999FF0'
```

To specify alternate colors for the TOC item's text color and background color, use the *`%%TOC-Settings`* extension. Enter code similar to the following in the TOC's `BEGIN-HEADING` section:

```
Print-Direct printer=html '%%TOC-Settings #66FF33FF33660'
```

To specify the width of the TOC as a percentage of the entire page, use the *`%%TOC-WIDTH`* extension. Enter code similar to the following in the TOC's `BEGIN-HEADING` section:

```
Print-Direct printer=html '%%TOC-WIDTH 15%'
```

To specify an image for the TOC background, use the `%%TOC-BACKGROUND` extension. Enter code similar to the following in the TOC's `BEGIN-HEADING` section:

```
Print-Direct printer=html '%%TOC-BACKGROUND parchment.gif'
```

The url to image file can be relative.

To specify a color for the TOC's background, use the `%%TOC-BODY-BGCOLOR` extension. Enter code similar to the following in the TOC's `BEGIN-HEADING` section:

```
Print-Direct printer=html '%%TOC-BACKGROUND #ff0066'
```

Specifying Navigation Bar Attributes

To import an image to the navigation bar, enter code similar to the following:

```
Print-Direct Printer=html '%%NAV-IMAGE <url to image file> <X Position> <Y
position> <height in pixels> <width in pixels>'
```

For example:

```
Print-Direct Printer=html '%%Nav-Image logo.gif 15 100 20 40'
```

To set the background color for the navigation bar, enter code similar to the following:

```
Print-Direct Printer=html '%%Nav-Body-BgColor #0000FF'
```

To set the background image for the navigation bar, enter code similar to the following:

```
Print-Direct Printer=html '%%Nav-Background D:\jpegdir\house.jpg'
```

The background attribute can be any URL. If you do not specify Nav-Background while the body background is specified, then the background image specified for the body is used both in the body and in the navigation bar. If you want to prevent this and want no image to appear in the navigation bar, then you must specify in code similar to the following:

```
Print-Direct printer=html '%%Nav-Background EMPTY'
```

Specifying Cell Borders

To specify an alternate cell border color and cell border style for an object, use the `%%Border` and `%%ResetBorder` extensions. Enter code similar to the following before and after printing an object:

```
Print-Direct printer=html '%%Border #9999FF,double,1,112'
Print 'Hello World' (1,1)
Print-Direct printer=html '%%ResetBorder'
```

Specifying Expand/Collapse and Filter Features

To specify HTML Expand/Collapse features on an object, use these extensions:

- `%%Hide`
- `%%Show`

To specify HTML Filter features on an object, use these extensions:

- `%%Filter-Hide`
- `%%Filter-Show`
- `%%Begin-Select`
- `%%End-Select`

Note:

You cannot add expand/collapse or filter functionality to generic Production Reporting reports. To use this type of *streaming* functionality, you must write the Production Reporting report in a streaming fashion, which is contrary to normal Production Reporting operation. The following guidelines apply to expand/collapse and filter features:

- Once an expand/collapse region starts, you cannot print the report by a random vertical position.
  - ❍ Each row must print in order from top to bottom.
  - ❍ Be careful when using expand/collapse functionality in *before* and *after* procedures.
  - ❍ You can print headings above the first expand/collapse region.
- Do not burst a report by page or by section.

  Bursting by page or section 'turns off' expand/collapse functionality.
- Do not use the `POSITION` command to add extra vertical white space between rows for HTML output.
  - ❍ Some vertical white space is ignored.
  - ❍ Vertical white space appears if other printer output types are used.
  - ❍ Vertical white space appears if bursting is used (expand/collapse or filtering is turned off).
- Do not create a collapsible area using multiple prints for the label row.

  In version 8.0, multiple prints to one line are *not* concatenated into one string. To treat multiple columns as one string, use the `CONCAT` command.
- Open and close each region with a hide and show extension.
  - ❍ Production Reporting does not close any open regions at the end of the report; instead, the program must close each region.
  - ❍ Regions must begin with a *hide* extension and end with a *show* extension.
- Do not overlap collapsible regions or filter regions.
  - ❍ Overlapping collapsible or filter regions are not allowed.
  - ❍ Nesting collapsible regions is allowed.

Expand/Collapse Code Example

The following is a complete code example using Expand/Collapse features. The `%%Hide` extension is used in a *before* procedure, and the `%%Show` extension is used in an *after* procedure.

```
Begin-Setup
 Declare-Layout Default
  Orientation = Portrait
  Paper-Size = (8.5,440)
  Top-Margin    = 0.500
  Bottom-Margin = 0.500
  Left-Margin   = 0.500
  Right-Margin  = 0.500
  Line-Height = 1
  Char-Width  = 1
 End-Declare
 Declare-TOC Default
  Entry = BRB_TOC_Proc
 End-Declare
End-Setup
Begin-Heading 24 For-Tocs=(default)
 Graphic (21,1,540) Horz-Line 20
 Alter-Printer Font=4 Point-Size=16    ! [SQR.INI] 4=Arial,proportional
 Print 'Table of Contents' (15,415,17)
 Print-Direct printer=html '%%TOC-Title Table of Contents'
 Alter-Printer Font=4 Point-Size=10
End-Heading
Begin-Procedure BRB_TOC_Proc
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
 Let #indent-size = 24
 Let #indentation = 1 + (#indent-size * (#sqr-toc-level - 1))
 Print $sqr-toc-text (12,#indentation)
 Print #sqr-toc-page (12,523)
 Next-Listing Skiplines=4 Need=14
End-Procedure
Begin-Program
 Position (1,1)
 Do Master_Query
End-Program
Begin-Procedure Master_Query
Begin-Select
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
DNAME &Master_Query_DNAME () On-Break Set=6 Level=1 Print=Never
Before=Master_QueryDNAME_BeforeProc601 After=Master_QueryDNAME_AfterProc601
 Next-Listing
DEPTNO &Master_Query_DEPTNO
 Do Emp(&Master_Query_DEPTNO)
From  DEPT
Order By DNAME
End-Select
 Next-Listing
End-Procedure
Begin-Procedure Master_QueryDNAME_BeforeProc601
 Print-Direct printer=html '%%Hide'
 Next-Listing  Need=12
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
 Print &Master_Query_DNAME (12,1,14)
 Toc-Entry text=&master_query_dname level=1
   Position (+12,)
 Next-Listing
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
End-Procedure
```

```
Begin-Procedure Master_QueryDNAME_AfterProc601
 Next-Listing
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
 Print-Direct printer=html '%%Show'
End-Procedure
Begin-Procedure Emp (#P1_DEPTNO)
Begin-Select
DEPTNO &_Emp_DEPTNO=number
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
ENAME &_Emp_ENAME (17,36,10)
JOB_TITLE &_Emp_JOB_TITLE (17,146,30)
SAL &_Emp_SAL (17,362) Edit 99999.99na
 Next-Listing  Need=17
From  EMP
Where DEPTNO = #P1_DEPTNO
Order By ENAME
End-Select
 Next-Listing
End-Procedure
Begin-Heading 60
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
 Print $current-date (12,1) edit 'MM/DD/YYYY'
 Page-Number (12,520)
 Print 'Dname' (46,1,5) Underline  Bold
 Alter-Printer Font=4 Point-Size=10
End-Heading
```

Filter Code Example

The following is a complete code example using Filter features. The extensions appear in bold.

```
Begin-Setup
 Declare-Layout Default
  Orientation = Portrait
  Paper-Size = (8.5,440)
  Top-Margin    = 0.500
  Bottom-Margin = 0.500
  Left-Margin   = 0.500
  Right-Margin  = 0.500
  Line-Height = 1
  Char-Width  = 1
 End-Declare
 Declare-TOC Default
  Entry = BRB_TOC_Proc
 End-Declare
End-Setup
Begin-Heading 24 For-Tocs=(default)
 Graphic (21,1,540) Horz-Line 20
 Alter-Printer Font=4 Point-Size=16    ! [SQR.INI] 4=Arial,proportional
 Print 'Table of Contents' (15,415,17)
   Print-Direct printer=html '%%TOC-Title Table of Contents'
 Alter-Printer Font=4 Point-Size=10
End-Heading
Begin-Procedure BRB_TOC_Proc
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
 Let #indent-size = 24
 Let #indentation = 1 + (#indent-size * (#sqr-toc-level - 1))
 Print $sqr-toc-text (12,#indentation)
 Print #sqr-toc-page (12,523)
 Next-Listing Skiplines=4 Need=14
End-Procedure
Begin-Program
 Position (1,1)
 Do Master_Query
End-Program
Begin-Procedure Master_Query
Begin-Select
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
DNAME &Master_Query_DNAME () On-Break Set=10 Level=1 Print=Never
Before=Master_QueryDNAME_BeforeProc1001
After=Master_QueryDNAME_AfterProc1001
 Next-Listing
DEPTNO &Master_Query_DEPTNO
 Do Emp(&Master_Query_DEPTNO)
From  DEPT
Order By DNAME
End-Select
 Print-Direct printer=html '%%End-Select'
 Next-Listing
End-Procedure
Begin-Procedure Master_QueryDNAME_BeforeProc1001
 Print-Direct printer=html '%%Begin-Select'
 Print-Direct printer=html '%%Filter-Hide'
 Next-Listing  Need=12
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
 Print &Master_Query_DNAME (12,1,14)
 Toc-Entry text=&master_query_dname level=1
   Position (+12,)
 Next-Listing
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
End-Procedure
Begin-Procedure Master_QueryDNAME_AfterProc1001
 Next-Listing
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
 Print-Direct printer=html '%%Filter-Show'
 Print-Direct printer=html '%%End-Select'
End-Procedure
Begin-Procedure Emp (#P1_DEPTNO)
Begin-Select
DEPTNO &_Emp_DEPTNO=number
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
ENAME &_Emp_ENAME (17,36,10)
JOB_TITLE &_Emp_JOB_TITLE (17,146,30)
SAL &_Emp_SAL (17,362) Edit 99999.99na
 Next-Listing  Need=17
From  EMP
Where DEPTNO = #P1_DEPTNO
Order By ENAME
End-Select
 Print-Direct printer=html '%%End-Select'
 Next-Listing
End-Procedure
Begin-Heading 60
 Alter-Printer Font=4 Point-Size=10    ! [SQR.INI] 4=Arial,proportional
 Print $current-date (12,1) edit 'MM/DD/YYYY'
 Page-Number (12,520)
 Print 'Dname' (46,1,5) Underline  Bold
 Alter-Printer Font=4 Point-Size=10
End-Heading
```

Generating Standard HTML

To generate a report using standard HTML, use these methods:

- Use the `-PRINTER:HT` command-line flag.

  When you use the `-PRINTER:HT` command-line flag, ensure that the PrinterHT setting in the [Default Settings] section of the SQR.INI file is set to *Standard*.

  See "Specifying the Output Type" on page 206.

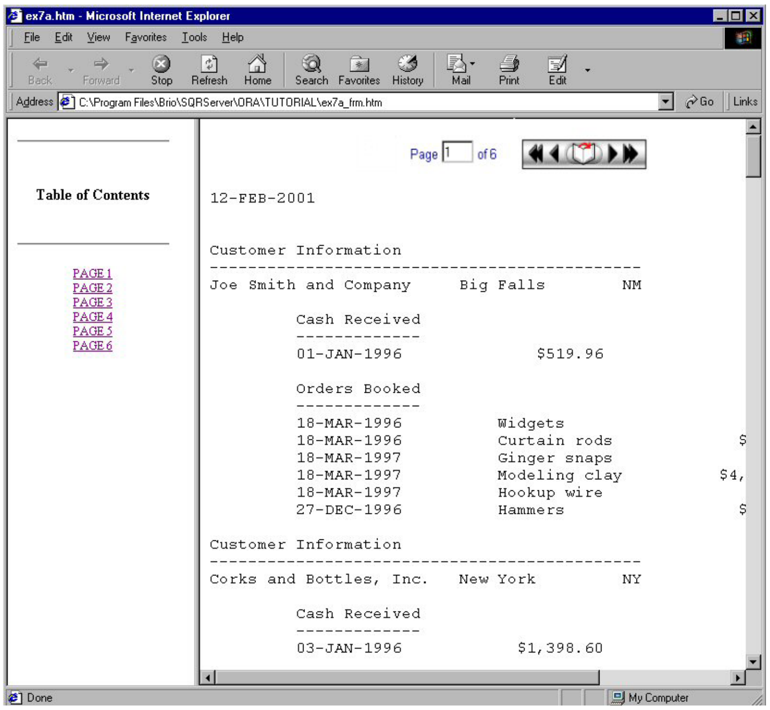
- Use the argument `TYPE-HT` or `USE-PRINTER-TYPE HT` in the `DECLARE-PRINTER` command.

When you produce a report using standard HTML, Production Reporting generates HTML 2.0 files as follows:

- All output displays as preformatted text, using the HTML <PRE> </PRE> tags.
- Text is positioned on the page by the position coordinates specified in the Production Reporting program.
- Text is displayed using a fixed-width font such as Courier.
- Font sizes are mapped to an appropriate HTML font size.
- HTML reserved characters are mapped into the corresponding HTML sequence. The characters <, >, &, " are mapped into the character sequences &*lt;*, &*gt;*, &*amp;*, and &*quot;*, respectively. This prevents the Web browser from mistaking such output as an HTML sequence.

Chapter 8, "Master/Detail Reports," contains a sample program, *Program ex7a.sqr*, which produces a simple master/detail report. By running it with a standard HTML setting, you can produce HTML output which, when viewed from a Web browser, is similar to the following example. Note that a left frame is produced with hyperlinks to each page of the report. The right frame also features a *navigation bar* that appears at the top of every page in the report. The *navigation bar* permits you to move the first page and last page or move one page forward or back from your relative page viewing position.

Figure 10 Standard HTML Output *Program ex7a.sqr* in a Web browser


ex7a.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Go Links
Page 1 of 6
Done
My Computer

Address C:\Program Files\Brio\SQRServer\ORA\TUTORIAL\ex7a_frm.htm

Table of Contents

PAGE 1
PAGE 2
PAGE 3
PAGE 4
PAGE 5
PAGE 6

```
12-FEB-2001


Customer Information
---------------------------------------------
Joe Smith and Company     Big Falls        NM

         Cash Received
         -------------
         01-JAN-1996             $519.96

         Orders Booked
         -------------
         18-MAR-1996          Widgets
         18-MAR-1996          Curtain rods              $
         18-MAR-1997          Ginger snaps
         18-MAR-1997          Modeling clay           $4,
         18-MAR-1997          Hookup wire
         27-DEC-1996          Hammers                   $

Customer Information
---------------------------------------------
Corks and Bottles, Inc.   New York         NY

         Cash Received
         -------------
         03-JAN-1996           $1,398.60
```

“Bursting” and Demand Paging

We have shown you how Production Reporting allows you to generate standard or enhanced HTML reports. But what if you want your HTML files to be smaller for faster load time or divided on the basis of report page ranges? Or, you might want to preview a report’s Table of Contents in you Web browser without generating an entire report. You can do all of these things by using -BURST:{*xx*} in conjunction with `-PRINTER:EH` or `-PRINTER:HT`.

Using `-BURST:P` (or `BURST:P1`) with `-PRINTER:EH` or `-BURST:P1` with `-PRINTER:HT`, you can generate HTML ouput files *burst* by report page numbers—one report page per .htm file. (This is frequently referred to as *demand paging*.) So, if you have a 25 page report, it is divided into 25 separate .htm output files. Using `-PRINTER:HT`, you can also specify the report page ranges you want to see within an HTML file. For example, `-BURST:P0,1,3-5` generates an HTML file containing only report page numbers 1, 3, 4 and 5. You can then focus on information that is truly of interest.

Similarly, if you specify `-PRINTER:HT` with `-BURST:T`, only the Table of Contents file is generated. And, if you specify `-PRINTER:HT` with `-BURST:S`, report output is generated according to symbolic Table of Contents entries. Using `-BURST:S`, you can specify the numeric level to burst on. (for example, `-BURST:S2` bursts on level 2). If you have used `DECLARE-TOC` and `TOC-ENTRY` commands in your Production Reporting program, your Table of Contents provides more detailed information than simple hyperlinked page numbers as illustrated in the following example.

As an example of how simple it is to use `DECLARE-TOC` and `TOC-ENTRY` to improve the information available in generated HTML output, we modified *Program ex7a.sqr*.

We added the following code to the beginning of *Program ex7a.sqr*:

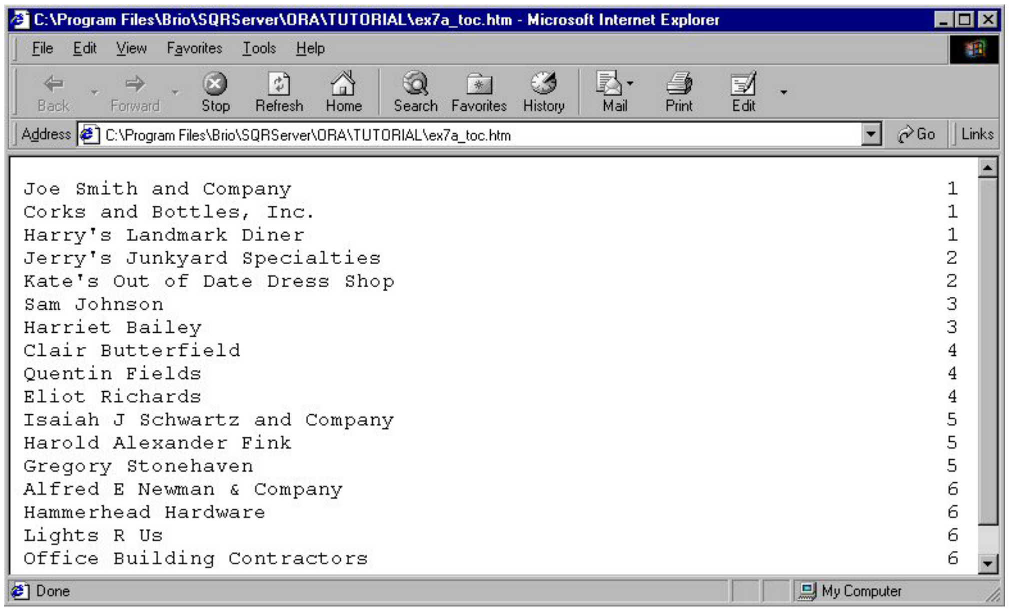
```
begin-setup
declare-toc common
       for-reports=(all)
       dot-leader=yes
       indentation=2
end-declare
end-setup
```

We also added code to the body of the program—in the main procedure immediately after the begin-select and Print 'Customer Information' (,1)

```
toc-entry text = &name
```

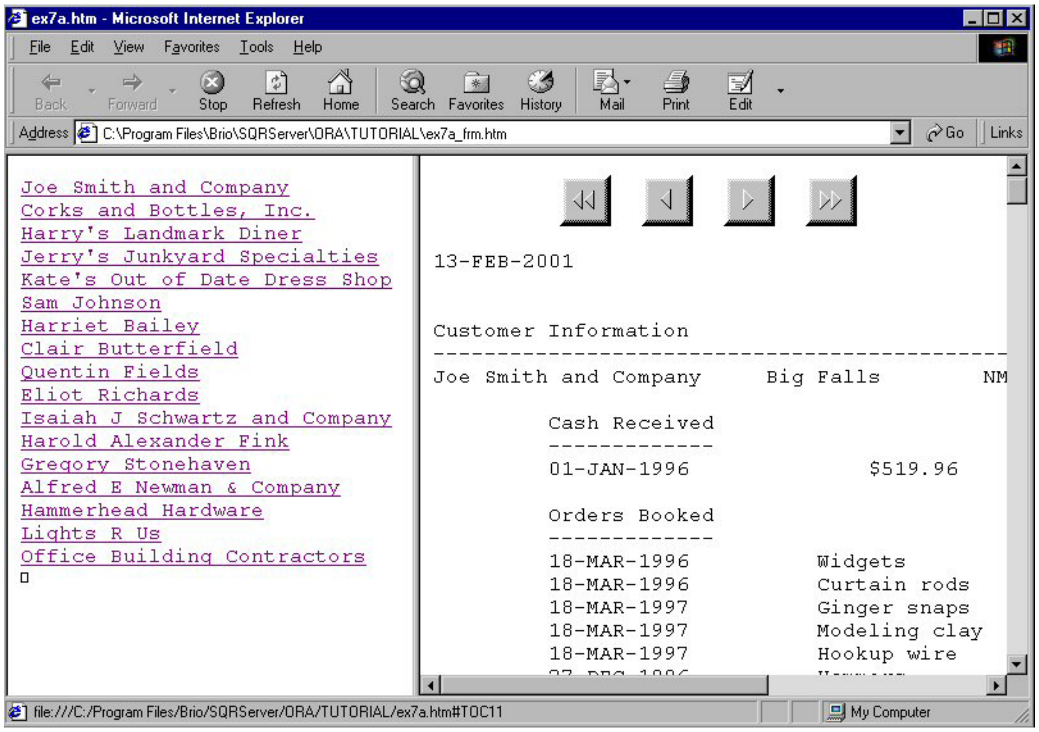
The HTML ouput from the modified *Program ex7a.sqr* is shown in the illustrations that follow.

Figure 11 -PRINTER:HT with -BURST:T Output


C:\Program Files\Brio\SQRServer\ORA\TUTORIAL\ex7a_toc.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address C:\Program Files\Brio\SQRServer\ORA\TUTORIAL\ex7a_toc.htm
Go Links
Done
My Computer

```
Joe Smith and Company                                          1
Corks and Bottles, Inc.                                        1
Harry's Landmark Diner                                         1
Jerry's Junkyard Specialties                                   2
Kate's Out of Date Dress Shop                                  2
Sam Johnson                                                    3
Harriet Bailey                                                 3
Clair Butterfield                                              4
Quentin Fields                                                 4
Eliot Richards                                                 4
Isaiah J Schwartz and Company                                  5
Harold Alexander Fink                                          5
Gregory Stonehaven                                             5
Alfred E Newman & Company                                      6
Hammerhead Hardware                                            6
Lights R Us                                                    6
Office Building Contractors                                    6
```

Figure 12 -PRINTER:HT Output


ex7a.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address C:\Program Files\Brio\SQRServer\ORA\TUTORIAL\ex7a_frm.htm
Go Links
file:///C:/Program Files/Brio/SQRServer/ORA/TUTORIAL/ex7a.htm#TOC11
My Computer

```
Joe Smith and Company
Corks and Bottles, Inc.
Harry's Landmark Diner
Jerry's Junkyard Specialties
Kate's Out of Date Dress Shop
Sam Johnson
Harriet Bailey
Clair Butterfield
Quentin Fields
Eliot Richards
Isaiah J Schwartz and Company
Harold Alexander Fink
Gregory Stonehaven
Alfred E Newman & Company
Hammerhead Hardware
Lights R Us
Office Building Contractors
```

```
13-FEB-2001


Customer Information
---------------------------------------------
Joe Smith and Company     Big Falls         NM

         Cash Received
         -------------
         01-JAN-1996              $519.96

         Orders Booked
         -------------
         18-MAR-1996          Widgets
         18-MAR-1996          Curtain rods
         18-MAR-1997          Ginger snaps
         18-MAR-1997          Modeling clay
         18-MAR-1997          Hookup wire
```

Figure 13 -PRINTER:EH Output Table of Contents File

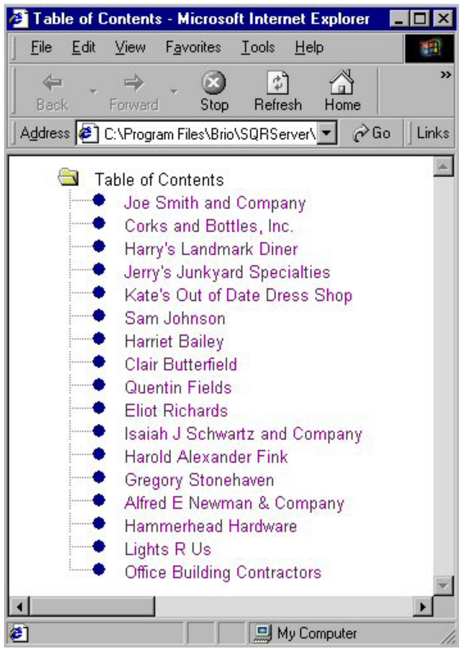


Table of Contents - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address C:\Program Files\Brio\SQRServer\
Go Links
My Computer

Table of Contents
- Joe Smith and Company
- Corks and Bottles, Inc.
- Harry's Landmark Diner
- Jerry's Junkyard Specialties
- Kate's Out of Date Dress Shop
- Sam Johnson
- Harriet Bailey
- Clair Butterfield
- Quentin Fields
- Eliot Richards
- Isaiah J Schwartz and Company
- Harold Alexander Fink
- Gregory Stonehaven
- Alfred E Newman & Company
- Hammerhead Hardware
- Lights R Us
- Office Building Contractors

Using HTML Procedures in a Production Reporting Program

To enhance the appearance of the HTML output, use HTML procedures in an Production Reporting program. Review the following sections for information on:

Using HTML Procedures

Use these guidelines when placing HTML procedures in Production Reporting programs:

- HTML procedures are contained in a file called *html.inc*. To use HTML procedures, the Production Reporting program must include the command:

  ```
  #include 'html.inc'
  ```

  The file *html.inc* is located in the SAMPLE (or SAMPLEW) directory. Use the command-line flag -I to specify its path.
- The Production Reporting program must call the procedure *html_on* at the start of the program. The command that calls this procedure is:

  ```
  do html_on
  ```
- The program must specify a large page length to prevent page breaks. Production Reporting automatically inserts the page navigation hypertext links and an HTML <HR> tag at a page break. If a page break falls in the middle of an HTML construct, such as a table, the output can display incorrectly. Use the command `DECLARE-LAYOUT` with a large `MAX-LINES` setting to prevent page breaks from occurring.

  Note:

  When you use HTML procedures, you must run the Production Reporting program with the command-line flag -PRINTER:HT and the PrinterHT entry in the SQR.INI file set to Standard.

Positioning Objects

When HTML procedures are turned on:

- HTML output is generated without the <PRE></PRE> tags.

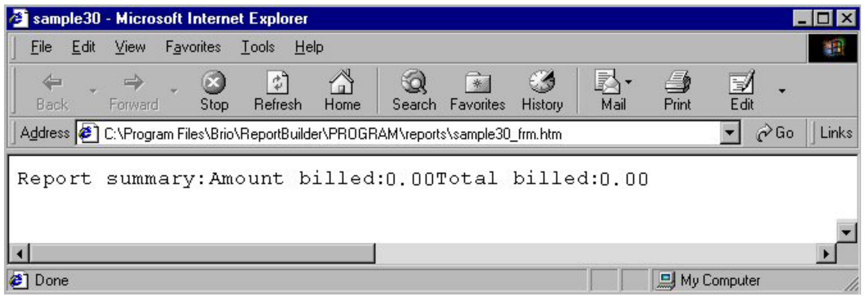
- All position qualifiers in the Production Reporting program are ignored, and program output and HTML tags are placed in the output file in the order in which they are generated, regardless of their position qualifiers.
- The text printed in a `BEGIN-HEADING` section does not appear at the top of the page. Since no positioning is done, text in the heading appears at the bottom.
- White space, such as spaces between `PRINT` commands is removed.

Thus, the HTML procedures must be used to format the report.

The following Production Reporting code does not use the HTML procedures to format the output.

```
print 'Report summary:' (1,1)
print 'Amount billed:' (3,1)
print #amount_amount (3,20)
print 'Total billed:' (4,1)
print #total_amount (4,20)
```

The output from the sample code, as displayed by the Web browser, follows. Note that the text appears on the same line with no spaces between the data.


sample30 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address C:\Program Files\Brio\ReportBuilder\PROGRAM\reports\sample30_frm.htm
Go Links
Done
My Computer

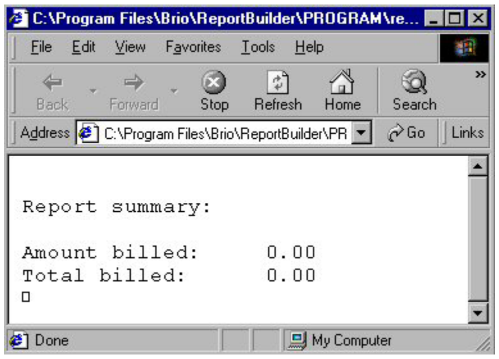
```
Report summary:Amount billed:0.00Total billed:0.00
```

With the HTML procedures for line breaks and a table, the output can be formatted properly.

The following Production Reporting code uses the procedure *html_br* to separate the first two lines of text. The procedures *html_table*, *html_tr*, *html_td*, and *html_table_end* are used to display the totals in a tabular format. Note that an empty string is passed to each procedure as it is called. This empty string is required if no other argument is passed.

```
print 'Report summary:' (1,1)
do html_br(2,'')
do html_table('')
do html_tr('')
do html_td('WIDTH=300')
print 'Amount billed:' (3,1)
do html_td('')
print #amount_amount (3,20)
do html_tr('')
do html_td('WIDTH=300')
print 'Total billed:' (4,1)
do html_td('')
print #total_amount (4,20)
do html_table_end
```

The output from the preceding code is displayed by the Web browser as follows.


C:\Program Files\Brio\ReportBuilder\PROGRAM\re...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address C:\Program Files\Brio\ReportBuilder\PR
Go Links
Done
My Computer

```
Report summary:

Amount billed:      0.00
Total billed:       0.00
```

Defining Titles and Background Images

To define a title and background image for your HTML output, use the HTML procedures *html_set_head_tags* and *html_set_body_attributes*. You must call these procedures at the start of the program. For example:

```
do html_set_head_tags('<TITLE>Monthly Report</TITLE>')
do html_set_body_attributes('BACKGROUND="/images/mylogo.gif"')
```

The first line of this code causes the title "Monthly Report" to display. Specifically, the entire sequence '<TITLE>Monthly Report</TITLE>' is passed as an argument to the procedure *html_set_head_tags*. Note that the argument is enclosed in single quotes.

The second line causes the background image *mylogo*.gif to displayed. Again, an argument is passed to the procedure. Note that the entire argument is enclosed in single quotes, while the file name and path are enclosed in double quotes. Together, these two lines of code generate the following HTML output.

```
<HTML><HEAD><TITLE>My Report</TITLE></HEAD>
<BODY BACKGROUND="/images/mylogo.gif">
```

Table Procedures

When the HTML procedures are turned on, all positioning values in the Production Reporting program are ignored. Thus, the position values cannot be used to display records in a tabular format. To display records in a tabular format use the following procedures:

- **Tables**—Call *html_table* to start a table and *html_table_end* to end a table.
- **Captions**—Call *html_caption* to mark the start of a table caption and *html_caption_end* to mark the end of the table caption. The end is typically implied and *html_caption_end* is not needed, but it can be used for completeness.
- **Rows**—Call *html_tr* to mark the start of a new row in the table and *html_tr_end* to mark the end of the row. The end is typically implied and *html_tr_end* is not needed, but it can be used for completeness.
- **Column headings**—Call *html_th* to mark the start of a column heading and *html_th_end* to mark the end of the column heading. The end is typically implied and *html_th_end* is not needed, but it can be used for completeness.

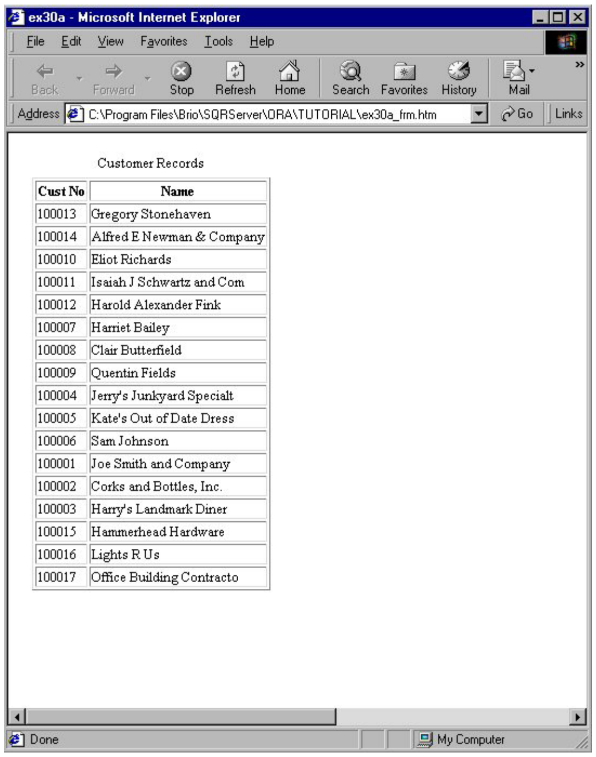
- **Columns**—Call *html_td* to mark the start of a column and *html_td_end* to mark the end of the column. The end is typically implied and *html_td_end* is not needed, but it can be used for completeness.

Program ex28a.sqr uses these table procedures to display information in a tabular format.

Program ex28a.sqr

```
#include 'html.inc'
begin-program
   do main
end-program
! set a large page length to prevent page breaks
begin-setup
  declare-layout default
    max-lines=750
  end-declare
end-setup
begin-procedure main
! turn on HTML procedures
do html_on
! start the table and display the column headings
do html_table('border')
do html_caption('')
print 'Customer Records' (1,1)
do html_tr('')
do html_th('')
print 'Cust No' (+1,1)
do html_th('')
print 'Name'    (,10)
! display each record
begin-select
   do html_tr('')
   do html_td('')
cust_num            (1,1,6) edit 099999
   do html_td('')
name            (1,10,25)
   next-listing skiplines=1 need=1
from customers
end-select
! end the table
do html_table_end
end-procedure
```

Figure 14 Output for *Program ex28a.sqr*


ex30a - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address C:\Program Files\Brio\SQRServer\ORA\TUTORIAL\ex30a_frm.htm
Go Links
Done
My Computer

Customer Records

Cust No	Name
100013	Gregory Stonehaven
100014	Alfred E Newman & Company
100010	Eliot Richards
100011	Isaiah J Schwartz and Com
100012	Harold Alexander Fink
100007	Harriet Bailey
100008	Clair Butterfield
100009	Quentin Fields
100004	Jerry's Junkyard Specialt
100005	Kate's Out of Date Dress
100006	Sam Johnson
100001	Joe Smith and Company
100002	Corks and Bottles, Inc.
100003	Harry's Landmark Diner
100015	Hammerhead Hardware
100016	Lights R Us
100017	Office Building Contracto

Headings

The heading procedures display text using heading levels such as those used in this book. The available heading levels range from one to six; a first-level heading is the highest. To use the heading procedures, call the appropriate heading procedure before the text is output. After the text is output, call the corresponding *end* procedure.

The following Production Reporting code displays text as a second-level heading:

```
do html_h2('')
print 'A Level 2 Heading' (1,1)
do html_h2_end
```

Highlighting

The highlighting procedures provide the ability to display text in the various HTML highlighting styles. Highlighting is also called logical markup.

To use the highlighting procedures, call the appropriate highlighting procedure before the text is output. After the text is output, call the corresponding *end* procedure.

The following highlighting procedures are available:

- **Blink**—Call *html_blink* and *html_blink_end*
- **Citation**—Call *html_cite* and *html_cite_end*

- **Code**—Call *html_code* and *html_code_end*
- **Keyboard**—Call *html_kbd* and *html_kbd_end*
- **Sample**—Call *html_sample* and *html_sample_end*
- **Strike**—Call *html_strike* and *html_strike_end*
- **Subscript**—Call *html_sub* and *html_sub_end*
- **Superscript**—Call *html_sup* and *html_sup_end*

The following Production Reporting code displays text in the subscript style:

```
print 'Here is ' (1,1)
do html_sub('')
print 'subscript' ()
do html_sub_end
print ' text' ()
```

Hypertext Links

The hypertext link procedures provide the ability to create hypertext links and hypertext link anchors. When the user clicks on the hypertext link, the Web browser switches to the top of the specified HTML document, to a point within the specified document, or to a link anchor within the document. A hypertext link can point to the home page of a Web site, for example.

To insert a hypertext link, use the procedure *html_a*, output the information that is to become the hypertext link, and use the procedure *html_a_end* to mark the end of the hypertext link. Two useful attributes for the procedure *html_a*, are the `HREF` and `NAME` attributes. Use the `HREF` attribute to specify where the hypertext link points to. Use the `NAME` attribute to specify an anchor to which a hypertext link can point. These attributes are passed as arguments to the procedure *html_a*.

The following Production Reporting code creates an anchor and two hypertext links. The anchor is positioned at the top of the document. The first hypertext link points to the HTML document *home.html*. The second hypertext link points to the anchor named *TOP* in the current document. Note the pound sign (#) in the argument, which indicates that the named anchor is a point within a document. The third link points to an anchor named *POINT1* in the document *mydoc*.html.

```
do html_a('HREF=home.html')
print 'Goto home page' ()
do html_a_end
do html_a('NAME=TOP')
do html_a_end
print 'At the top of document' ()
do html_br(40, '')
print 'At the bottom of document' ()
do html_p('')
do html_a('HREF=#TOP')
print 'Goto top of document' ()
do html_a_end
do html_a ('HREF=mydoc.html#POINT1')
print 'Goto point1 in mydoc.html' ()
do html_a_end
```

Images

An image can be included in an HTML output with the PRINT-IMAGE command or the procedure *html_img*. Both of these produce the HTML <IMG> tag.

The PRINT-IMAGE command displays images for all printer types but only allows you to specify the image type and source. The *html_img* procedure displays images only for HTML printer type but it allows you to specify any of the attributes available for an HTML <IMG> tag.

For HTML output, only files of the GIF or JPEG format can be used. With PRINT-IMAGE, use the argument TYPE=GIF-FILE or TYPE=JPEG-FILE, respectively.

Lists

The list procedures display lists. To use these procedures, call the appropriate procedure before the list is output. After the list is output, call the corresponding *end* procedure.

The following list procedures are available:

- **Definition (for lists of terms and their definitions)**—Call *html_dl* and *html_dl_end*
- **Directory**—Call *html_dir* and *html_dir_end*
- **Menus**—Call *html_menu* and *html_menu_end*
- **Ordered (numbered or lettered) lists**—Call *html_ol* and *html_ol_end*
- **Unordered (bulleted) lists**—Call *html_ul* and *html_ul_end*

To display a list, except for the definition list, call the appropriate list procedure before starting the output. Call *html_li* to identify each item in the list; you can also call *html_li_end* for completeness. After specifying the output, call the corresponding *end* procedure.

The following code displays an ordered list:

```
do html_ol('')
do html_li('')
print 'First item in list' (1,1)
do html_li_end
do html_li('')
print 'Second item in list' (+1,1)
do html_li_end
do html_li('')
print 'Last item in list' (+1,1)
do html_li_end
do html_ol_end
```

To display a definition list call *html_dl* before starting the output. Call *html_dt* to identify a term and *html_dd* to identify a definition. After specifying the output, call *html_dl_end*. You can also call *html_dd_end* and *html_dt_end* for completeness.

The following code displays a definition list:

```
do html_dl('')
do html_dt('')
print 'A daisy' (1,1)
do html_dt_end
do html_dd('')
print 'A sweet and innocent flower' (+1,1)
do html_dd_end
do html_dt('')
print 'A rose' (+1,1)
do html_dt_end
do html_dd('')
print 'A very passionate flower' (+1,1)
do html_dd_end
do html_ol_end
```

Paragraph Formatting

The HTML procedures provide various paragraph-formatting capabilities. To use these procedures, call the appropriate paragraph procedure before the list is output.

The following procedures are available:

- **Paragraph breaks—**Call *html_p* to mark the start of a paragraph and *html_p_end* to mark the end. Many HTML constructs imply an end of paragraph; thus, the procedure *html_th_end* is not needed, but it can be used for completeness.
- **Line breaks—**Call *html_br* to insert a line break.
- **Horizontal dividers (usually a sculpted line)—**Call *html_hr* to insert a horizontal divider.
- **Prevent line wrapping—**Call *html_nobr* to mark the start of a section of text that cannot be wrapped by the Web browser to fit the width of the browser window. Use the procedure *html_nobr_end* to mark the end.

The following code uses the paragraph-formatting procedures to format text into paragraphs:

```
print 'Here is some normal text' (1,1)
do html_p('ALIGN=RIGHT')
print 'Here is right aligned text' (+1,1)
do html_br(1,'')
print 'and a line break' (+1,1)
do html_p_end
do html_hr('')
do html_nobr('')
print 'A very long line of text that cannot be wrapped' (+1,1)
do html_nobr_end
```

User-Defined HTML

You can incorporate your own HTML tags into the HTML output. To do so, use the `PRINT` command with the argument `CODE-PRINTER=HT`.

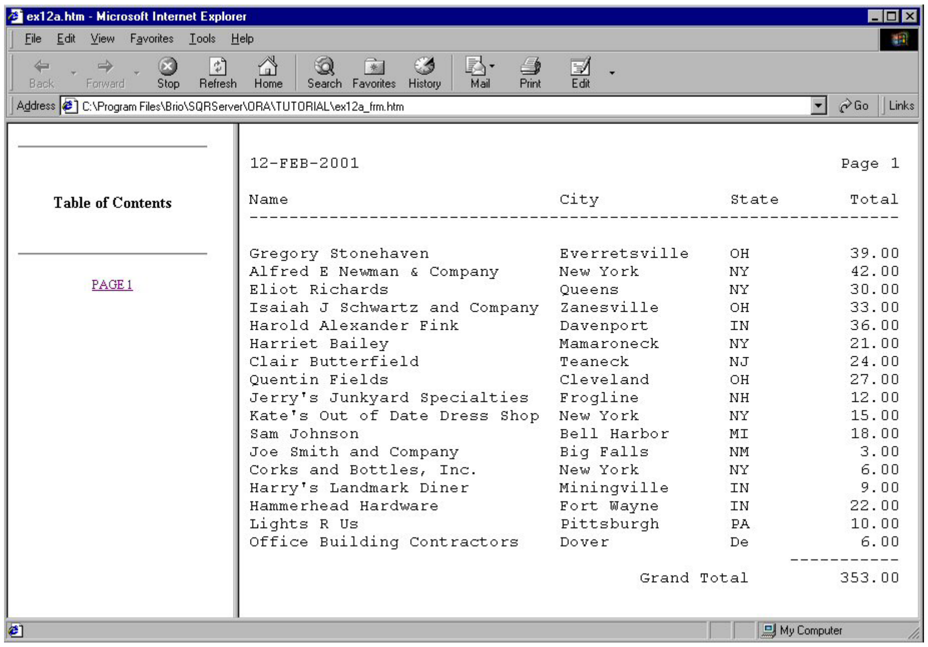
Text printed with this argument is placed only in the HTML output generated when the HTML printer type is specified. With all other printer types, the text is not placed in the output. In addition, the specified text is placed directly in the HTML output without any modifications, such as the mapping of reserved characters.

The following Production Reporting code uses the HTML <*B*> tag to print bold text:

```
print '<B>' () code-printer=ht
print 'Bold text' ()
print '</B>' () code-printer=ht
```

Modifying Existing Production Reporting Programs

In this section, an existing Production Reporting program is modified to use HTML procedures. First, examine the output from when this program is run without modifications using -PRINTER:HT. Three HTML files are generated. If your Web browser supports HTML frames, you should see the following:


ex12a.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address C:\Program Files\Brio\SQRServer\ORA\TUTORIAL\ex12a_frm.htm
Go Links
Table of Contents
PAGE 1
My Computer

```
12-FEB-2001                                                      Page  1

Name                             City              State        Total
-----------------------------------------------------------------------

Gregory Stonehaven               Everretsville     OH           39.00
Alfred E Newman & Company        New York          NY           42.00
Eliot Richards                   Queens            NY           30.00
Isaiah J Schwartz and Company    Zanesville        OH           33.00
Harold Alexander Fink            Davenport         IN           36.00
Harriet Bailey                   Mamaroneck        NY           21.00
Clair Butterfield                Teaneck           NJ           24.00
Quentin Fields                   Cleveland         OH           27.00
Jerry's Junkyard Specialties     Frogline          NH           12.00
Kate's Out of Date Dress Shop    New York          NY           15.00
Sam Johnson                      Bell Harbor       MI           18.00
Joe Smith and Company            Big Falls         NM            3.00
Corks and Bottles, Inc.          New York          NY            6.00
Harry's Landmark Diner           Miningville       IN            9.00
Hammerhead Hardware              Fort Wayne        IN           22.00
Lights R Us                      Pittsburgh        PA           10.00
Office Building Contractors      Dover             De            6.00
                                                           -----------
                                         Grand Total          353.00
```

Program ex28b.sqr

```
#include 'html.inc'
begin-setup
  declare-layout default
    max-lines=10000
  end-declare
end-setup
begin-program
 do main
end-program
begin-procedure main
do html_on
print $current-date (1,1) edit 'DD-MON-YYYY'
do html_p('')
do html_table('BORDER')
do html_tr('')
do html_th('WIDTH=250')
print 'Name'    (3,1)
do html_th('WIDTH=120')
print 'City'    (,32)
do html_th('WIDTH=60')
print 'State'   (,49)
```

```
do html_th('WIDTH=90')
print 'Total'  (,61)
begin-select
  do html_tr('')
  do html_td('')
name  (,1,30)
  do html_td('')
city  (,+1,16)
  do html_td('')
state (,+1,5)
  do html_td('ALIGN=RIGHT')
tot   (,+1,11) edit 99999999.99
  next-listing no-advance need=1
  let #grand_total = #grand_total + &tot
from customers
end-select
  do html_tr('')
  do html_tr('')
  do html_td('COLSPAN=3 ALIGN=RIGHT')
print 'Grand Total' (+1,40)
  do html_td('ALIGN=RIGHT')
print #grand_total (,55,11) edit 99999999.99
do html_table_end
end-procedure ! main
```

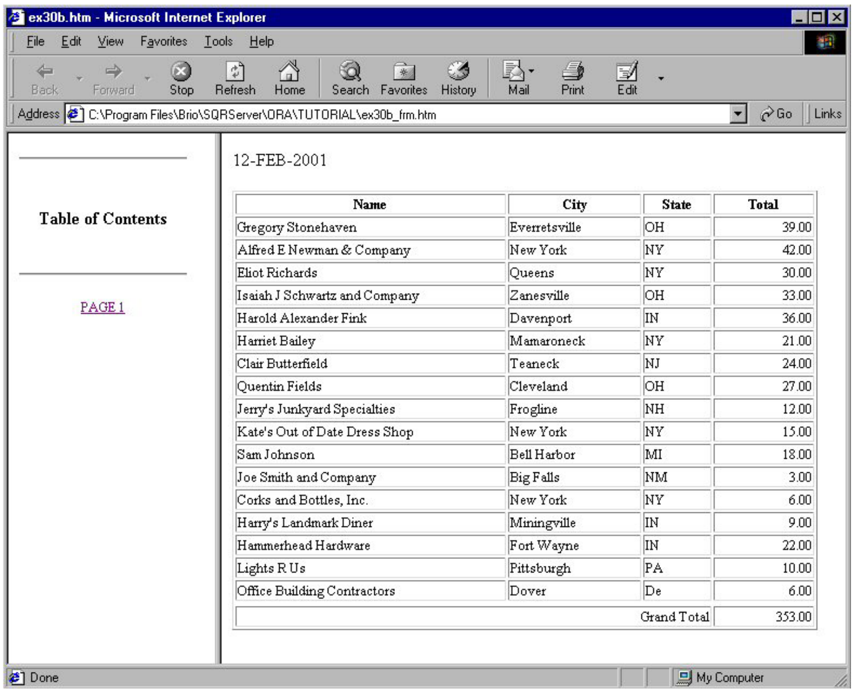
In this code, a DECLARE-LAYOUT command with a large page length setting specified in the MAX-LINES argument is issued to prevent page breaks.

The procedure *html_on* is used to turn on the HTML procedures.

The procedures *html_table*, *html_tr*, *html_td*, and *html_th* are used to position the information in a tabular format. Note the arguments passed to the HTML procedures. BORDER produces the sculpted border seen in the output that follows. WIDTH defines the width of the columns. ALIGN right-aligns the text in the *Total* column. COLSPAN causes the label *Grand Total* to be spanned beneath three columns of data.

Instead of using a HEADING section, the procedure *tr_th* is used to display column headings.

Figure 15 Output for *Program ex28b.sqr*


ex30b.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address C:\Program Files\Brio\SQRServer\ORA\TUTORIAL\ex30b_frm.htm
Go Links
Table of Contents
PAGE 1
Done
My Computer

12-FEB-2001

Name	City	State	Total
Gregory Stonehaven	Everretsville	OH	39.00
Alfred E Newman & Company	New York	NY	42.00
Eliot Richards	Queens	NY	30.00
Isaiah J Schwartz and Company	Zanesville	OH	33.00
Harold Alexander Fink	Davenport	IN	36.00
Harriet Bailey	Mamaroneck	NY	21.00
Clair Butterfield	Teaneck	NJ	24.00
Quentin Fields	Cleveland	OH	27.00
Jerry's Junkyard Specialties	Frogline	NH	12.00
Kate's Out of Date Dress Shop	New York	NY	15.00
Sam Johnson	Bell Harbor	MI	18.00
Joe Smith and Company	Big Falls	NM	3.00
Corks and Bottles, Inc.	New York	NY	6.00
Harry's Landmark Diner	Miningville	IN	9.00
Hammerhead Hardware	Fort Wayne	IN	22.00
Lights R Us	Pittsburgh	PA	10.00
Office Building Contractors	Dover	De	6.00
		Grand Total	353.00

Publishing Reports

A report generated by an Production Reporting program can be published onto the Web site. Thereafter, the user of a Web browser can view the report over the Internet or an Intranet by specifying the report's URL address.

➤ To publish a report:

1 **Run the Production Reporting program.**

2 **Determine where the report output is stored on the Web server.**

The directory must be one that is pointed to by a URL on your server. See your Webmaster for more details on creating a URL.

3 **Copy the generated HTML output files to the chosen directory on the Web server.**

If the output is generated on a client PC, use a utility such as FTP to transfer the HTML output files to the Web server.

If you choose the zip file option, a zip file is created for the generated HTML output in addition to the files being placed in the file system.

4 **Create hypertext links in a home page or other Web site that point to the report files so users browsing the network can navigate to the report and view it.**

To support older Web browsers that do not support the HTML FRAME construct, create two separate hypertext links—one pointing to the FRAME file (.htm) and labeled to indicate *frame version*, and another pointing to the report output file and labeled to indicate nonframe *version*. If the report was created with HTML procedures, however, it should only contain one page. In that case, a listing of report pages contained in the FRAME file would not be needed. Only the report output file would be required for publication on a Web site.

Viewing Published Reports

You use a Web browser to view a report that is published onto a Web site. To do this, specify a URL address in your Web browser, for example:

```
http://www.myserver.com/myreport.htm
```

Publishing Using an Automated Process

The Webmaster can create a program that automates the publishing process. The program should run the Production Reporting program and copy the output to the appropriate location. You can even launch the program using a scheduling utility to automatically run the program and publish it on the Web site at specified times. See the documentation of your scheduling software for more details.

The sample Bourne shell program that follows performs these tasks:

- Sets the necessary environment variables.
- Runs the Production Reporting program */usr2/reports/myreport.sqr* and generates the output files */usr2/reports/myreport.htm* and */usr2/reports/myreport.h00*.
- Specifies */dev/null* as the source of standard input to prevent the program from hanging if it requires input.
- Redirects the standard output to */usr2/reports/myreport.out* to capture any status messages. The output file can be viewed at a later time to diagnose any problems.
- Copies the generated report files to the directory */usr2/web/docs* to publish it on the Web server. (Use the directory name appropriate for your server.)

Here is the code:

```
#! /bin/sh
# set the appropriate environment values
ORACLE_SID=oracle7; export ORACLE_SID
ORACLE_HOME=/usr2/oracle7; export ORACLE_HOME
SQRDIR=/usr2/sqr/bin; export SQRDIR
# invoke the Production Reporting program
sqr /usr2/reports/myreport.sqr orauser/orapasswd \
    -PRINTER:ht -I$SQRDIR \
    > /usr2/reports/myreport.out 2>&1 < /dev/null
# copy over the output
cp /usr2/reports/myreport.htm /usr2/web/docs
cp /usr2/reports/myreport.h00 /usr2/web/docs
```

Note:

The environment variables and the file names must be adjusted to fit your environment.

Publishing Using a CGI Script

In the CGI script method, the user of a Web browser can run an Production Reporting report and view the output. One way to allow the user to run an Production Reporting report is by providing a fill-out form.

When you run an Production Reporting report through a Web site, the following process occurs:

1. The user of the Web browser navigates to a fill-out form.
2. The user enters information on the fill-out form and presses a button to invoke the CGI script.
3. The CGI script runs the Production Reporting program.
4. The CGI script copies the report output file to the standard output.
5. The user views the report.

This process requires these items:

- The fill-out form
- The CGI script
- The Production Reporting program

Creating the Fill-Out Form

This section explains how to create an HTML fill-out form that allows the user to enter some values and launch the request.

To implement an HTML fill-out form, see HTML documentation available in print or on the Internet.

The following HTML code defines a fill-out form with three radio buttons and a submit button. The radio buttons allow the user to specify the sorting criteria. The submit button invokes the CGI script.

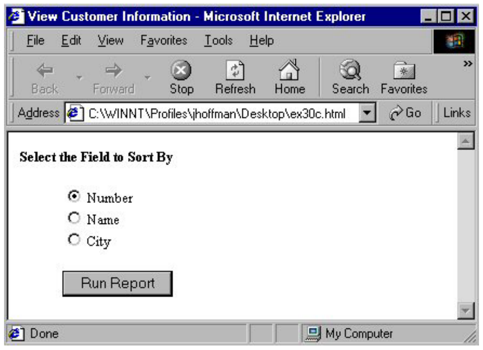
Here is the HTML code:

```
<HTML>
<TITLE>View Customer Information</TITLE>
<FORM METHOD=POST ACTION="/cgi-bin/myreport.sh">
<B>Select the Field to Sort By</B><P><DIR>
<INPUT TYPE="radio" NAME="rb1" VALUE="cust_num" CHECKED> Number<BR>
<INPUT TYPE="radio" NAME="rb1" VALUE="name"> Name<BR>
<INPUT TYPE="radio" NAME="rb1" VALUE="city"> City<BR>
<P><INPUT TYPE="submit" NAME="run" VALUE="Run Report"></DIR>
</FORM>
</HTML>
```

The FORM METHOD tag specifies that the CGI script */cgi-bin/myreport.sh* is invoked when the submit button is pressed. You must adjust the URL address of the CGI script to fit your environment.

In the INPUT tags, the attribute *TYPE="radio"* defines a radio button. The VALUE attribute of the selected radio button is passed by way of the CGI script to the Production Reporting program, as shown in the following example.

The fill-out form looks like the form shown here.


View Customer Information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address C:\WINNT\Profiles\jhoffman\Desktop\ex30c.html Go Links
Done
My Computer

Select the Field to Sort By

[✓ Number]
Name
City

Run Report

Creating the CGI Script

The CGI script is launched when a user makes a request from a fill-out form. A CGI script can be any executable program. We do not recommend that Production Reporting be called directly as a CGI script—a Perl script, a shell script, or a C program all provide simpler routines for processing as a CGI script.

To implement a CGI script, see the HTML documentation available in print or on the Internet.

The CGI script performs these tasks:

- Reads the contents of the standard input stream and parses it to obtain the values entered on the fill-out form. If the fill-out form has no input fields, this step is not required.
- Identifies the output as being in HTML format by outputting the string "Content-type: text/html" along with an extra empty line to the standard output stream.
- Invokes the Production Reporting program. Values entered by the user on the fill-out form are passed to the Production Reporting program by way of the CGI script and the command line.
- Outputs the generated LIS file to the standard output stream. The HTM file is not used because it points to the LIS file with a relative URL address. The relative address does not tell the Web browser where to find the LIS file. We also recommend that you make provisions within your Production Reporting program to output an error message.

The following Bourne shell is an example of a CGI script.

```
#! /bin/sh
# set the appropriate environment values
ORACLE_SID=oracle7; export ORACLE_SID
ORACLE_HOME=/usr2/oracle7; export ORACLE_HOME
SQRDIR=/usr2/sqr/bin; export SQRDIR
# identify the output as being HTML format
echo "Content-type: text/html"
echo ""
# get values from fill-out form using the POST method
read TEMPSTR
```

```
SORTBY=`echo $TEMPSTR | sed "s;.*rb1=;;
s;&.*;;"`
# invoke the Production Reporting program
sqr7 /usr2/reports/myreport.sqr orauser/orapasswd \
    -PRINTER:ht -f/tmp/myreport$$.lis -I$SQRDIR "$SORTBY" \
    > /tmp/myreport$$.out 2>&1 < /dev/null
if [ $? -eq 0 ]; then
   # display the output
   cat /tmp/myreport$$.lis
else
   # error ocurred, display the error
   echo "<HTML><BODY><PRE>"
   echo "FAILED TO RUN Production Reporting PROGRAM"
   cat /tmp/myreport$$.out
   echo "</PRE></BODY></HTML>"
fi
# remove temp files
rm /tmp/myreport$$.*
```

This script first sets the necessary environment variables. Next, it outputs the string *Content-type: text/html* along with an extra empty line to the standard output stream to identify the text as being HTML format.

The script retrieves the value of the selected radio button into the variable *SORTBY*. The value is passed to the Production Reporting program on the command line.

The script runs the Production Reporting program. The report file */usr2/reports/myreport.sqr* is used and the file */tmp/myreport$$.lis* is generated. In addition, the script redirects the standard input from */dev/null* to prevent the program from hanging if the program requires any input. It also redirects the standard output to */tmp/myreport$$.out* to capture any status messages. The $$ is the process ID of the program and is used as a unique identifier to prevent any multiuser problems.

The script then copies the generated report file to the standard output stream. If an error occurs it outputs the status message file instead to allow the user to view the status messages. Finally, it deletes any temporary files.

Passing Arguments to the Production Reporting Program

The Production Reporting program must be modified to accept values entered by the user on the fill-out form.

The following code is the procedure *main* from *Program ex28b.sqr*. It is modified to use the `SORT BY` value passed from the CGI script. The *$sortby* variable is obtained from the command line with an `INPUT` command and used as dynamic variables in the `ORDER BY` clause. The modified lines are shown in bold.

```
begin-procedure main
input $sortby 'Sort by' type=char
do html_on
do html_table('')
do html_tr('')
do html_th('')
print 'Name'   (3,1)
```

```
do html_th('')
print 'City'   (,32)
do html_th('')
print 'State'  (,49)
begin-select
  do html_tr('')
  do html_td('')
name  (,1,30)
  do html_td('')
city  (,+1,16)
  do html_td('')
state (,+1,5)
next-listing no-advance need=1
 let #grand_total = #grand_total + &tot
from customers
order by [$sortby]
end-select
```

33

Tables of Contents

In This Chapter

DECLARE-TOC

DECLARE-TOC defines a Table of Contents and its attributes. When generating multiple reports and Tables of Contents from one Production Reporting program, you can also use the TOC argument in DECLARE-REPORT.

DECLARE-TOC must be issued in your program's SETUP section similar to the following example:

```
begin-setup
        declare-toc toc_name
            for-reports = (all)
            dot-leader = yes
            indentation = 2
        end-declare
     .
     .
     .
end-setup
```

After DECLARE-TOC, specify a Table of Contents name. The FOR-REPORTS argument allows you to specify the reports within the Production Reporting program that use this Table of Contents. (Use (all) if you want the reports to use one Table of Contents.) Specifying individual report names is only necessary when you are generating multiple reports with different TOCs from one program. DOT-LEADER specifies whether or not a dot leader precedes the page number. (The default setting is NO and the DOT-LEADER is suppressed in all HTML output except when -BURST:T with -PRINTER:HT are also specified.) INDENTATION specifies the number of spaces that each level is indented by. (The default setting is 4.)

DECLARE-TOC also supports procedures frequently used for setup and initialization purposes: BEFORE-TOC, AFTER-TOC, BEFORE-PAGE, and AFTER-PAGE. BEFORE-TOC specifies a procedure to be executed before the Table of Contents is generated. If no Table of Contents is generated, the procedure does not execute. AFTER-TOC specifies a procedure to be executed after the Table of Contents is generated. If no Table of Contents is generated, the procedure does not execute. BEFORE-PAGE specifies a procedure to be executed at the start of every page. AFTER-PAGE specifies a procedure to be executed at the end of each page.

TOC-ENTRY

TOC-ENTRY places an entry into the Table of Contents and takes the mandatory argument TEXT which specifies the text to be placed in the Table of Contents. Legal text includes text literals, variables, and columns. To include levels in your Table of Contents, use the LEVEL argument, which specifies the level at which to place the text. (If this argument is not specified, the previous level's value is used.)

If you are writing programs that generate multiple reports, you have some options to choose from. As previously mentioned, you can use the FOR-REPORTS argument of the DECLARE-TOC command to identify the reports to which the DECLARE-TOC applies. Alternatively, you can use the TOC argument of the DECLARE-REPORT command to specify the name of the Table of Contents you want the report to use. Your program can have multiple DECLARE-TOC statements and multiple DECLARE-REPORT statements. However, you must include the FOR-TOCS argument in your DECLARE-TOC statements or the TOC argument in your DECLARE-REPORT statements.

To specify the name of the Table of Contents applicable to a given report using the TOC argument of the DECLARE-REPORT command, include code similar to the following in the SETUP section of your program:

```
begin-setup
        declare-report
            toc = toc_name
        end-declare
     .
     .
end-setup
```

In Chapter 30, "Printing Issues," we modified *ex7a.sqr* to use the DECLARE-TOC and TOC-ENTRY commands. Then, we generated HTML output from the modified program using the -PRINTER:EH and -PRINTER:HT command-line flags. Under HTML, the Table of Contents file is a hyper-linked point of navigation for the online report.

But, you may also want to generate output files for printing reports on paper. And the Table of Contents features work here as well. To test this, run the modified version of *ex7a.sqr* program from Chapter 30, "Printing Issues," and print it from a LIS file (or use -PRINTER:WP on Windows). The Table of Contents output contain the traditional dot leaders and necessary page numbers relating to a hardcopy report.

Cust.sqr

The following program is based on *cust.sqr*, which is located in the SAMPLE (or SAMPLEW) directory. The program identifies the Table of Contents with the specific name *cust_toc*. The dot leader is turned ON. Indentation is set to 3. One Table of Contents level is set using the TOC-

ENTRY command's LEVEL=1 argument. The BEFORE-PAGE and AFTER-TOC arguments of the DECLARE-TOC command are used to print simple messages here.

Table of Contents Sample Program #1

```
begin-setup
  declare-toc cust_toc
    for-reports=(all)
    dot-leader=yes
    indentation=3
    after-toc=after_toc
    before-page=before_page
  end-declare
end-setup
begin-program
  do main
end-program
begin-procedure after_toc
  position (+1,1)
  print 'After TOC' () bold
  position (+1,1)
end-procedure
begin-procedure before_page
  position (+1,1)
  print 'Before Page' () bold
  position (+1,1)
end-procedure
begin-procedure main
begin-select
  print 'Customer Info' ()
  print '-'        (+1,1,62) Fill
name     (+1,1,25)
  toc-entry text = &name level = 1
cust_num (,35,30)
city     (+1,1,16)
state    (,17,2)
phone    (+1,1,15) edit (xxx)bxxx-xxxx
  position (+2,1)
from customers
order by name
end-select
end-procedure       ! main
begin-heading 3
 print $current-date (1,1) Edit 'DD-MON-YYYY'
 page-number (1,69) 'Page '
end-heading
```

The following program is also based on *cust.sqr*. It is similar to the previous program but declares two Table of Contents levels. This program also creates Table of Contents specific headings and footings. The FOR-TOCS argument of the BEGIN-HEADING and BEGIN-FOOTING commands allows you to specify, by name, the Table of Contents to which the heading or footing section applies. So, if your program is generating multiple reports with multiple Tables of Contents, you can apply unique or common headings and footings to reports and Tables of Contents.

The Table of Contents heading of this program prints *Table of Contents* and the page number. The page numbers in the Table of Contents print as roman numerals. The Table of Contents footing prints *Company Confidential.*

Table of Contents Sample Program #2

```
begin-setup
  declare-report cust
  end-declare
  declare-toc cust_toc
    for-reports=(cust)
    dot-leader=yes
    indentation=3
    after-toc=after_toc
    before-page=before_page
  end-declare
  declare-variable
    integer #num_toc
    integer #num_page
  end-declare
end-setup
begin-program
  use-report cust
  do main
end-program
begin-procedure after_toc
  position (+1,1)
  print 'After TOC' () bold
  position (+1,1)
end-procedure
begin-procedure before_page
  position (+1,1)
  print 'Before Page' () bold
  position (+1,1)
end-procedure
begin-procedure main
begin-select
  print 'Customer Info' ()
  print '-'                (+1,1,62) Fill
name     (+1,1,25)
  toc-entry text = &name level = 1
cust_num (,35,30)
city     (+1,1,16)
state    (,17,2)
phone    (+1,1,15) edit (xxx)bxxx-xxxx
  position (+2,1)
  do orders(&cust_num)
  position (+2,1)
from customers
order by name
end-select
end-procedure ! main
begin-procedure orders (#cust_num)
  let #any = 0
begin-select
  if not #any
```

```
    print 'Orders Booked' (+2,10)
    print '-------------' (+1,10)
    let #any = 1
  end-if
b.order_num
b.product_code
order_date                (+1,10,20) Edit 'DD-MON-YYYY'
description               (,+1,20)
  toc-entry text = &description level=2
c.price * b.quantity      (,+1,13) Edit $$$$,$$0.99
from  orders a, ordlines b, products c
where a.order_num = b.order_num
  and b.product_code = c.product_code
  and a.cust_num = #cust_num
order by b.order_num, b.product_code
end-select
end-procedure ! orders
begin-footing 3
 for-tocs=(cust_toc)
 print 'Company Confidential' (1,1,0) center
print $current-date (1,1) Edit 'DD-MON-YYYY'
end-footing
begin-heading 3
 for-tocs=(cust_toc)
 print 'Table of Contents' (1,1) bold center
 let $page = roman(#page-count)
 print 'Page ' (1,69)
 print $page ()
end-heading
begin-heading 3
 print $current-date (1,1) Edit 'DD-MON-YYYY'
 page-number (1,69) 'Page '
end-heading
```

Figure 16   Two Level Table of Contents HTML File with PRINTER:EH Output

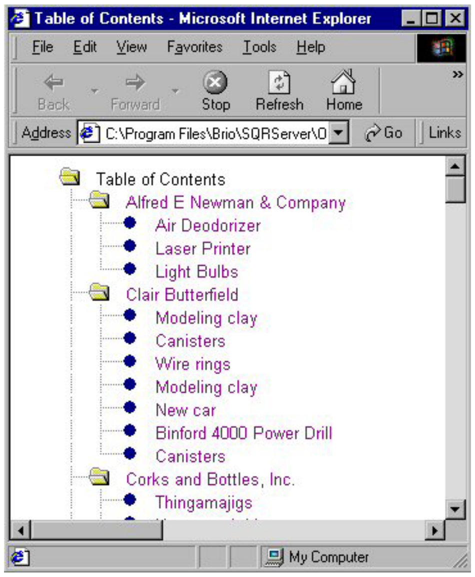


Table of Contents - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address C:\Program Files\Brio\SQRServer\0
Go Links
My Computer

Table of Contents
Alfred E Newman & Company
Air Deodorizer
Laser Printer
Light Bulbs
Clair Butterfield
Modeling clay
Canisters
Wire rings
Modeling clay
New car
Binford 4000 Power Drill
Canisters
Corks and Bottles, Inc.
Thingamajigs

34 Customizing the HTML Navigation Bar

In This Chapter

Prerequisites

Before attempting to write a custom template, read this chapter in its entirety. Also learn the basics of Production Reporting, XML, and HTML. The following are prerequisites for starting Navigation Bar template customizations:

- Familiarity with the Navigation Bar template elements
- Knowledge of Production Reporting, XML, and HTML

About XML

XML stands for **EX**tensible **M**arkup **L**anguage. Use the following components to build an XML document:

- **Elements**—Contain attributes and data. Elements have the format `<element></element>`. Every XML document must have a root element, which contains all other elements. For example, the root element of *template.xml* is `<navbar></navbar>`.
- **Data**—Information specified within elements. Data has the format `<element>data</element>`. Some elements have no data.
- **Attributes**—Add extra information about elements and usually have the format `<element attribute="value">`. Some elements have no attributes.
- **Child elements**—Elements that are nested within other elements. For example, the following child element is nested in a parent element:

```
<parent>
  <child>
  </child>
</parent>
```

  Child elements can also have their own additional child elements.

The Default Navigation Bar Template

Production Reporting provides a template language that allows you to highly customize the layout and functionality of the Navigation Bar in Production Reporting HTML reports. By modifying the `Template.xml` file, you can quickly customize the navigation bar and table of contents frames in Production Reporting reports.

To apply navigation bar changes, modify the `defaultTemplate` setting in the SQR.INI file to include the path to the new template. If the you modify the `Template.xml` file without modifying the SQR.INI `defaultTemplate` setting to point to that file, the template is not applied to your output. See Volume 2 in the *Production Reporting Developer's Guide* for more details on the SQR.INI file.

The default navigation bar template used by the Production Reporting HTML navigation bar is internal to the EHTML driver. The following code example (*template.xml*) is a copy of the default template located in:

```
\Hyperion\products\biplus\docs\samples\Production Reporting
```

The sections following the sample describe each of the XML elements and attributes used in *template.xml*.

For detailed instructions on changing specific attributes, see "Customizing Navigation Bar Attributes" on page 251.

template.xml

```
<?xml version="1.0" encoding="utf-8"?>
<navbar height="25" tocWidth="25%" frames="visible"
        xmlns="http://www.hyperion.com/Navigation Bar Template.xsd"
        xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.hyperion.com/Navigation Bar
Template.xsd">
  <messages>
     <locale language="english">
        <message name="dtoc">Display Table Of Contents</message>
        <message name="dpdf">Display PDF file</message>
        <message name="dps">Show analysis applet</message>
        <message name="dcsv">Download the data in CSV (comma-delimited)
format</message>
        <message name="dbqd">Download the data in BQD format</message>
        <message name="dxml">Display XML file</message>
        <message name="gfp">Go To First Page</message
        <message name="gpp">Go To Previous Page</message>
        <message name="gtp">Go To Page</message>
        <message name="gnp">Go To Next Page</message>
        <message name="glp">Go To Last Page</message>
        <message name="pmsg">Page number must be between 1 and {#}</
message>
        <message name="penb">Page {@} of {#}</message>
        <message name="load">Loading new page</message>
     </locale>
     <locale language="french">
        <message name="dtoc">Afficher la table des matières</message>
        <message name="dpdf">Afficher le fichier PDF</message>
        <message name="dps">Analyse des rapports</message>
        <message name="dcsv">Obtenir les données dans le format CSV</
message>
        <message name="dbqd">Obtenir les données dans le format BQD </
message>
        <message name="dxml">Afficher le fichier XML</message>
        <message name="gfp">Première page</message>
        <message name="gpp">Page précédente</message>
        <message name="gtp">Aller à la page</message>
        <message name="gnp">Page suivante</message>
        <message name="glp">Dernière page</message>
        <message name="pmsg">Le numéro de la page doit être entre 1 et {#}</
message>
        <message name="penb">Page {@} de {#}</message>
        <message name="load">prepare nouvelle page</message>
      </locale>
  </messages>
  <var name="controls">1</var>
  <var name="pageCount">{#}</var>
  <var name="loadMessage" src="load" />
  <var name="pageMessage" src="pmsg" />
  <row>
     <position style="top:0pt; left:20pt;" />
     <position style="top:0pt; left:50pt;" />
     <position style="top:0pt; left:80pt;" />
     <position style="top:0pt; left:110pt;" />
     <position style="top:0pt; left:140pt;" />
     <position style="top:0pt; left:170pt;" />
     <position style="top:0pt; left:200pt;" />
     <icon src="toc.gif" type="JavaScript" height="24" width="22"
message="dtoc">TOC(); return false;</icon>
     <icon src="ps.gif" type="extension"  height="24" width="22"
message="dps">_A0.htm</icon>
     <icon src="ps.gif" type="extension"  height="24" width="22"
message="dps">_G.htm</icon>
     <icon src="pdf.gif" type="extension"  height="24" width="22"
message="dpdf">.pdf</icon>
     <icon src="bqd.gif" type="extension"  height="24" width="22"
message="dbqd">.bqd</icon>
     <icon src="csv.gif" type="extension"  height="24" width="22"
message="dcsv">.csv</icon>
     <icon src="xml.gif" type="extension"  height="24" width="22"
message="dxml">.xml</icon>
  </row>
  <entry src="penb" message="pmsg" type="JavaScript" value="1" size="3"
style="top:0pt; right:150px;" name="go">go(); return false</entry>
     <icon src="first.gif" message="gfp"  type="JavaScript" height="24"
width="22" style="top:0pt; right:138px;">go('1'); return false;</icon>
     <icon src="prev.gif" message="gpp"  type="JavaScript" height="24"
width="22" style="top:0pt; right:116px;">go('-'); return false;</icon>
     <icon src="next.gif"  message="gnp"  type="JavaScript" height="24"
width="22" style="top:0pt; right:94px;">go('+'); return false;</icon>
     <icon src="last.gif"  message="glp"  type="JavaScript" height="24"
width="22" style="top:0pt; right:72px;">go(pageCount); return false;</icon>
</navbar>
```

Navbar Element

The navbar element is the root element of *template.xml*, and it defines basic navigation bar attributes. The navbar element has 6 child elements: `<messages>`, `<var>`, `<row>`, `<entry>`, `<icon>`, and `<image>`. The first element must be `<messages>` then `<var>` followed by `<row>` and `<entry>`. The last two tags, `<icon>` and `<image>`, can be used in any order.

Note:

The <icon> and <entry> elements also can be child elements of <row>.

Table 9 Attributes of the Navbar Element

Attribute	Description
`height=""`	Navigation bar height in pixels or percent. A value of 0 removes the navigation bar. Default = 36 pixels
`tocWidth=""`	Table of contents width as a percent of the total page or in pixels. A value of 0 removes the table of contents. Default = 20%
`frames=""`	*Visible* displays frames. *Invisible* hides frames. Default = *visible*
`xmlns=""`	Name spaces specification. Value must be `"http://wwwhyperioncom/Navigation Bar Templatexsd"`
`xmlns:xsi=""`	XML required name spaces specification. Value must be `"http://wwww3corg/2001/XMLSchema-instance"`
`xsi:schemaLocation=""`	Physical location of schema document. Value must be `"http://wwwhyperioncom/Navigation Bar Templatexsd"`

Messages, Locale, and Message Elements

The `<messages>`, `<locale>`, and `<message>` elements work together to define the language and messages (or “mouse over” text) to be used in the Production Reporting HTML document. As illustrated in *template.xml*, these three elements are part of the XML hierarchy and must be nested in a specific order: `<messages>` is the parent element of `<locale>`, and <locale> is the parent element of <message>. Use these elements in the following order:

```
<messages>
  <locale>
    <message> </message>
  </locale>
</messages>
```

The following sections describe the child elements of `<messages>` in more detail.

Locale Element

The `<locale>` element has one required attribute, `language`, and one child element, `<message>`.

Attribute	Description
`Language=""`	Language used for all mouse over text. Although, you can store many languages in template xml files, you can only use one at a time. In *template.xml*, for example, French and English mouse over text are defined, but only English messages are used.

Message Element

A `<message>` element has one attribute, `name`, and surrounds a string of data. The data must be a string of UTF-8 characters.

Attribute	Description
`name`	Name for a text message For example, the message “Display PDF file” is named “dpdf”, and the message “Show analysis applet” is named “dps” The icon element uses these names to add a message to an icon. The number of pages replaces the sequence {#} in the report. In a ten-page report for example, the message “Page number must be between 1 and {#}” displays “Page number must be between 1 and 10” The sequence {@} is used in conjunction with an `entry` tag. For example, the message “Go to page:{@}” displays a blank area in which you can type a page number.

Var Element

The `<var>` element describe JavaScript variable statements to be used by the navigation bar JavaScript functions. The `<var>` element has one required attribute: `name`. It has one optional attribute, `src`, and optional string data.

Attribute	Description
`name=""`	Required. Defines a name for the variable. The following example equals the number of pages in the report: `<var name="pageCount”>`
`src=""`	Optional. Corresponds to data in the `<message>` element. The following example has a `src` attribute value that corresponds to data in the `<message>` element named “`pmsg`": `<var name="pageMessage" src="pmsg"/>`

Row, Position, Icon, and Entry Elements

The `<row>` element has no attributes but does have the following three child elements:

- `<position>`—Places the icons and entries in a specific order.
- `<icon>`—Imports the navbar icons.
- `<entry>`—Adds JavaScript or HTML widgets to the navbar.

Place the `<position>` elements first since they set up the places (or slots) in which to place the icons and entries. Follow the `<position>` element with `<icon>` and `<entry>` elements in the order they appear in the navigation bar. The first icon or entry in the list displays in the right-most position of the navigation bar, and so on.

The `<row>` element and its child elements allow for flexible positioning of navigation bar items. For example, if a report has no table of contents icon, the next icon in the list moves to the position typically allotted for the table of contents icon, and the empty space becomes occupied.

Note:

In some cases, it is advantageous to have an icon or entry widget in a stationary position, so the `<icon>` and `<entry>` tags can be added outside of the `<row>` element. In this case, they lose the flexibility of being associated with the `<position>` element because the `<position>` element cannot be placed outside of the `<row>` element.

Any `<icon>` or `<entry>` tag placed outside of the `<row>` element must have a definite, unmoving position, which is defined with the `style` attribute as in the following example:

```
<icon src="bqd.gif" type="extension" height="24" width="22"
message="dbqd" style="top:0pt; left:20pt;">.bqd</icon>
```

The following sections describe the child elements of the `<row>` element.

Position Element

The `<position>` element specifies the position of each navigation bar icon and/or entry. It has one attribute, `style`. The `<position>` element must come first followed by any `<icon>` or `<entry>` elements.

Attribute	Description
`style=""`	Required. Position of each navigation bar icon and/or entry. Enter the number of pixels from the top and from the left of the navigation bar. The following example places the icon 0 points from the top and 110 points from the left: `<position style="top:0pt; left:110pt;"/>`

Icon Element

The `<icon>` element imports, positions, sizes, and labels icons in the navigation bar. The `<icon>` element has one required attribute: `src`. It has five optional attributes: `type`, `message`, `style`, `height`, and `width`.

Table 10 Attributes of the Icon Element

Attribute	Description
src=""	URL to an image file supported by browsers through the HTML image tag (GIF, PNG, JPEG) The default value depends on the default icon. See Table 13 for the list of default icons.

type=""	Type of icon. Values include:
	● **extension**—Matches output files in `SQROutputxml` or in the `-eh_csv` type flags. The following example has a BQD extension in the data area:
	`<icon src="bqdgif" type="extension" >bqd</icon>`
	See "Production Reporting Command-line Flags" in Volume 2 of the *Production Reporting Developer's Guide* for more details on `-eh_csv`
	● **JavaScript**—Used only for page and TOC controls. The following example has a JavaScript call in the data area:
	`<icon src="tocgif" type="JavaScript" >TOC(); return false;</icon>`
	● **url**—Opens a new window displaying the URL in the string data The following example has a URLin the data area:
	`<icon src="imagegif" type="url" >http://wwwanotherurlcom/</icon>`
	Values are case sensitive. Default = *extension*

message=""	Message name as defined in the `<message>` element. Default = null.
style=""	Positions the icon except when the icon tag is in a row. If the icon is in a row, use `style=" "` in the `<position>` element as described on "Position Element" on page 248. If the icon is not in a row, enter the number of points from the top, left of the navigation bar. The following example places the icon 0 points from the top and 110 points from the left: `<icon style="top:0pt; left:110pt;"/>` The default is null, which positions the icon in the upper left corner of the navigation bar.
height=""	Non-negative integers that specify the height of the icon image in pixels Default = 1.
width=""	Non-negative integers that specify the width of the icon image in pixels Default = 1.

Entry Element

The `<entry>` element creates entry widgets in the navigation bar. The `<entry>` element has two required attributes, `name` and `src`, and surrounds a string of data. The data must be a string that represents either a JavaScript call or a URL to use when you press the [Enter] key while the cursor is in the entry widget. The `<entry>` element also has five optional attributes: `type`, `value`, `size`, `message`, and `style`.

Table 11 Entry Element Attributes

Attribute	Description
`name=""`	Required. String used to identify the entry widget from JavaScript. Must be unique for all `<entry>` elements.
`src=""`	Required. Names a message to use as the text surrounding the entry widget. The sequence {@} in the message text is replaced with the HTML entry. For example, `<entry name="go" src="myentry">go(); return false</entry>` creates an entry widget on the navigation bar with the words "Go to page:" as a label before a small box in which you type a page number. The message is designated in the `<message>` element. For example, the message “Go to page: {@}” displays when you enter a page number in the “Go to Page” entry widgit

`type=""`	Optional. Must be either "JavaScript" or "url" depending upon the data.
	● **JavaScript**—Used only for page and TOC controls The following example has a Java call in the data area:
	`<entry src="penb" message="pmsg" type="JavaScript" value="1" size="3" style="top:0pt; right:150px;" name="go">go(); return false</entry>`
	● **url**—Makes an entry linked to the url from the string data The following example has a url in the data area:
	`<entry name="search" src="myentry" type="url">http://wwwmysitecom/search</entry>`
	Similar to JavaScript, url creates a box in which you can enter character input and press return For url entry boxes, pressing return submits a form that passes the data as an HTTP GET request For example, if you enter Production Reporting in the entry box and press return, the browser opens the address: `http://wwwmysitecom/search?SQR`
	Default = JavaScript

`value=""`	Optional. String to display in the entry widget as a default value. Default = null.
`size=""`	Optional. Number of characters to display in the entry widget in non-negative integers. Default = 1.
`message=""`	Optional. Name of the message to display when passing the mouse over the widget. Define the mouse over text with the `<message>` element as described on “Message Element” on page 247 Default = null.
`style=""`	Optional. Used for positioning except inside a row. If the entry is in a row, do not include the attribute here; instead use it in the `<position>` element as described on “Position Element” on page 248 If the entry is not in a row, enter the number of points from the top and from the left of the navigation bar to place the entry. The following example places the entry 0 points from the top and 110 points from the left: `<entry style="top:0pt; left:110pt;"/>` The default is null, which positions the entry in the upper left corner of the navigation bar.

Image Element

The `<image>` element imports, positions, sizes, and labels images in the navigation bar. The icon tag has one required attribute: `src`. It has five optional attributes: `type`, `message`, `style`, `height`, and `width`.

Table 12 Image Element Attributes

Attribute	Description
`src=""`	Required. URL to an image file supported by browsers through the HTML image tag (GIF, PNG, JPEG).
`height=""`	Required. Non-negative integers to specify the height of the icon image in pixels. Default = 1
`width=""`	Required. Non-negative integers to specify the width of the icon image in pixels Default = 1
`message=""`	Optional. Message name in the messages section. Default = null
`style=""`	Optional. Number of points from the top, left of the navigation bar. The following example places the image 0 points from the top and 110 points from the left: `<image style="top:0pt; left:110pt;"/>` The default is null, which positions the image in the upper left corner of the navigation bar.

Note:

Since you cannot place an image element within a row, you cannot use the `<position>` element to place images.

Customizing Navigation Bar Attributes

You can customize the look and functionality of your navigation bar by changing the attributes and data of the elements in the *template.xml* file. Review the following topics for information on:

- Working with Templates
- Specifying Navigation Bar Height and TOC Width
- Adding and Placing an Image
- Working with Navbar Icons

Working with Templates

You use templates to create a consistent format for all documents used with the template. You can customize and apply a template or multiple templates to Production Reporting HTML output.

Creating Custom Templates

➤ To customize the *template.xml* file:

1. Save a backup copy of *template.xml*, found in `\Hyperion\products\biplus\docs\samples\Production Reporting`.
2. In a text editor, open the copy of *template.xml*.
3. Make any of the changes described in the following sections.
4. Save your changes.

   You can save many XML templates and use them as needed. See the following section for instructions on applying the template.

Applying an XML Template to Your Output

You can designate one XML template file as the default template for all output.

➤ To define the default template, add the path to the XML template in the `DefaultTemplate` entry in the [Enhanced-HTML] section of the SQR.INI file. (See “[Enhanced-HTML] Section” in Volume 2 of the *Production Reporting Developer's Guide* for more details.)

Specifying Navigation Bar Height and TOC Width

You can specify the height of the navigation bar and the width of the table of contents in the `<navbar>` element of the *template.xml*. In the following example, the navigation bar height is 36 pixels, the table of contents width is 25 percent of the total width of the page, and the frames are visible”

```
<navbar height="36" tocWidth="25%" frames="visible">
```

See “Navbar Element” on page 246 for a detailed description.

Adding and Placing an Image

You can embed an image, such as a company logo, and specify its horizontal and vertical location within the Navbar frame.

In the following example, the file, logo.gif, is 24 pixels high by 250 pixels wide, it is located 0 points from the top of the navigation bar frame and 300 pixels from the right, and it has “Company Logo” as alternate text.

```
<image src="logo.gif" height="24" width="250" style="top:0pt; right:300px"
alt="Company Logo"/>
```

See “Icon Element” on page 248.

Working with Navbar Icons

Production Reporting HTML navigation bar icons are used as links to navigate internally within an Production Reporting HTML document and also as links to open Production Reporting output in multiple file formats. For example, the TOC (table of contents) icon launches the table of contents frame, allowing users to navigate through the HTML document, and the PDF (Portable Document Format) icon launches an external Adobe Acrobat window displaying the Production Reporting report in PDF format.

Review the following section for information on using navigation bar icons:

- Default Icons
- Adding Icons to the Navigation Bar
- Associating Icons with Output Files
- Changing the Default Icon Images
- Changing the Default Order of Icons
- Changing the Default Mouse Over Text
- Associating Icons with File Extensions
- Setting the Navigation Bar Language

Default Icons

`Template.xml` provides default icons that are associated with a specific file type and have mouse over text that describes them.

Note:

Though default icons are associated with file types such as PDF, they are not linked to specific output files. This linking is done with an XML file called `SQROutput.xml`. See "Associating Icons with Output Files" on page 254.

You can use the default icons, replace them, or add additional icons as required. In addition, you can write new mouse over text and associate icons with new file types.

Table 13 describes the default icons and their mouse over text provided in the Navigation Bar template.

Table 13 Default Icons and Messages

Icon	Name	Message	Description
📁	dtoc	Display Table of Contents	Displays the Table of Contents frame
📄	dpdf	Display PDF file	Displays the report in Portable Document Format (PDF) and launches it inside a new browser window

Icon	Name	Message	Description
▦	dcsv	Download the data in CSV (comma-delimited) format	Launches the CSV plug-in or Helper Application in new browser window
▣	dbqd	Download the data in BQD format	Launches the BQD Helper Application in a new browser window
◙	dxml	Display XML file	Launches the XML plug-in or Helper Application in a new browser window
⏮	gfp	Go To First Page	Displays the first page of the report in the current browser window
◁	gpp	Go To Previous Page	Displays the previous page of the report in the browser window
▷	gnp	Go To Next Page	Displays the next page of the report in the current browser window
⏭	glp	Go To Last Page	Displays the last page of the report in the current browser window

Adding Icons to the Navigation Bar

Use the `<icon>` element to add new icons to the navbar. (See "Icon Element" on page 248 for more details.)

Following are the basic steps involved in adding a new icon to the navigation bar. Each step contains a cross reference to a section in this chapter with more detailed information.

1. Select an image and import the file. (See "Changing the Default Icon Images" on page 256.)
2. Place the icon. (See "Changing the Default Order of Icons" on page 256.)
3. Label the icon. (See "Changing the Default Mouse Over Text" on page 257.)
4. Associate the icon with a file extension. (See "Associating Icons with File Extensions" on page 257.)
5. Link the icon with a specific output file. (See "Associating Icons with Output Files" on page 254.)

Tip:

Add only as many navigation bar icons as there are file formats published for your Production Reporting output.

Associating Icons with Output Files

Whether you use the default icons or add custom ones, you must associate each icon with a specific output file. Associating icons with a file enables that file to be opened in a new browser window when you select the icon in Production Reporting HTML.

When you use a navigation bar template, you must also create a file named `SQROutput.xml`. This file contains file names and optional mouse over text. When you run SQR to produce HTML output, SQR uses the information in the `SQROutput.xml` file to associate files with the report. If the `SQROutput.xml` file does not exist, then the `-EH_CSV` type command line flags are used. See "Production Reporting Command-line Flags" in Volume 2 of the *Production Reporting Developer's Guide* for more details.

The `SQROutput.xml` file reads he following icon attributes:

- File name
- File path
- File type
- Tool tip

➤ To associate an icon with an output file:

1 **Use the *template.xml* file to associate the icon with a file extension.**

The file extension is the data of the message element, so you must add it between the angled brackets as follows:

```
<icon>file extension</message>
```

In the following example, PDF is the file extension associated with the icon:

```
<icon src="http://www.mydomain.com/images/myimage.png" height="24"
width="22" message="dpdf">.pdf</icon>
```

2 **Write a file called `SQROutput.xml`.**

`SQROutput.xml` is a manifest file that you must include in your output directory for linking each icon with a specific output file. For each icon you include in your `Template.xml` template, you must write a path in the data portion of the `<file>` element of your `SQROutput.xml` file. For example, `<file>customer.csv</file>` **links the CSV icon to the file called customer.csv.**

When creating your `SQROutput.xml` file, write it exactly as shown in *Sample SQROutput.xml* below; change only the `<file>` elements to match your file output. The file elements appear in bold face.

Each `<file>` element tag in *Sample SQROutput.xml* points to a different file format of the "customer" report. The first element points to the `customer.csv` file, the second element points to the `customer.bqd` file, and the third element points to the `customer.xml` file. These files are located in the same directory as the Production Reporting HTML file. If they were located in another directory, a longer file path would be necessary.

The file element has one attribute, `message`, which is an alternate method of attaching mouse over text to the icon. For example, the xml icon displays *special xml file*.

3 **Save the `SQROutput.xml` file to your output directory.**

Sample SQROutput.xml

```
<?xml version="1.0"?>
<output xmlns="http://www.hyperion.com"
```

```
        xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.hyperion.com
                                SQROutput.xsd">
    <file>customer.csv</file>
    <file>customer.bqd</file>
    <file message="special xml file">customer.xml</file>
</output>
```

Changing the Default Icon Images

Specify images for icons in the Navbar frame by adding a URL to an image file in the `src` attribute for the `<icon>` tag. For example, the following icon uses the default `pdf.gif` file available from the images directory. The `src` attribute is in bold face.

```
<icon src="pdf.gif"type="extension" height="24" width="22"
message="dpdf">.pdf</icon>
```

➤ To change the default images for icons:

1 **Change the `src` attribute in the `<icon>` tag of the *template.xml* file.**

The icon tag in this example uses an icon available on an HTTP server.

```
<icon src="http://www.mydomain.com/images/pdf.png" height="24" width="22"
message="dpdf">.pdf</icon>
```

2 **Change the `height` and `width` attributes in the `<icon>` tag to match the dimensions of the new icon.**

If the dimensions of the icon change, but the height and width attributes do not, the image may look distorted.

3 **Ensure that the file type associated with the new icon is correct.**

If the icon represents a PDF file, then you must type the PDF file extension in the data area of the icon element. For example:

```
<icon src="http://www.mydomain.com/images/pdf.png" height="24" width="22"
message="dpdf">.pdf</icon>
```

4 **Save your changes.**

To open a file using an icon, associate the icon with a file extension. See "Associating Icons with File Extensions" on page 257 for the next step of this process.

Changing the Default Order of Icons

You can specify horizontal placement of icons and the order in which to display the icons in your navigation bar. If the icons are within the `<row>` element, use the `<position>` element to place the icons.

To specify the placement of icons, use the `<position>` element, which is nested in the `<row>` element. In the following example, the PDF icon appears to the left of the BQD icon:

```
<row>
  <position style="top:0pt; left:20pt;"/>
  <position style="top:0pt; left:50pt;"/>
  <icon src="pdf.gif" type="extension" height="24"
width="22"     message="dpdf">.pdf</icon>
  <icon src="bqd.gif" type="extension" height="24"
width="22"     message="dbqd">.bqd</icon>
</row>
```

If the icons are not nested in the <row> element, you can use the `style` attribute of the `<icon>` element to place the icon, as in the following example.

```
<icon src="bqd.gif" type="extension" height="24"
width="22"     message="dbqd" style="top:0pt; left:20pt;">.bqd</icon>
```

Changing the Default Mouse Over Text

You can create and edit mouse over text for default and new icons in the navigation bar. Mouse over text can be used as "tool tips" that display when you pass a mouse over an icon and as alternate text for users who use HTML screen readers.

Define mouse over text with the `name` attribute of the `<message>` element, and attach mouse over text to an icon with the `message` attribute of the `<icon>` element.

The following example defines mouse over text in two parts:

1. The `name` attribute of the `<message>` tag defines the message text as "Display PDF file" and gives it the name of "dpdf."

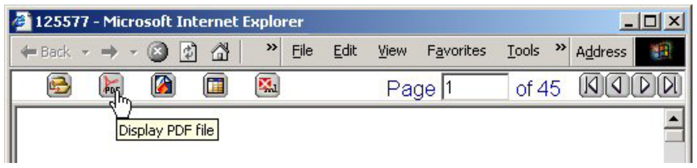
   ```
   <message name="dpdf">Display PDF file</message>
   ```

2. The message named '"dpdf" is attached to the icon with the `message` attribute of the `<icon>` element.

   `<icon src="pdf.gif" type="extension" height="24" width="22"`
   message="dpdf"`>.pdf</icon>`

As shown in Figure 17, the message, "Display PDF" displays when you pass the mouse over the icon.

Figure 17 Display PDF Icon Message



See "Setting the Navigation Bar Language" on page 258.

Associating Icons with File Extensions

You can associate icons with additional file extension types for Production Reporting and non-Production Reporting generated files. In addition Production Reporting HTML supports multiple icons of a specific file type. For example, you could include multiple BQD and DOC file icons in the Navbar of one Oracle's Hyperion® SQR® Production Reporting HTML report. Example extension include:

- .PDF
- .BQD
- .CSV
- .XML
- .HTM
- .DOC
- .XLS
- .PPT

➤ To associate icons according to known file extension types for Production Reporting and non-Production Reporting generated files:

1 **Select the <icon> element of an icon from the default icons in the Navbar template or write an <icon> element for a custom icon.**

  See "Icon Element" on page 248 for details on the attributes and data of the icon element. See "Changing the Default Icon Images" on page 256 for information on adding custom icons.

2 **Write the file extension in the data area of the <icon> element.**

  In the following example the PDF file extension is designated in the data area of the icon element:

```
<icon src="pdf.gif" type="extension" height="24" width="22"
message="dpdf">.pdf</icon>
```

  See "Associating Icons with Output Files" on page 254 for instructions on linking the icons with specific files.

Tip:

To identify each file when multiple files of one file type exist, use mouse over text to describe each icon. See "Changing the Default Mouse Over Text" on page 257.

Setting the Navigation Bar Language

You can set the language used for the HTML navigation bar by using the command line flag: –`eh_language:`*`language`*.

Note:

This flag is only applicable when -PRINTER:EH or -PRINTER:EP is specified.

You can choose from the following languages provided in the default template:

- English
- French
- German

- Portuguese
- Spanish
- Simplified Chinese
- Traditional Chinese
- Japanese
- Korean

Each of the languages in the template has its own set of messages translated for that language. See Table 13 for a list of default messages in English. All languages have an equal number of messages, and all corresponding messages share the name. For example, the “Display TOC” message is named “dtoc” regardless of the locale.

Index