JAM 7

JAM/ReportWriter Developers Guide

April 1996

This software manual is documentation for JAM/ReportWriter® 7. It is as accurate as possible at this time; however, both this manual and JAM/ReportWriter itself are subject to revision.

Send suggestions and comments regarding this document to:
Technical Publications Manager
JYACC, Inc.
116 John Street
New York, NY 10038
(212) 267–7722

Table of Contents

About this Guide

JAM/ReportWriter is an add-on to JAM that lets you define and produce reports that run with a JAM application or independently. This manual is aimed at first-time users and more experienced users who have specific questions. It assumes that you are already familiar with JAM and especially the JAM editing environment.

If you are new to JAM, you should first refer to the JAM documentation set. In particular, you can ground yourself in the fundamentals of JAM development by reading this components:

- *Getting Started* contains a tutorial that guides you step-by-step through creation of a JAM application. Refer to page 21.
- Chapter 1 in the *Application Development Guide*, "JAM Development Overview," describes JAM components and the development process.
- *Editors Guide* describes the screen editor components and the screen and widget properties.

If you are upgrading from earlier versions of ReportWriter, refer to Appendix A for information on running the upgrade utility `rw6to7` and the new features in ReportWriter 7.

Conventions

The following typographical and terminological conventions are used in this guide:

`expression` Monospace (fixed-spaced) text is used to indicate:

- Code examples.
- Words you’re instructed to type exactly as indicated.
- Filenames, directories, library functions, and utilities.
- Error and status messages.

`KEYWORDS` Uppercase, fixed-space font is used to indicate:

- SQL keywords.
- Mnemonics or constants as they appear in JAM include files.

numeric-value Italicized helvetica is used to indicate placeholders for information you supply.

[option-list] Items inside square brackets are optional.

`{x | y}` One of the items listed inside curly brackets needs to be selected.

x ... Ellipses indicate that you can specify one or more items, or that an element can be repeated.

new terms Italicized text is used to indicate defined terms when used for the first time.

JAM Documentation

The JAM documentation set includes the following guides and reference material:

Read Me First — Consists of three sections:

- *What’s New in JAM* — Briefly describes what’s new in JAM 7.
- *Installation Guide* — Describes how to install JAM on your specific platform and environment.
- *License Manager Installation* — Instructions for installing the License Manager (used on many UNIX and VMS platforms).

Getting Started — Contains useful information for orienting you to JAM. Includes a description of the JAM environment and features, how JAM addresses real-world application development issues, and a guided tutorial for building a mini-JAM database application.

Editors Guide — Instructions about using the JAM authoring environment; learn how to use the graphical tools for creating, editing, and designing your application interface. Includes detailed descriptions of the screen editor, screen wizard, menu bar editor, and styles editor. The *Editors Guide* is also provided online on GUI platforms. It is installed with the installation of the JAM software and can be accessed by selecting help from within the screen editor.

Application Development Guide — Information by topic to guide you in developing your JAM application. This includes components of the JAM development environment such as the repository, hook functions, and menu bars, as well as sections on the SQL executor, SQL generator and the transaction manager.

Language Reference — Describes JPL, JAM's proprietary programming language. Also includes reference sections for JPL commands, built-in functions and JAM library functions. The man pages in the reference sections are arranged alphabetically.

Database Guide — Instructions for using JDB, JYACC's prototyping database, and for the commands and variables available in the database interfaces. Includes an Database Drivers section containing instructions unique to each database driver.

Configuration Guide — Instructions for configuring JAM on various platforms and to your preferences. Some options that can be set relate to messages, colors, keys and input/output. Also includes information on GUI resource and initialization files.

Master Index and *Glossary* — Provides an index into the entire documentation set and a dictionary of terms used in the documentation set. This is in addition to the indexes in the individual volumes.

Upgrade Guide — Online only. Information for upgrading from JAM 5.

Online Documentation

JAM's documentation set is available online and included with the JAM distribution. The books can be viewed through the DynaText^{TM} browser on GUI platforms. It can be accessed by choosing Help from within JAM or by running DynaText's read-only browser from the command line or by clicking on the DynaText icon. For instructions on using DynaText, request Help while you have a browser window open.

Collateral Documentation

The following information is also provided with your JAM installation:

- Database Driver Notes — JAM 7 has database drivers for most popular relational database engines, as well as JDB, JAM's proprietary database. Information for JDB, Sybase, Oracle, Informix and ODBC are located in the *Database Guide*; others are included separately.
- Online help — This guide and the JAM *Editors Guide* is provided in online form through the DynaText browser on GUI platforms. It can be accessed by choosing Help from the screen editor. For instructions on using DynaText, request Help while you have a browser window open.
- Online README file.

Additional Help

JYACC provides the following product support services; contact JYACC for more information.

- Technical Support
- Consulting Services
- Educational Services

1

Overview of ReportWriter

JAM/ReportWriter is a report authoring tool that is completely integrated into JAM's authoring and runtime environment. This chapter offers an overview of the editing tools that you use to create reports. It also describes creation of a simple report. The last sections briefly describe other important ReportWriter features.

Authoring Environment

ReportWriter uses JAM's screen editor to create reports. You can move from screens to reports during the same editing session, using the same tools to create and test an entire application. For detailed information about the screen editor, refer to the *Editors Guide*.

The report authoring environment is integrated within the screen editor to include access to these JAM development features:

- Wizard-generated reports — Use the report wizard to create the report types most often in demand: column, record-by-record, row, matrix, and reports that output address labels. The wizard creates finished-looking reports that you can use straight away, or that you can easily edit for further enhancements.

- Repositories — Because ReportWriter and JAM share the same repositories, it's easy to maintain consistency among reports and applications.
- Multiple font support — Report output can be set to any printing font that your platform supports, including PostScript and TrueType fonts. Combine different fonts and font sizes along with italics, bold and underline on a widget-by-widget basis. The graphical editor lets you immediately evaluate font selections as you make them.
- Graphs and graphics — Display report output as bar and pie graphs, or in any of the other graph formats that JAM supports. You can also include scanned input such as photos, or computer-generated drawings.

Editor Extensions

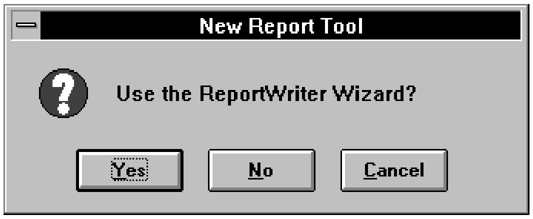
JAM's screen editor is customized and extended for ReportWriter:

- The File⇒New and File⇒Open menus include a Report option:
  - New⇒Report invokes a message asking whether to use the report wizard to create a report.
  - Open⇒Report invokes a dialog that initially shows files with the default report file extension `*.jrw`.
- The Properties window has two categories that are unique to ReportWriter:
  - Inclusions contains report file properties—JPL Procedures and Report Files.
  - Composition contains properties that are unique to report widgets and nodes.
- Create menu options and equivalent tool box icons vary, depending whether a report or screen has focus. The Create menu for reports contains a subset of JAM widget types, such as graph and line widgets. The menu also lists several widget types that are specific to reports.
- The View menu contains a Report Structure option that opens the report structure window. The report structure schematically depicts specifications for report generation.
- The Options menu contains a Drag Screen Size option. When this option is enabled, resizing a screen conforms to previous JAM behavior. When it is disabled, you can resize a report layout viewport and leave the report's Geometry properties unchanged.

- The menu bar contains a Report menu for report-specific options such as Page Setup.

Creating a Report

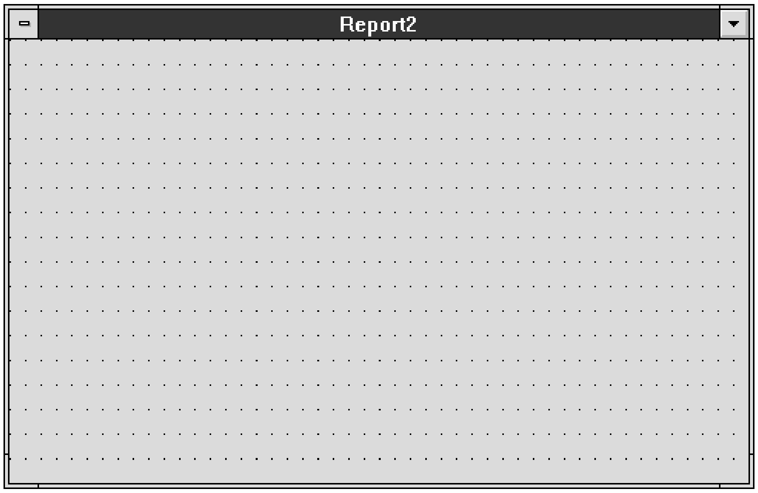
You create a report by selecting from the File menu New⇒Report. ReportWriter asks whether you want to use the report wizard:



New Report Tool

Use the ReportWriter Wizard?

For information about using the report wizard, refer to the next chapter (page 21). To build a report manually, choose No. When you make this choice, ReportWriter opens an unnamed report file and displays its layout window:



The layout window is one of two editing windows that you use to build a report. The second window shows the report structure and is opened by choosing View⇒Report Structure. Both windows offer complementary views into the report definition:

Layout window
Defines the format and content of report output. It typically contains one or more *layout areas*, named spaces whose contents are output at runtime. A layout area is

invoked through a print node in the report structure. The timing and sequence of the layout area's output depends on execution of its print node.

Report structure window
Schematically shows how a report executes. Stages of report execution are depicted in a hierarchy composed of *nodes*. The topmost node defines a report; when this report runs, all nodes below and subordinate to it execute in top-down order.

Laying Out the Report

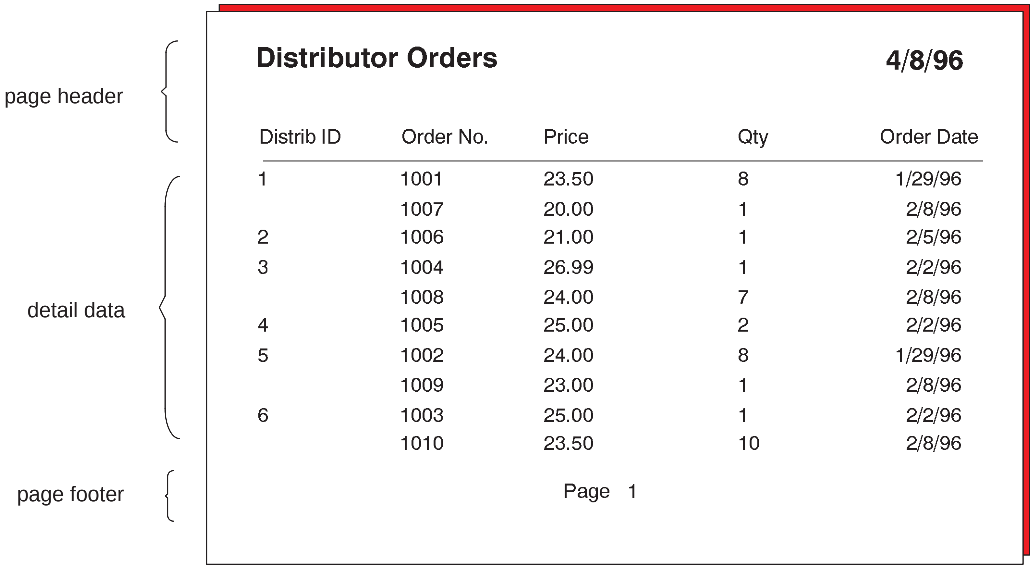
The layout window typically contains several layout areas that define different components of a report's output—for example, detail data, column headings, page header and footer, and group subtotals. A layout area outputs data through the widgets that populate it. Each widget's properties determine the source and format of its data; the widget's position within the layout area determines where that output appears in relation to other output from the same layout area.

output widget types

Most report output is generated through two widget types:

- *dynamic output* widgets get their data at runtime—for example, from a database or from other widgets.
- *static output* widgets have their data set in the editor; their data remains constant.

For example, the following report outputs order information and contains three elements—a page header, detail output, and page footer:


page header
detail data
page footer

Distributor Orders **4/8/96**

Distrib ID	Order No.	Price	Qty	Order Date
1	1001	23.50	8	1/29/96
	1007	20.00	1	2/8/96
2	1006	21.00	1	2/5/96
3	1004	26.99	1	2/2/96
	1008	24.00	7	2/8/96
4	1005	25.00	2	2/2/96
5	1002	24.00	8	1/29/96
	1009	23.00	1	2/8/96
6	1003	25.00	1	2/2/96
	1010	23.50	10	2/8/96

Page 1

Figure 1. Report whose output is composed of page header and footer data, and detail data.

Output for each of these elements is defined in its own layout area. The following sections show how to create each of these layout areas.

Note: *To reproduce this sample report, open the* `vbizplus.dic` *repository, which is part of the JAM/ReportWriter distribution.*

Detail Layout

Detail data is typically derived from a database; the data is output through dynamic output widgets whose database properties specify the source of their output. You can easily populate a detail layout area by copying widgets from an open repository whose entries were imported from a database; for more information about importing database tables into a repository, refer to page 59 in the *Editors Guide*.

This report's detail section is output through a layout area whose widgets are copied from the `vbizplus.dic` repository. Creating this layout area is a two-step process:

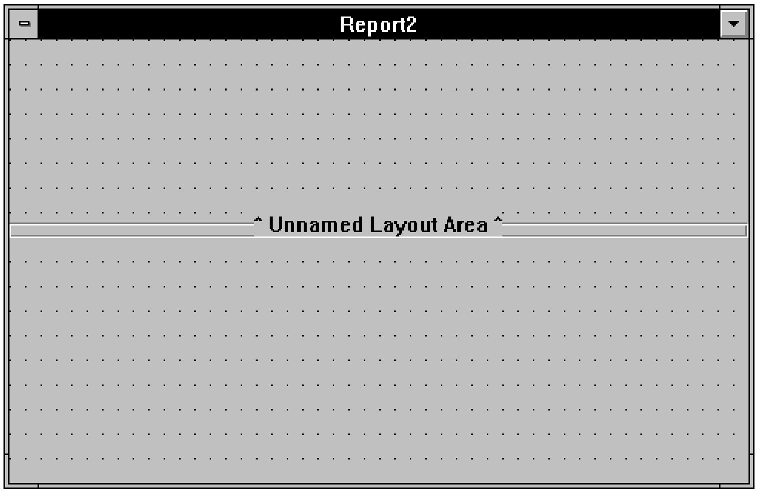
- Define the layout area.
- Populate the layout area by copying widgets from the open repository.

Define the detail's layout area:

1. Give the layout window focus.
2. Choose Create⇒Layout Area or the corresponding toolbox button.

3. Point and click where you want the new area to appear in the layout window.

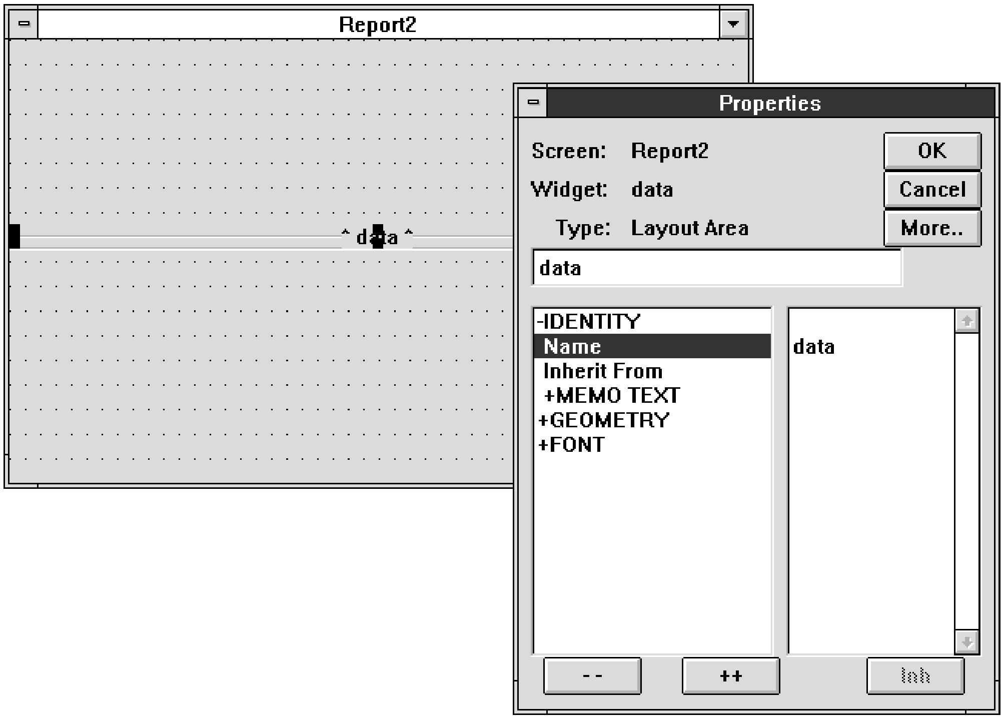
   ReportWriter creates an unnamed layout area widget, which defines the area above it:



Note: *You can locate a layout area anywhere within the layout window without regard to its actual output when the report is generated.*

4. Name the layout area:

   - Select the layout area widget.
   - Give focus to the Properties window.
   - Under Identity, set the Name property to `data`.



Screen: Report2

Widget: data

Type: Layout Area

data

Copy repository widgets into the layout area:

1. Open the `vbizplus.dic` repository.

2. Open two repository screens: `orders` and `order_items`.

3. Copy these widgets into the `data` layout area (either drag and drop or copy and paste):

   - From `orders`:

     `distrib_id`, `order_num`, `order_date`

   - From `order_items`:

     `price`, `qty`, and link widget `K1order_items`

   ReportWriter automatically converts all copied widgets (except the link widget) into dynamic output widgets.

4. Move the link widget below the layout area widget, near the bottom of the layout window. Widgets that are outside a defined layout area cannot output any data.

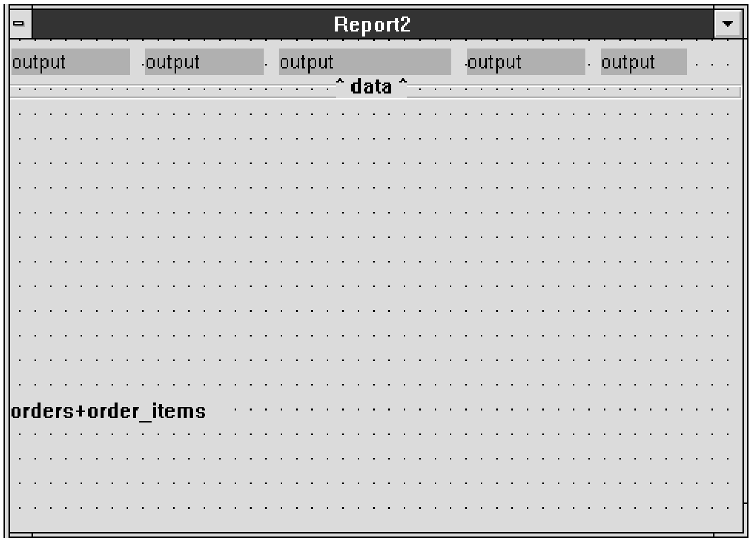
5. Rearrange and realign the other copied widgets on the same row—from the Edit menu, use Align⇒Bottom and Space⇒Horizontal.

6. If desired, edit the widgets' properties to change their format, size, and font. For example, you can set `order_date`'s Format Type property to `DEFAULT DATE`.

7. If necessary, close up unnecessary white space within the layout area:

   - Select and drag the output widgets closer to the top of the layout area.
   - Drag the layout area widget closer to the output widgets.

The layout window should look like this:



Page Header Layout

The page header output shown earlier contains several static output widgets that contain column headings, a static output widget that contains the report title, and a dynamic output widget that contains the system date.

Creating this layout area is two-step process:

- Define a layout area.
- Populate the layout area by creating widgets.

Define the page header's layout area:

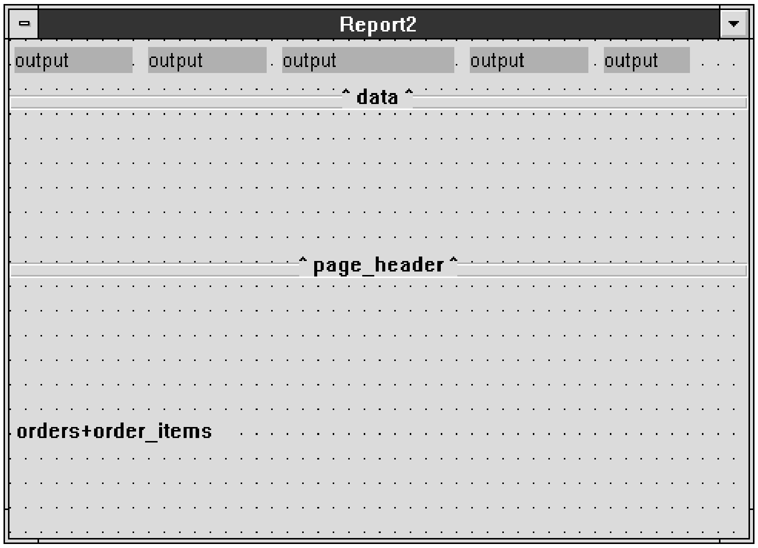
1. Give the layout window focus.

2. Choose Create⇒Layout Area or the corresponding toolbox button.

3. Point and click below the `data` layout area.

4. Name the new layout area `page_header`.

5. If necessary, resize the layout area by dragging the layout area widget up or down.



Create the column headings and report title:

1. Give the layout window focus.

2. Choose Create⇒Static Output or the corresponding toolbox button.

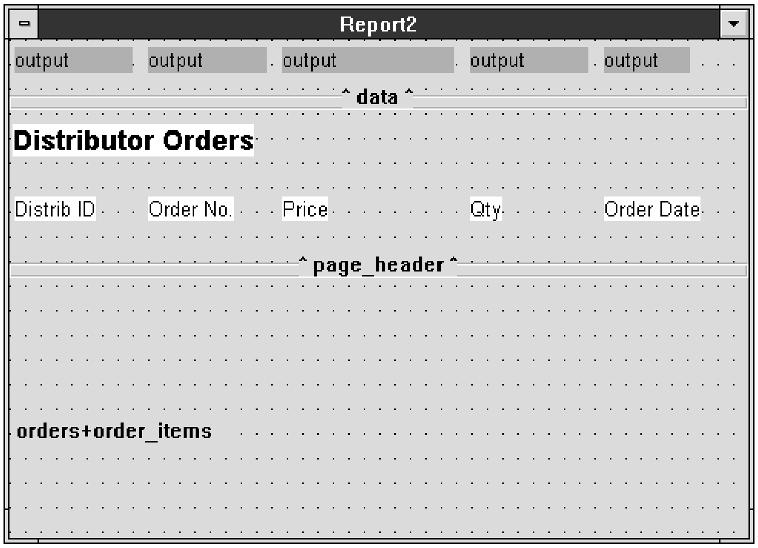
3. Create the widget by pointing and clicking in the `page_header` area.

4. Edit the widget's Label property—for example, `Distrib ID` for the widget above `distrib_id`'s output.

5. Repeat steps 2–5 for each column heading and the report's title.

6. Position the widgets as necessary.

7. If desired, edit the widgets' properties to change their format, size, and font. For example, the widget that contains the report's title has its Bold property set to Yes and its Font Size property set to 14.



Create a date widget:

1. Choose Create⇒Dynamic Output or the corresponding toolbox button.

2. Create the widget by pointing and clicking in the `page_header` area.

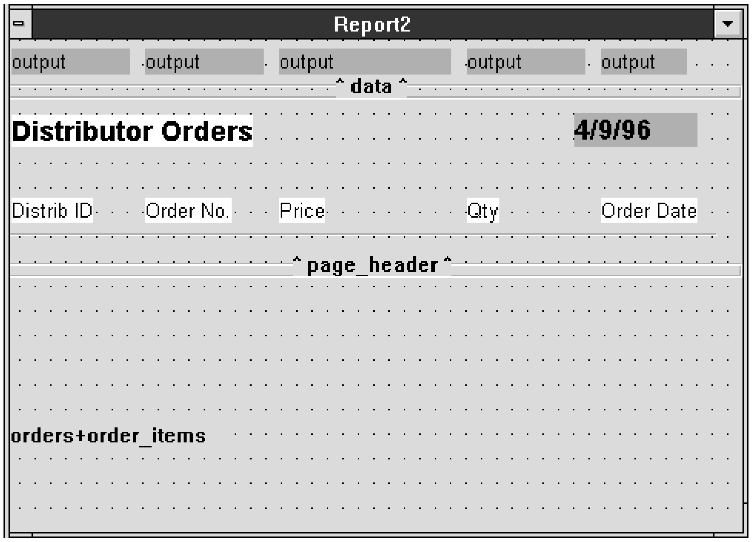
3. Set these properties under Format/Display:

   Date Formatting = `Date/Time`
   Format Type = `DEFAULT DATE`

   If desired, edit other properties. For example, you might want this widget to have the same font size and attributes as the report title.

4. Align the widget and the report title on the same row.

Finally, create a line widget below the column headings. The layout window now contains two layout areas that look like this:



Page Footer Layout

The page footer output shown earlier contains two widgets: a static output widget whose Label property is set to `Page`; and a widget that outputs the current page number.

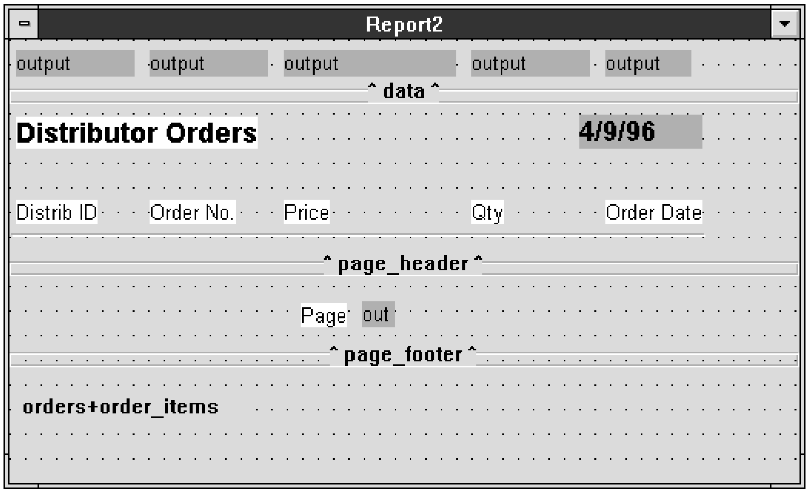
To define this output:

- Create a layout area below `page_header` and set the name of the layout area widget to `page_footer`.
- Create a static label and set its Label property to `Page`.
- Create a page number widget.

Create a page number widget:

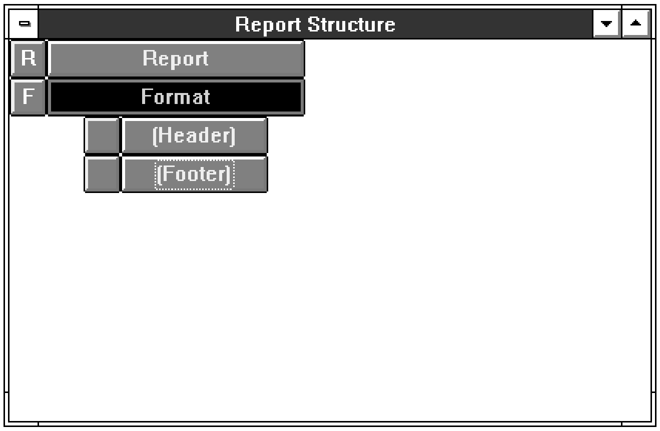
1. Choose Create⇒Dynamic Output or the corresponding toolbox button.
2. Create the widget by pointing and clicking in the `page_footer` area.
3. Under Composition, set the widget’s Value property to Page Number.
4. Close up the space between this widget and the static label and align them horizontally.

The finished layout window should look like this:



Defining the Report Structure

While layout areas define the physical layouts used by a report, specifications for report generation are defined in the report structure. To view a report's structure window, choose View⇒Report Structure. A new report has this empty structure:



The initial structure begins with an unnamed report node. This node and the nodes connected to it define the report. Through their properties, each one defines an aspect of the report or an action performed during execution. For more about node types, refer to page 62.

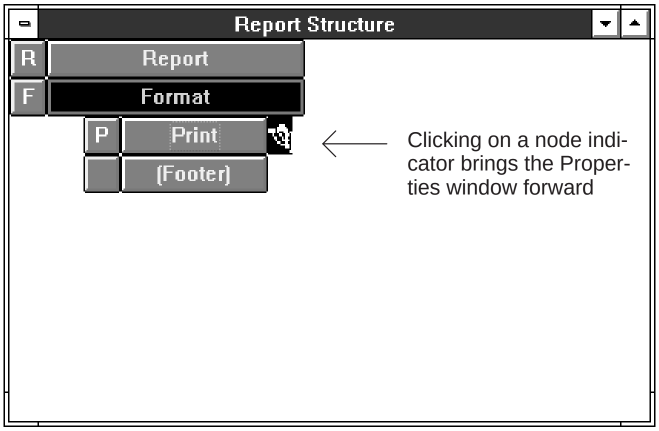
To generate the report shown in Figure 1, you need to modify this structure to include these nodes:

- Print nodes that output the page header and footer data.
- A detail node that specifies how the report gets its data.
- A print node that outputs detail data.
- A group node that specifies to group data for each distributor ID.

Print page header and footer data:

1. Give focus to the report structure window.
2. Choose Create⇒Print or the corresponding toolbox button.
3. Click on the empty page header node (labeled Header).

   ReportWriter creates a print node directly below the page format node, which replaces the empty page header:

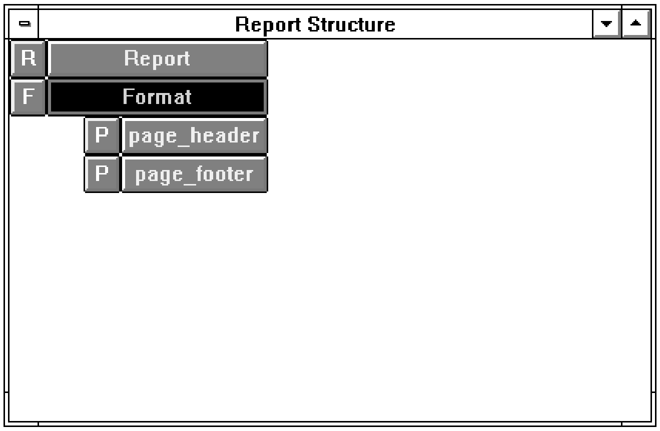


Clicking on a node indicator brings the Properties window forward

4. Bring the Properties window forward by double-clicking on the indicator to the right of the new print node.
5. Under Identity, set the print node’s Area property to `page_header`: either select this layout area from the drop-down list, or enter the name directly.
6. Choose Create⇒Print or the corresponding toolbox button.
7. Click on the empty footer node.

   ReportWriter replaces the empty footer node with a print node.
8. Set this print node’s Area property to `page_footer`.

The report structure now contains print nodes for page header and footer output:

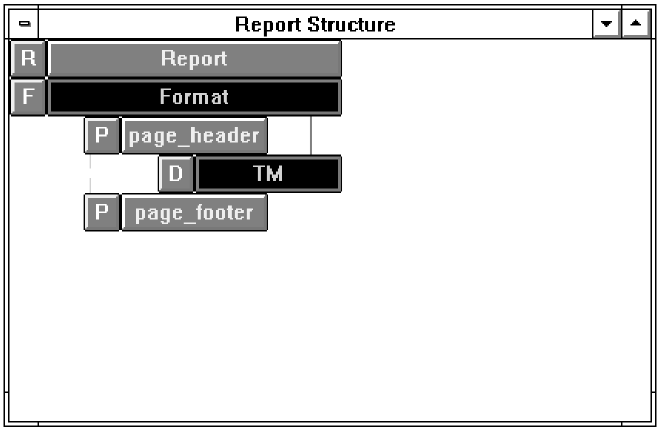


Fetch data from a database:

A report relies on the structure’s detail node to fetch data. Detail node properties specify how the report fetches its data; the print node attached to a detail node specifies how this data is presented.

1. Choose Create⇒Detail or the corresponding toolbox button.

2. Click on the page format node. ReportWriter inserts a detail node beneath the page format node:



The detail node’s Data Source property (under Identity) is initially set to TM View. At runtime, this setting specifies that JAM’s transaction manager generates the SQL needed to fetch the report data. A report created by the report wizard or built from repository components like this one contains the table views and link widgets required by the transaction manager.

To make sure that the fetched data is sorted appropriately, set the root table view's Sort Widgets property:

3. Open the Widgets list window (View⇒Widget List) and select the `orders` table view widget.

4. Under Database, set the Sort Widgets property to these widget names:

```
distrib_id
order_num
order_date
```

Print detail data:

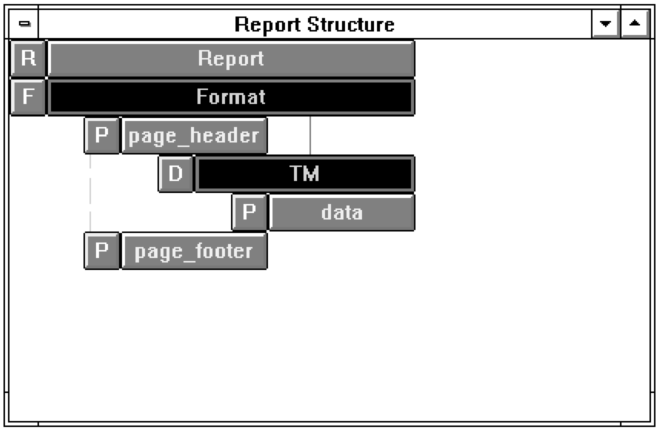
1. Choose Create⇒Print or the corresponding toolbox button.

2. Click on the detail node.

   ReportWriter creates a print node directly below the detail node.

3. Set this print node's Area property to `data`.

The report structure now contains a print node for outputting its detail data:



Group report data:

1. Choose Create⇒Group or the corresponding toolbox button.

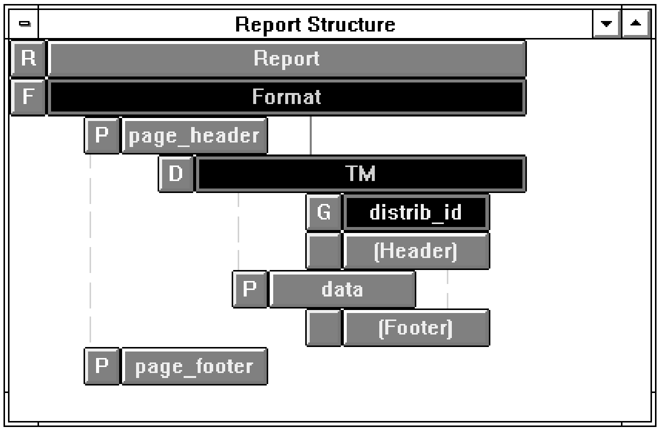
2. Click on the detail node.

   ReportWriter creates a group node directly below the detail node.

3. Under Identity, set this group node's Break On property to `distrib_id`. This property specifies how to group data at runtime—in this case, to generate a group for each `distrib_id` value.

4. Under Composition, set the Print Break Value property to First Use Only. This setting ensures that only the first instance of each `distrib_id` value is output.

The report structure now contains a group node for grouping report data on `distrib_id`:



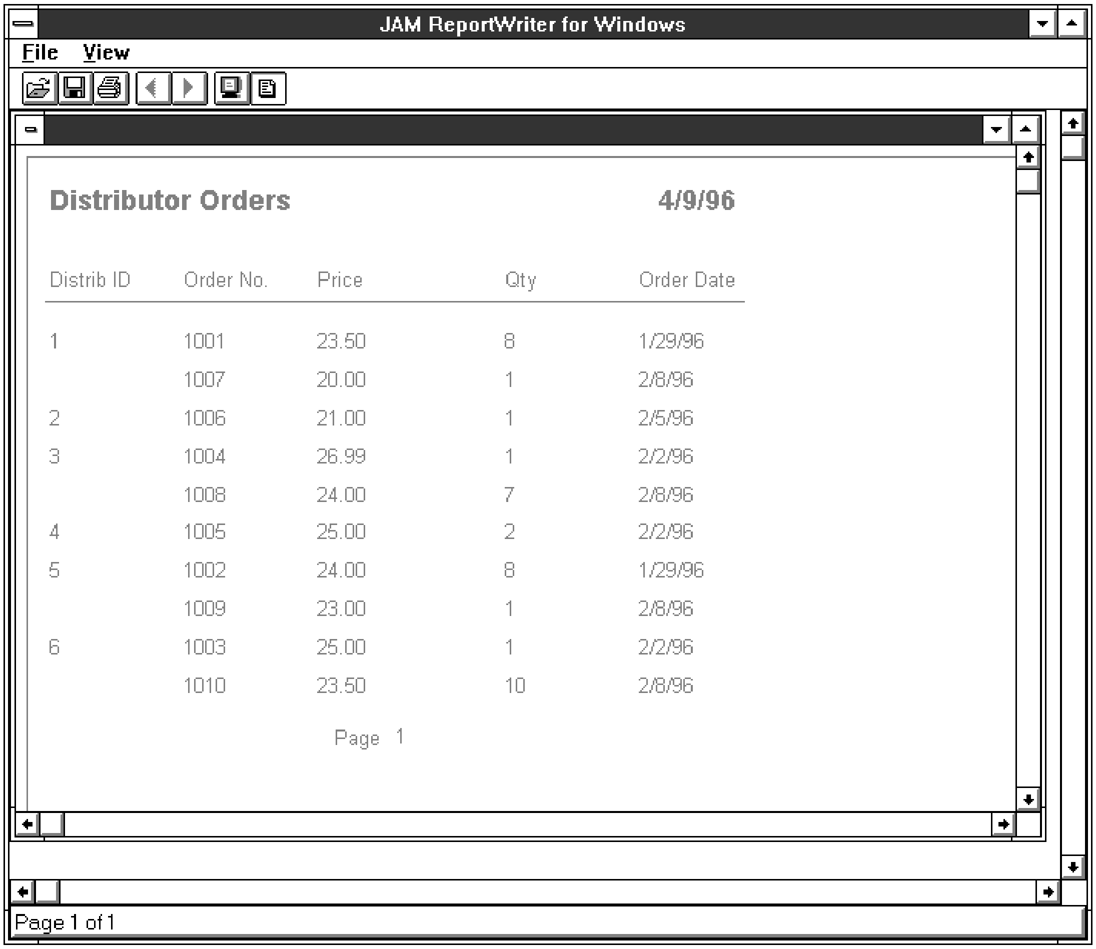
Testing the Report

The editor has its own viewer, which you can use to view report output at any stage of the editing process. To test the report built in previous sections:

1. Open the `vbizplus` database.

2. Choose File⇒Test Mode or the corresponding toolbar button.

The viewer outputs the report, scaled initially to fit in the viewer window. To view the report scaled to 100 percent, choose View⇒Actual Size or the corresponding toolbar button:



Distributor Orders 4/9/96

Distrib ID	Order No.	Price	Qty	Order Date
1	1001	23.50	8	1/29/96
	1007	20.00	1	2/8/96
2	1006	21.00	1	2/5/96
3	1004	26.99	1	2/2/96
	1008	24.00	7	2/8/96
4	1005	25.00	2	2/2/96
5	1002	24.00	8	1/29/96
	1009	23.00	1	2/8/96
6	1003	25.00	1	2/2/96
	1010	23.50	10	2/8/96

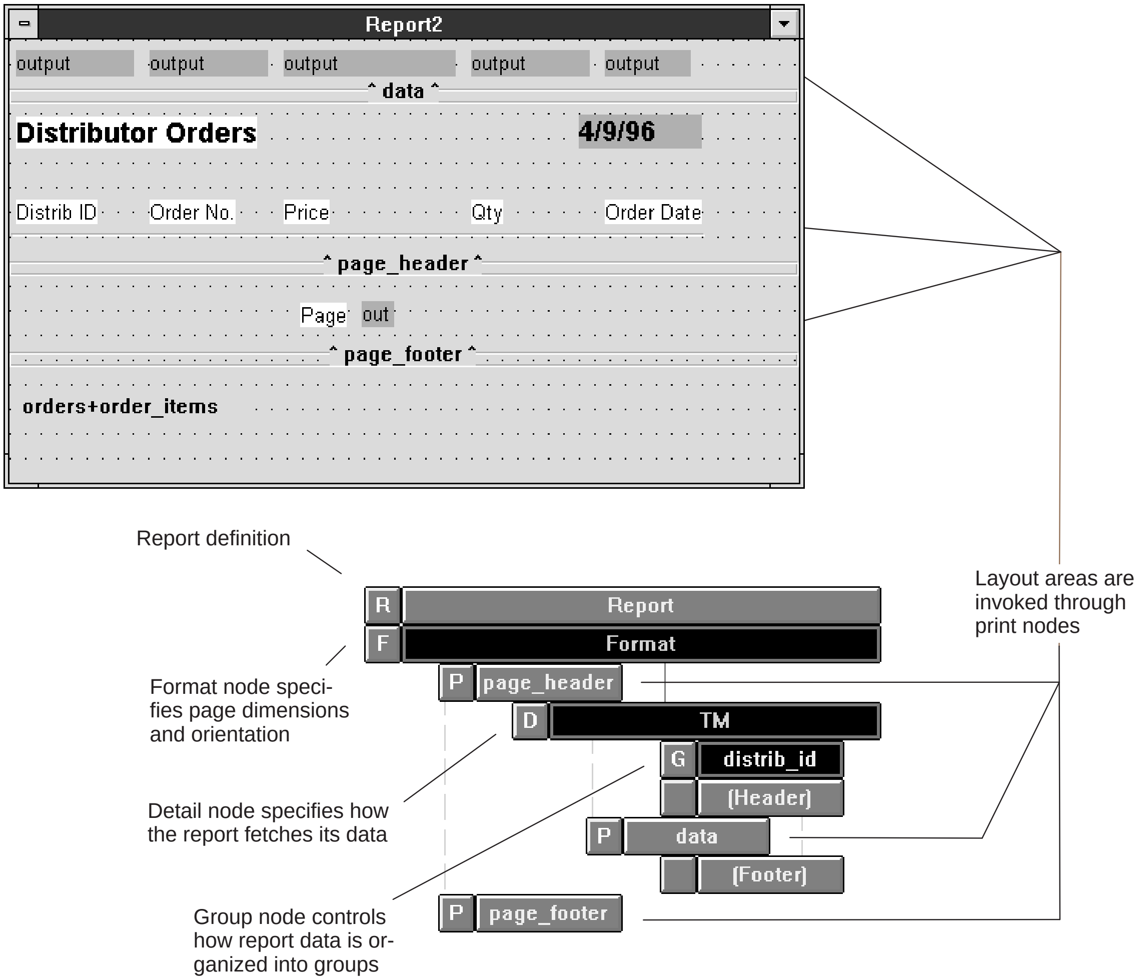
Page 1

Figure 2. The report viewer lets you preview and evaluate report output during an editing session.

Report Definition: A Composite View

A report’s structure and layouts comprise the essentials of a report definition. The report defined earlier looks like this:

Report Definition: A Composite View



Report definition

Layout areas are invoked through print nodes

Format node specifies page dimensions and orientation

Detail node specifies how the report fetches its data

Group node controls how report data is organized into groups

ReportWriter’s unique method of defining a report, which separates layout from execution, gives you complete control over the format and sequence of output at all stages of execution. For example, you can call a function at any stage of report execution by inserting a call node at the corresponding point in the report structure. This function might change at runtime the layout area that a print node invokes; or conditionally skip over a portion of the report structure.

You can make reports as simple or complex as you like. Most reports, like the one shown earlier, require a basic structure with only a few nodes. The real power of ReportWriter lies in the properties that you can set on individual nodes and widgets. For example, you can prevent page breaks within groups by setting the appropriate group node’s Keep on Page property to Yes. Or specify a widget to output subtotals simply by setting its Value and Value Source properties.

Automatic Calculations

At runtime, ReportWriter can automatically capture data as it is output and use it to calculate totals. It can also copy output to other locations in the report; or use an array to collect a series of output values.

You can also write your own functions to process output—for example, to calculate summary data such as averages, maximums, and minimums, or to control report execution by changing node properties at runtime.

Modular Architecture

A report file can contain one or more report definitions. These reports are accessible to each other and to reports in other files. So, you can build a report that is entirely defined in a single file; or one whose components are partially or entirely derived from other report files. This is particularly useful for objects that have the potential for being reused by different reports—for example, layouts for title and trailing pages, or inserted reports.

A report can access the contents of another report file in two ways:

- A subreport node can invoke a report that is defined in another file by specifying the file name and the name of the report.
- A report can specify report files to be included through its Report Files property. All report definitions, layout areas, and widgets in the included files are accessible to the entire report.

Cross-Platform Portability

You can create and run reports on any platform supported by JAM, including Windows, Macintosh, Motif, and on various character-mode platforms. Reports that are created on one platform can be run on another without editing; JAM handles all cross-platform issues such as font mapping to ensure equivalent output on all platforms.

Automatic SQL Generation

Reports that get their output from a database can take advantage of JAM’s transaction manager to generate the appropriate SQL. Wizard-created reports rely entirely on the transaction manager to fetch their data. The report wizard automatically creates the necessary widgets, links, and table views. Alternatively, you can write your own SQL statements. In either case, you can rely on JAM’s database drivers to interpret the SQL correctly and fetch the data you want.

Running Reports

While you are developing your report, you can preview its output at any time. You can either view reports at your terminal through the editor’s viewer, or you can send output to a file or printer. In graphical environments such as Windows, the viewer accurately shows the output you can expect from printed output. Because the viewer is integrated within the editor, you can respond immediately to the viewer output, then run the report to the viewer again to see how your changes look.

You can also use JAM’s debugger when you run reports in the editor—for example, to display the current value of an output widget.

When the report is complete, you can run it either from the command line or a JAM application. You can also set up a report so that users can influence report output.

ReportWriter supports these options for directing output:

- PostScript — The report is generated in Adobe standard PostScript.
- Macintosh/Windows — The report is generated using the printer driver for any installed printer. This is the default output driver when running Macintosh or Windows.
- Text — The report is generated in ASCII text.
- Metafile — The report is generated in ReportWriter’s metafile format for display in the report viewer.

2

Creating Reports with the Report Wizard

The report wizard helps you design a JAM report that uses JAM’s transaction manager to fetch data. It collects design information through a series of easy-to-use dialogs. Though simple to use, the report wizard yields sophisticated reports that you can use unchanged, or easily edit for added functionality and refinements.

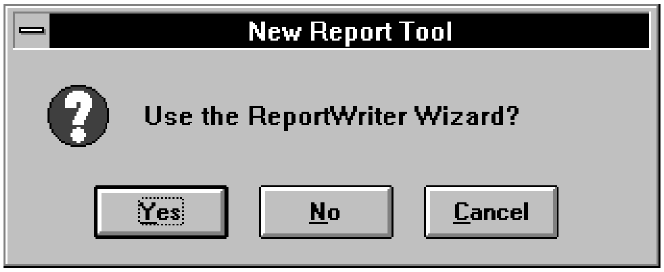
Invoking the Report Wizard

To run the report wizard:

1. Choose File⇒Open⇒Repository. The browser that opens lets you choose a repository. This repository must contain entries for the database tables, views and/or synonyms that have been imported from the database.

   For information on creating repositories, refer to page 55 in the *Editors Guide*.

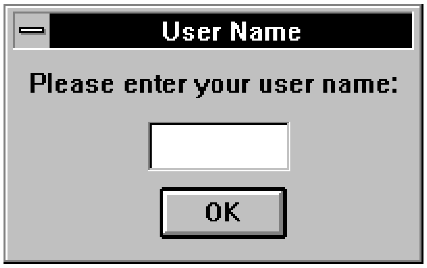
2. Choose File⇒New⇒Report. The New Report Tool dialog box opens:



3. Choose Yes to use the report wizard.

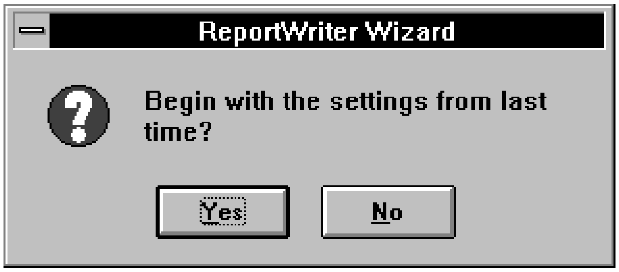
   If you choose No, a blank, untitled layout window opens in the screen editor workspace. If you choose Cancel, the new report operation is canceled.

4. If your user name is not set in your environment, the wizard asks to enter it:



   Your entry is used throughout this JAM session. A user name is required any time you access a repository or a library, but if `SMUSER`, `LOGNAME`, or `USER` is set in your environment, JAM uses that setting and omits this step. For information on setting variables, refer to page 13 in the *Configuration Guide*.

5. If you used the wizard to create a report earlier in this editing session, this dialog asks whether to start with your previous settings:



Choose one of the following:

- Yes to display previous choices in each dialog box. This lets you modify your previous report or quickly build a similar report.
- No to make new selections for each wizard dialog.

6. The Table Selection dialog box opens.

From this point, you are guided through several dialog boxes where you make these choices:

- ❍ Report type: record-by-record, column, row, graph, matrix, or address labels.
- ❍ Root database table, which provides the basis of the report information.
- ❍ Database columns to appear in the report.
- ❍ Sort order of the data.
- ❍ Columns that are totaled and how those totals appear.

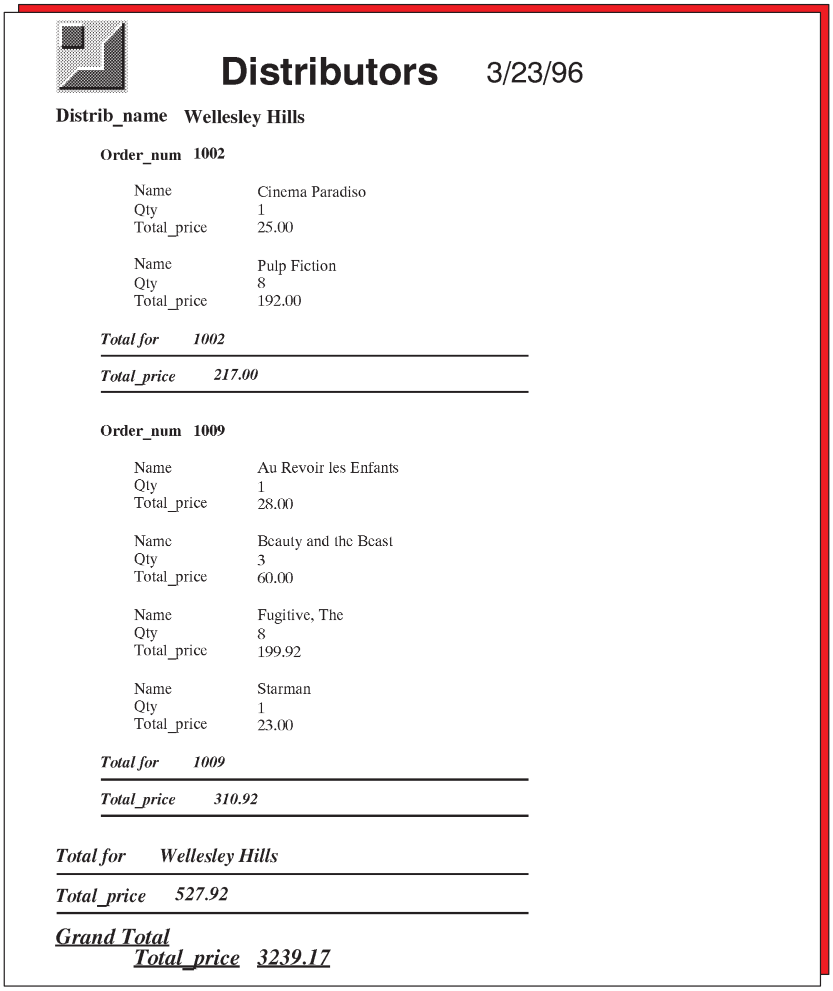
Report Types

The ReportWriter wizard can create six different report types, described in the following sections.

Record-by-Record

Record-by-record reports present each row in the select set individually. If you select group fields for the report, the data is sorted according to those groups, and a heading containing the group value is printed before the data. If any totals are requested, those totals appear after the data for each group.

In this example, `distrib_name` and `order_num` are group fields, and `total_price` is a total field.

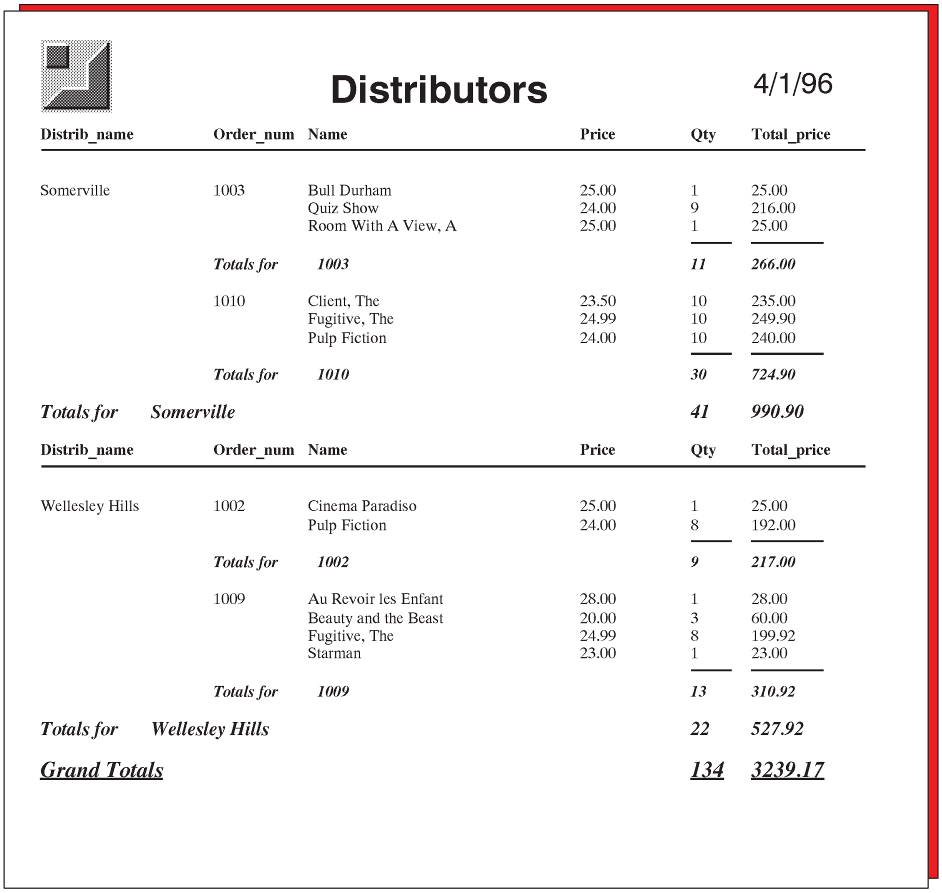


```
                    Distributors     3/23/96
Distrib_name   Wellesley Hills

     Order_num  1002

          Name             Cinema Paradiso
          Qty              1
          Total_price      25.00

          Name             Pulp Fiction
          Qty              8
          Total_price      192.00

     Total for     1002
     Total_price      217.00

     Order_num  1009

          Name             Au Revoir les Enfants
          Qty              1
          Total_price      28.00

          Name             Beauty and the Beast
          Qty              3
          Total_price      60.00

          Name             Fugitive, The
          Qty              8
          Total_price      199.92

          Name             Starman
          Qty              1
          Total_price      23.00

     Total for     1009
     Total_price      310.92

Total for     Wellesley Hills
Total_price     527.92
Grand Total
          Total_price   3239.17
```

Figure 3. Record-by-record reports list each row of data individually within group headings.

Column

Column reports present the data in columns. A single heading contains the column labels; the data is printed below each column label. If totals have been specified, they appear after the data for each group.



Distributors

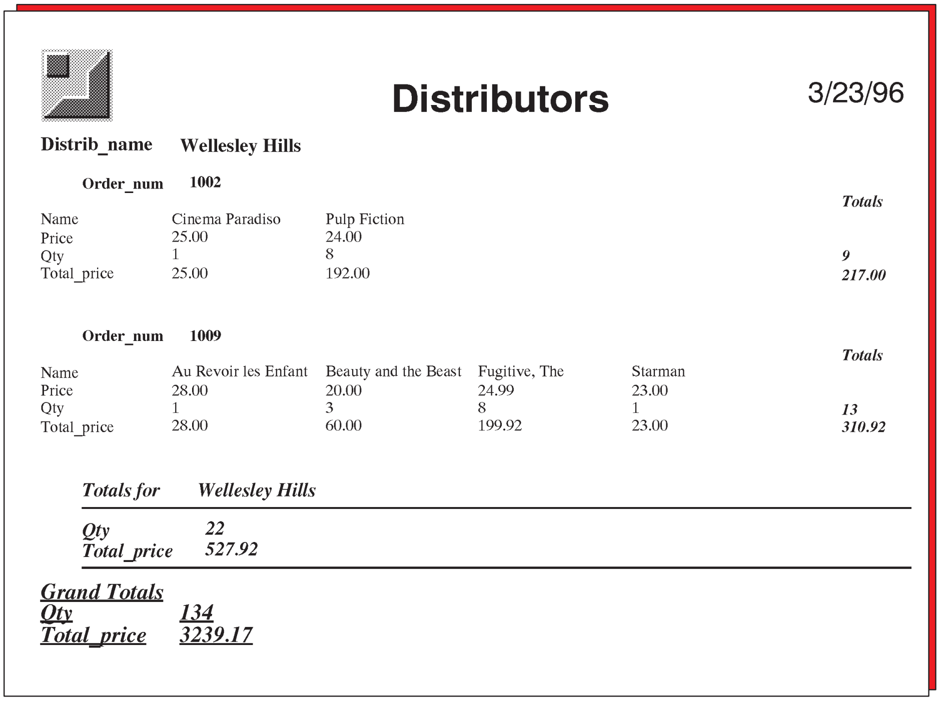
4/1/96

Distrib_name	Order_num	Name	Price	Qty	Total_price
Somerville	1003	Bull Durham	25.00	1	25.00
		Quiz Show	24.00	9	216.00
		Room With A View, A	25.00	1	25.00
	Totals for	***1003***		***11***	***266.00***
	1010	Client, The	23.50	10	235.00
		Fugitive, The	24.99	10	249.90
		Pulp Fiction	24.00	10	240.00
	Totals for	***1010***		***30***	***724.90***
Totals for	***Somerville***			***41***	***990.90***
Distrib_name	**Order_num**	**Name**	**Price**	**Qty**	**Total_price**
Wellesley Hills	1002	Cinema Paradiso	25.00	1	25.00
		Pulp Fiction	24.00	8	192.00
	Totals for	***1002***		***9***	***217.00***
	1009	Au Revoir les Enfant	28.00	1	28.00
		Beauty and the Beast	20.00	3	60.00
		Fugitive, The	24.99	8	199.92
		Starman	23.00	1	23.00
	Totals for	***1009***		***13***	***310.92***
Totals for	***Wellesley Hills***			***22***	***527.92***
Grand Totals				***134***	***3239.17***

Figure 4. Column reports list the data in columns across the page.

Row

Row reports present the data in rows across the page. Column labels are displayed on the left side of the page. To the right of each label are the values for that column. If totals are requested, row totals appear at the end of each row, and totals for higher-level groups appear after each set of row data.



Distributors 3/23/96

Distrib_name **Wellesley Hills**

Order_num **1002**

			Totals
Name	Cinema Paradiso	Pulp Fiction	
Price	25.00	24.00	
Qty	1	8	*9*
Total_price	25.00	192.00	*217.00*

Order_num **1009**

					Totals
Name	Au Revoir les Enfant	Beauty and the Beast	Fugitive, The	Starman	
Price	28.00	20.00	24.99	23.00	
Qty	1	3	8	1	*13*
Total_price	28.00	60.00	199.92	23.00	*310.92*

Totals for *Wellesley Hills*

Qty	*22*
Total_price	*527.92*

Grand Totals

Qty	***134***
Total_price	***3239.17***

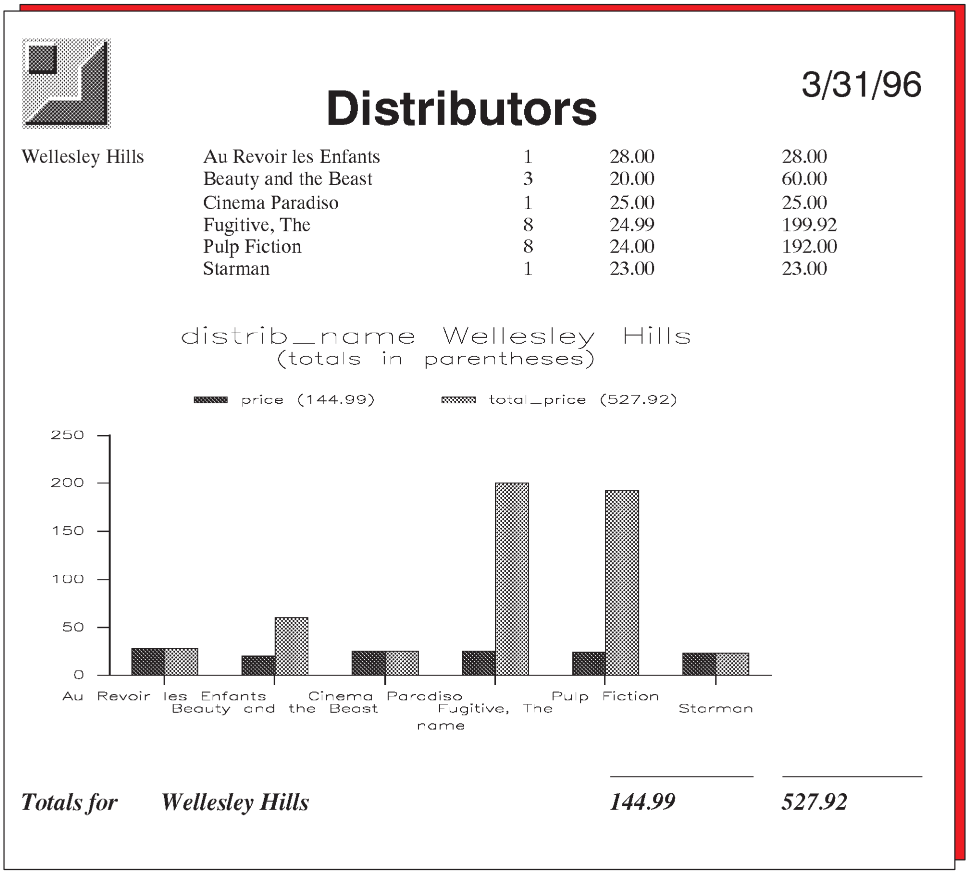
Figure 5. In row reports, the data for each column appears in a row across the page.

Graph

Graph reports display numeric data as pie or bar charts. A single report can contain a series of graphs depicting the detail section of the report, the totals section, or a combination of the two.

A graph in the detail section plots a numeric column's values for an entire group on one graph. If your report includes two or more numeric data columns, you can plot them on a single bar graph or plot each separately.

In this example, the detail data is displayed as text and in a graph while the totals are displayed as text only. The two numeric fields are displayed together in a multi-series bar graph.



Distributors

3/31/96

Wellesley Hills	Au Revoir les Enfants	1	28.00	28.00
	Beauty and the Beast	3	20.00	60.00
	Cinema Paradiso	1	25.00	25.00
	Fugitive, The	8	24.99	199.92
	Pulp Fiction	8	24.00	192.00
	Starman	1	23.00	23.00

Totals for ***Wellesley Hills*** ***144.99*** ***527.92***

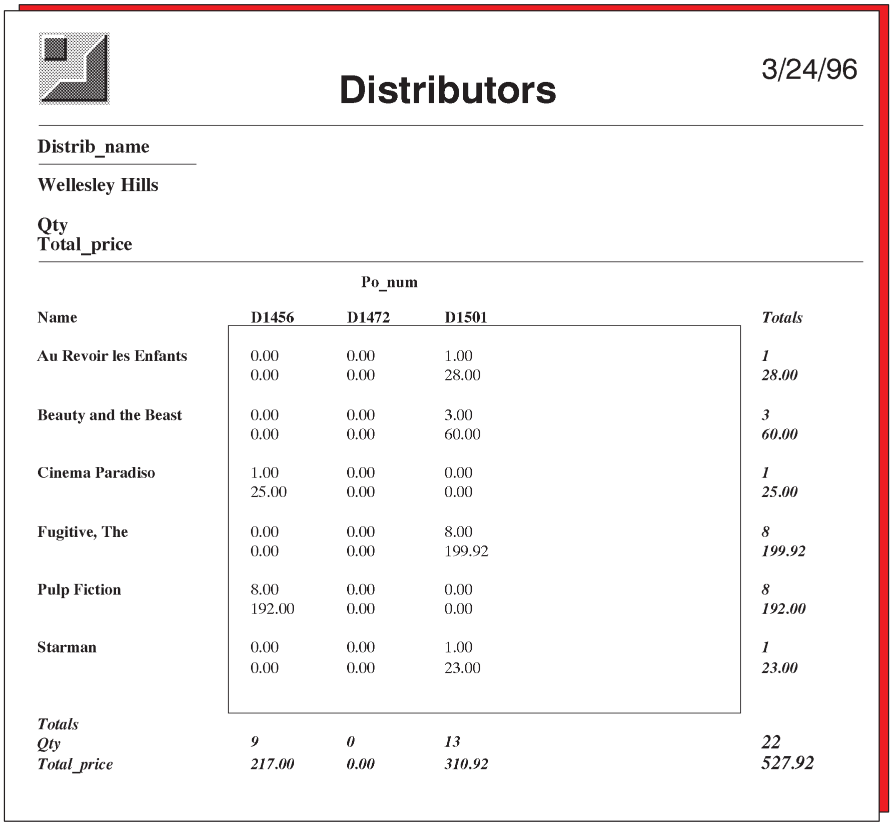
Figure 6. Graph reports can present detail data or totals in pie or bar charts.

Matrix

A matrix report presents data in a cross-tab format with totals reported independently for each row and each column in the matrix. You can also select a data group that sorts the data so that the report contains a matrix for each group. Generally, numeric data is selected to be in the cells of the matrix itself, but the column and row headings can be any column in the database.

The data items are positional in the appropriate cell; the labels for the data items appear before the matrix, following the group name. In the following sample report, the cells report the data for the `qty` and `total_price` columns.

For each fetch, a cell contains the fetch's values for the database columns that label the matrix. If more than one fetch has data for a given cell, the cell shows their total.



Distributors 3/24/96

Distrib_name

Wellesley Hills

Qty
Total_price

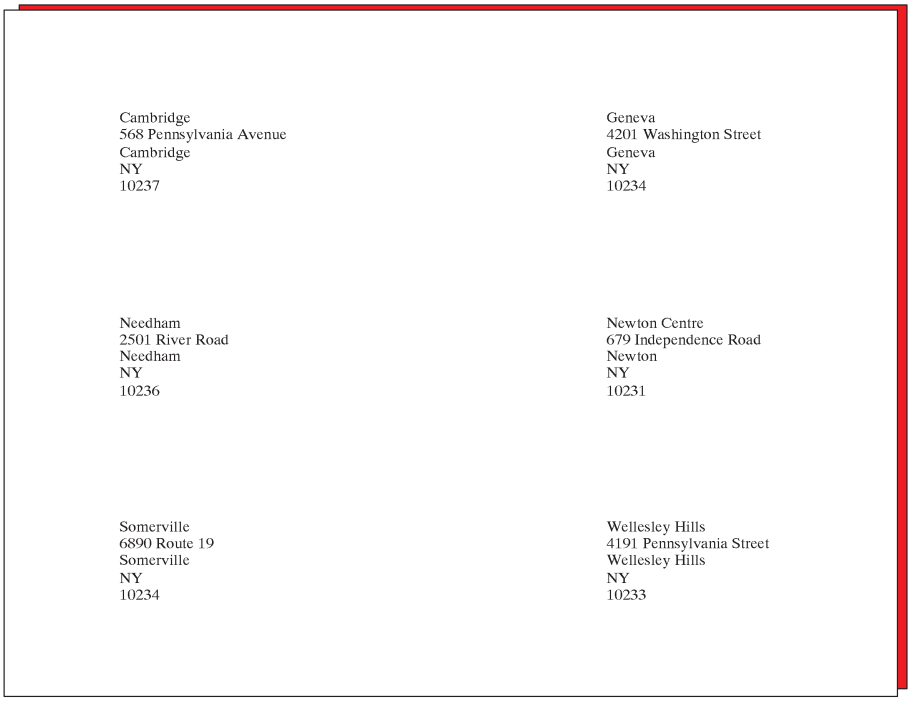
Po_num

Name	**D1456**	**D1472**	**D1501**	***Totals***
Au Revoir les Enfants	0.00 0.00	0.00 0.00	1.00 28.00	***1*** ***28.00***
Beauty and the Beast	0.00 0.00	0.00 0.00	3.00 60.00	***3*** ***60.00***
Cinema Paradiso	1.00 25.00	0.00 0.00	0.00 0.00	***1*** ***25.00***
Fugitive, The	0.00 0.00	0.00 0.00	8.00 199.92	***8*** ***199.92***
Pulp Fiction	8.00 192.00	0.00 0.00	0.00 0.00	***8*** ***192.00***
Starman	0.00 0.00	0.00 0.00	1.00 23.00	***1*** ***23.00***
Totals				
Qty	***9***	***0***	***13***	*22*
Total_price	***217.00***	***0.00***	***310.92***	*527.92*

Figure 7. Matrix reports total the values both horizontally and vertically for each cell column.

Address Labels

Address label reports produce formatted output for label sheets. The wizard aligns the widgets vertically, one widget per line. After the wizard is finished creating this report, edit the layout window and move widgets to new positions if you want more than one per line.



Cambridge
568 Pennsylvania Avenue
Cambridge
NY
10237

Geneva
4201 Washington Street
Geneva
NY
10234

Needham
2501 River Road
Needham
NY
10236

Newton Centre
679 Independence Road
Newton
NY
10231

Somerville
6890 Route 19
Somerville
NY
10234

Wellesley Hills
4191 Pennsylvania Street
Wellesley Hills
NY
10233

Figure 8. Address label reports lets you choose the number of columns per page and the number of rows per page.

Report Wizard Dialogs

The report wizard guides you through a variety of dialog boxes, prompting you for information it needs to create the appropriate presentation for your report. All dialogs provide these push buttons:

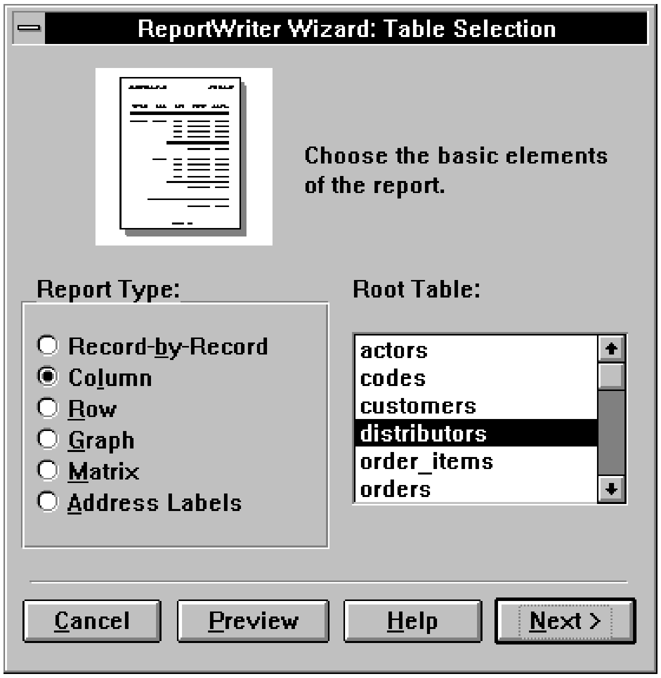
Next/Back
Traverse dialogs and their settings forwards and backwards. Choose Next to tell the report wizard that you are done with the current dialog and want to proceed to the next one. Choose Back to revisit a dialog to review its settings and make changes.

Preview
View the output from the report that is being built at any time. Acknowledge the preview display to return to the report wizard.

Help
Get information about the current dialog.

Selecting Report Type

The first dialog displayed by the report wizard, Table Selection, asks you to select the report type and root table view—the database table that forms the basis of the report:



ReportWriter Wizard: Table Selection

Choose the basic elements of the report.

Report Type:

○ Record-by-Record
● Column
○ Row
○ Graph
○ Matrix
○ Address Labels

Root Table:

actors
codes
customers
distributors
order_items
orders

Figure 9. On the Table Selection dialog, you choose the basic structure of the report.

1. Select the type of report:

   - Record-by-record — Creates a simple report comprised of a row-by-row representation of the data. To view a sample report, refer to page 24.

   - Column — (default) Creates a report where the data appears in columns down the page with the column totals following each group. To view a sample report, refer to page 25.

   - Row — Creates a report where the data appears in rows across the page. The totals for the primary group follow the group information. The totals for the lower-level data groups appear at the end of each row. To view a sample report, refer to page 26.

   - Graph — Creates a report that displays numeric data in a pie or bar chart. (Graph options are also available with other report types.) To view a sample report, refer to page 27.

- Matrix — Creates a multi-column report with both horizontal and vertical totals. To view a sample cross-tab report, refer to page 28.
- Address Labels — Creates a report with formatted output for label sheets. To view a sample report, refer to Figure 29.

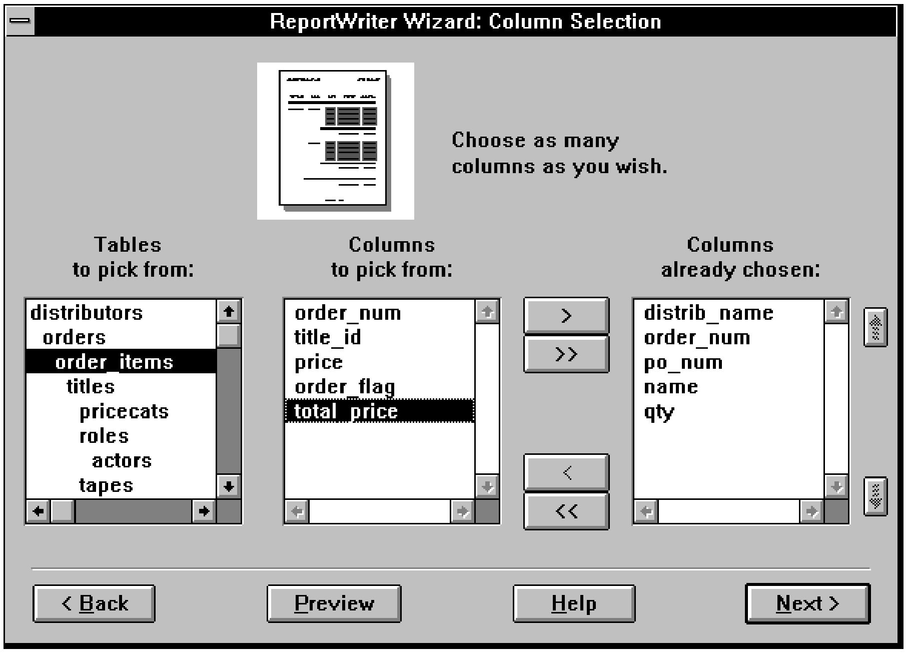
2. Select one of the database tables as the root table view.

   The root table view is the database table that forms the basis of the report. For example, a report that shows orders for each distributor can use a table view that contains distributor names for its root table view. Your report can contain data from the root table view and any table view related to it.

3. When you are done, choose Next.

Choosing Data

In the Column Selection dialog, you specify the columns to appear in the report. When the dialog opens, it lists all table views that are related to the root table view. The list is organized according to the link relationships in the repository. The root table view you selected earlier is at the top of the list. Levels of indentation indicate how all other tables are linked to the root table view and to each other:



ReportWriter Wizard: Column Selection

Choose as many columns as you wish.

Tables to pick from:

- distributors
  - orders
    - order_items
      - titles
        - pricecats
        - roles
          - actors
        - tapes

Columns to pick from:

- order_num
- title_id
- price
- order_flag
- total_price

Columns already chosen:

- distrib_name
- order_num
- po_num
- name
- qty

Figure 10. The Column Selection dialog displays a list of tables views in the open repository.

To specify columns for the report:

1. From Tables to Pick From, select a table view.

   When you select a table view, the adjoining list is populated with the names of columns in the selection.

2. To include columns in the report:

   - From Columns to Pick From, select one or more columns. Click+drag or Shift+click to select contiguous items, and Ctrl+click to select non-contiguous items.

   - Add the selections to the rightmost list, Columns Already Chosen, by choosing the > button; add all columns by choosing the >> button.

   The contents of Columns Already Chosen tells the report wizard which columns to include in the report and the order in which to display them.

3. Add or remove items from Columns Already Chosen as desired.

4. To add more tables and their columns to the report, repeat steps 1–3.

5. To change the display order of columns, reorder the items in Columns Already Chosen:

   - Select the columns that you want to move.

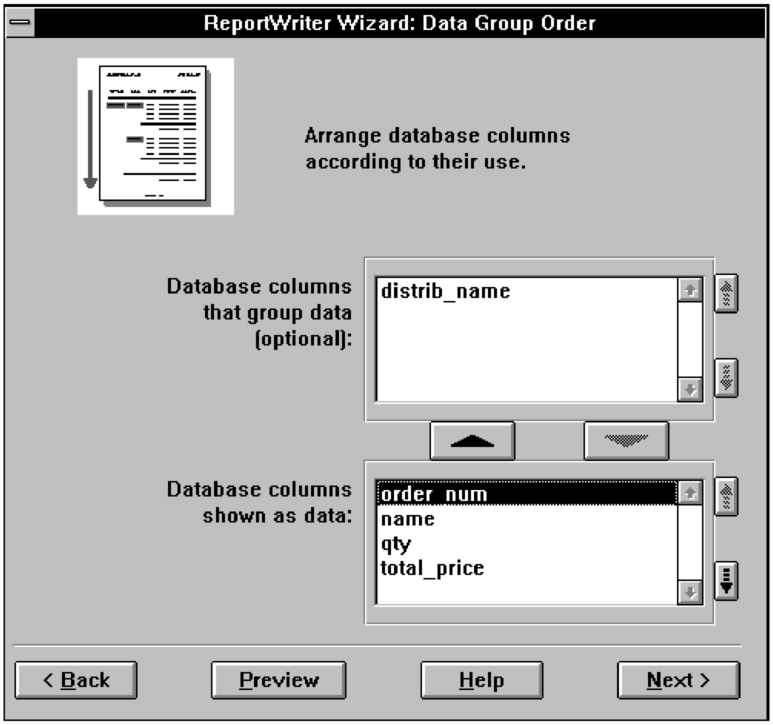
   - Use the Up or Down reposition buttons. The highlighted items shift up or down one position at a time.

6. When done, choose Next.

Grouping Data

In the Data Group Order dialog, you specify which columns to use for grouping data. Groups can help clarify relationships among data; they also let you subtotal data. For some types of reports such as matrix reports, you must also select row and column headings.

This dialog contains two list boxes. The bottom list box contains the columns chosen in the previous dialog. The list box above it, initially empty, shows the columns selected to define data groups. The first item in this box defines the primary group. Each subsequent item defines another group that is subordinate to the one listed above it.



ReportWriter Wizard: Data Group Order

Arrange database columns according to their use.

Database columns that group data (optional):

distrib_name

Database columns shown as data:

order_num
name
qty
total_price

Figure 11. In the Data Group Order dialog, select the columns that will sort the data.

To specify columns for grouping data:

1. Select a column from the bottom list box. Click+drag or Shift+click to select contiguous items, and Ctrl+click to select non-contiguous items.

2. Move the columns to the list box above by choosing ▲ .

3. Define the group hierarchy by reordering items in the top list box:

   - Select the columns from the list that you want to move.

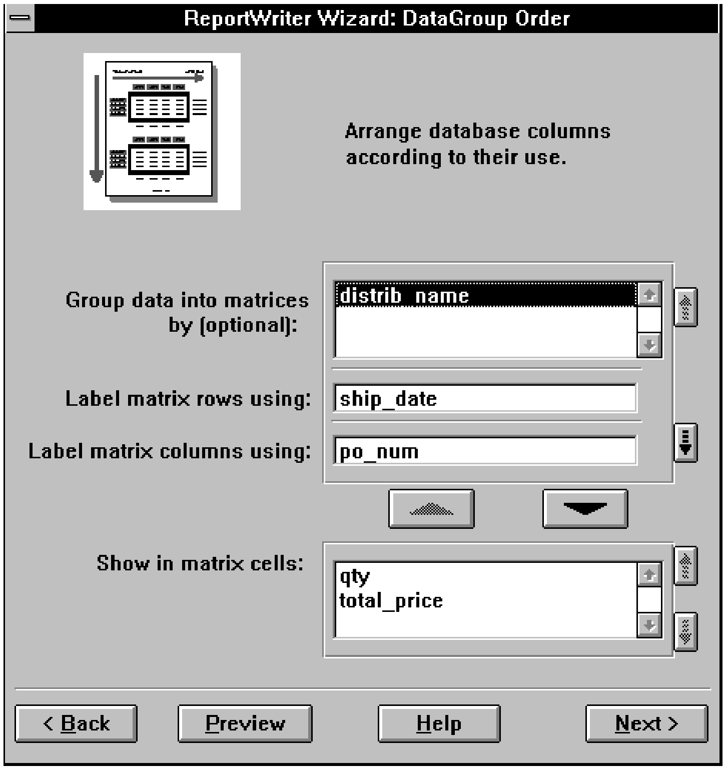
   - Use the Up or Down reposition buttons. The selected items shift up or down one position at a time.

4. When done, choose Next.

Matrix Data Groups

To create matrix reports, you must select labels for the data displayed in the matrix in addition to any group fields. For each group field, the report generates a separate matrix.

By default, the wizard lists all numeric columns in the bottom list box (Show in Matrix Cells):



ReportWriter Wizard: DataGroup Order

Arrange database columns according to their use.

Group data into matrices by (optional): distrib_name

Label matrix rows using: ship_date

Label matrix columns using: po_num

Show in matrix cells:
qty
total_price

Figure 12. For matrix reports, choose row and column headings along with the data groups.

To specify columns for grouping data:

1. Select one or more columns from the bottom list box to be a data group or a column or row label.

2. Move the selection up to Label Matrix Columns Using by choosing ▲ .

3. Select the next column from bottom list box and move it up by choosing ▲ .

   The previous selection moves up to the Label Matrix Rows Using list box.

4. Repeat step 3 until all groups and labels are selected.

5. Reorder selections so that the labels and group fields are in the desired order. Use the Up or Down reposition buttons to move items within a list.

In the final report, a separate matrix is created for the entry in the Group Data Into Matrices By list box. If more than one entry is listed, the first item in the group list specifies the primary group. Items below it specify subordinate groups in descending order.

Within each matrix, the cell entries are determined by the column label and the row label. When choosing the column label, you might want to consider the number of unique entries for that database column. The amount of space needed to report on each unique entry might exceed the width of the report. If this occurs, ReportWriter inserts ellipses to indicate missing entries.

6. Reorder the items in Show In Matrix Cells:

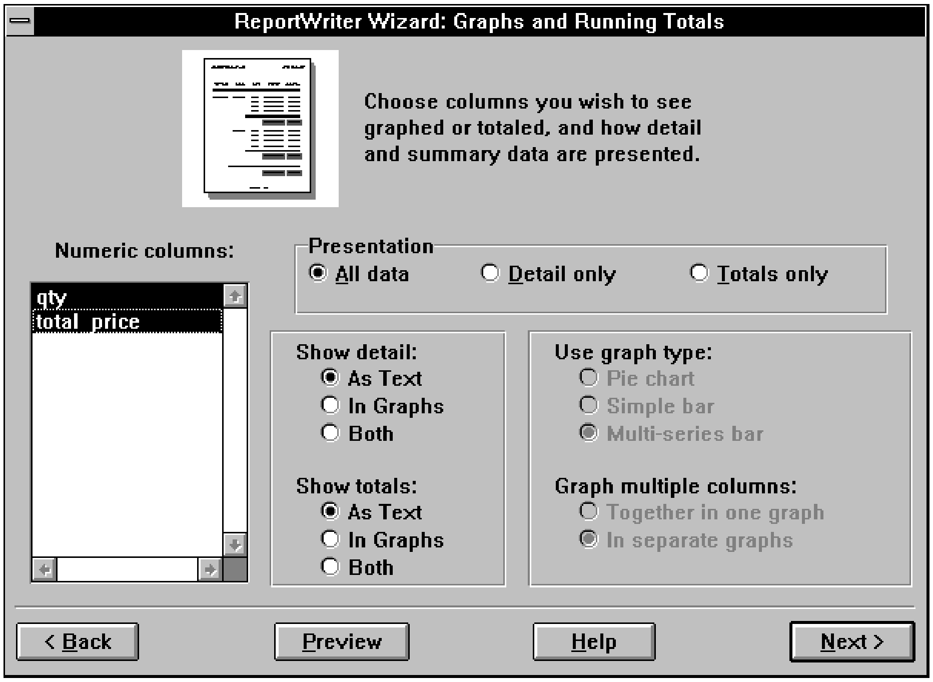
   - Select the items that you want repositioned.
   - Use the Up or Down reposition buttons to to shift highlighted items up or down one position at a time.

   When this report executes, row and column totals appear for each numeric column.

7. When done, choose Next.

Including Totals and Graphs

In the Graphs and Running Totals dialog, you decide which data to total and how to present the data. After numeric columns are selected, the default choices specify the display of all detail data with group totals. You can choose to view only the totals or only the detail data. You can also view data in graphs. The options that you can access in this dialog depend on the report type.



ReportWriter Wizard: Graphs and Running Totals

Choose columns you wish to see graphed or totaled, and how detail and summary data are presented.

Numeric columns:

qty
total_price

Presentation
● All data ○ Detail only ○ Totals only

Show detail:
● As Text
○ In Graphs
○ Both

Show totals:
● As Text
○ In Graphs
○ Both

Use graph type:
○ Pie chart
○ Simple bar
● Multi-series bar

Graph multiple columns:
○ Together in one graph
● In separate graphs

Figure 13. The Graphs and Running Totals dialog lets you decide how to present report data.

To include running totals in your report:

1. From the Numeric Columns list box, select one or more columns. Each column that you select has subtotals calculated for all data groups.

   Click+drag or Shift+click to select contiguous items, and Ctrl+click to select non-contiguous items.

2. The options available for this report type become active.

To specify the report's presentation format:

1. In Presentation, select one of these check boxes:

   - All data — Include both detail data and the totals.
   - Detail only — Include only detail data.
   - Totals only — Include only totals. This option is only available for Column and Graph reports.

2. In Show Detail, specify whether the data appears as text, graphs, or both.

3. In Show Totals, specify whether the totals appear as text, graphs, or both.

   For matrix reports, Show Totals is the only available option because each cell in the report already totals all database rows matching the cell’s row and column headings.

4. For graphs, you can specify additional graph options:

   Use graph type

   - Pie chart — The data is displayed in a pie chart.
   - Simple bar — The data is displayed in a bar chart.
   - Multi–series bar — Sets of data are displayed using multiple bar charts for each set.

   Graph multiple columns

   - Together in one graph — The values for all columns are in one graph.

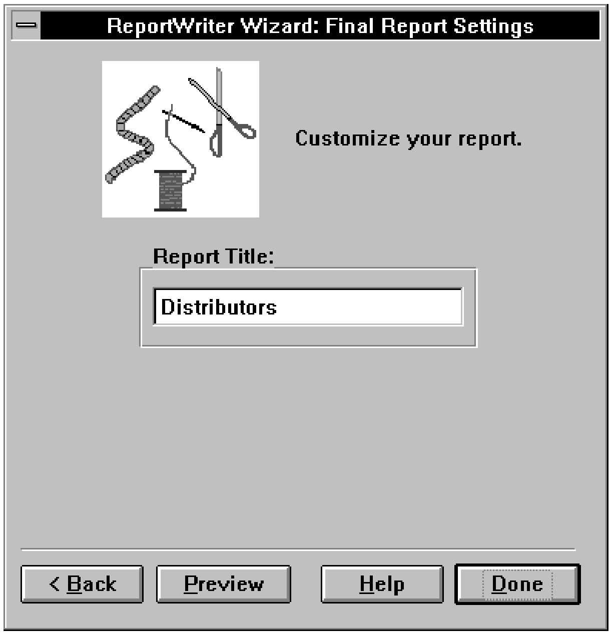
     If you choose Pie Chart or Simple Bar, each graph shows the data of one fetch. If you choose Multi–series Bar, the graph plots the history of all fetches for the innermost group.

   - In separate graphs — The values for each data set are in separate graphs. Each one shows the history of the data column’s values for all fetches in the innermost group.

5. When done, choose Next.

Finishing Up

In the Final Report Settings dialog, you set some final presentation options. The content of this dialog varies for each report type:



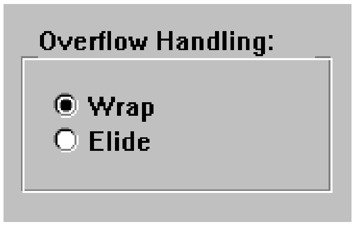
Report Title: Distributors

Figure 14. Add the final touches to your report on the Final Report Settings dialog.

To add final touches to your report:

1. Edit the Report Title entry. This title appears in the page headers.

2. For row reports, specify how to handle data overflow—that is, when the length of the row exceeds the width of the report.



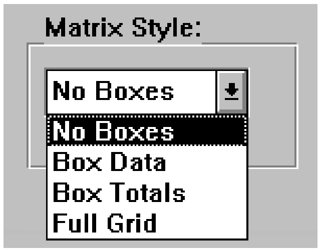
Overflow Handling:

● Wrap
○ Elide

Choose Wrap or Elide:

- Wrap — Wrap the overflow data wrap to the next line.
- Elide — Display one line of data and enter ellipses to indicate overflow.

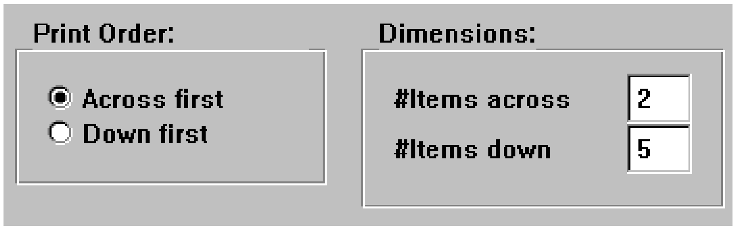
3. For matrix reports, select a style from this list box:



Matrix Style: No Boxes

- Box Data — Place a box around the detail data.
- Box Totals — Place separate boxes around the data and the totals.
- Full Grid — Display the data in a grid.

4. For address label reports, select the print order and dimensions:



Print Order:

● Across first
○ Down first

Dimensions:

#Items across 2
#Items down 5

- Print Order determines if each set of data, after it is sorted, is printed across the page first or down the page first.
- Dimensions determines the number of labels both across and down.

5. Choose Done.

   ReportWriter returns to the screen editor and displays the layout window for the completed report. To test the report output, connect to a database and choose either Report⇒Preview Report or File⇒Test Mode.

How It Looks

When the report wizard is complete, the layout window for your report appears in the editor workspace. You can save it and use it immediately; or you can continue to work on it—rearrange widgets, add decorations, or change property settings.

Layout Window

The layout window contains the layout areas created for the report. A horizontal line across the entire window defines the end of each layout area; the name of each layout area appears at the center of this line. The order of the layout areas is independent how they appear in the report, which is determined by the report structure.

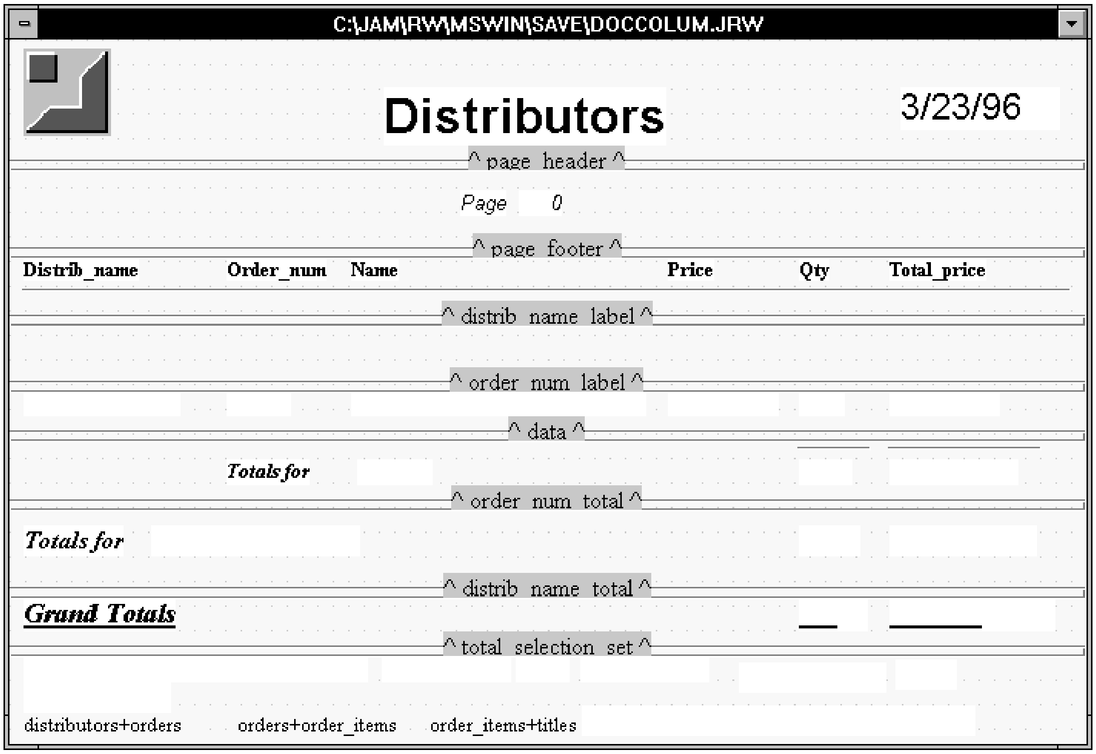


Figure 15. The layout view contains the layout areas formatted as they will appear in the report.

Generally, the layout window for wizard-created reports contains layout areas for these report components:

- Page header — Contains a pixmap with JYACC’s logo, the report title, and the current date.
- Page footer — Contains the page number.
- Group header labels — Depending on the report type, there are one or more label areas. Most reports have a separate label area for each group level, named after the database column that defines the group.
- Detail output — Contains the format of the detail data section.

- Chart — Contains a graph widget. Note that the properties that determine data sources for this graph are set at runtime.
- Group totals — Depending on the report type, there are one or more total areas used to output group footers. A total area is created for each column selected in the Graphs and Running Totals dialog. The layout area `total_selection_set` outputs grand totals.

All widgets that output data are located in layout areas. The unnamed area of a layout window—that is, the space below all named layout areas—contains widgets that do not output data. The wizard places link widgets and history widgets in this area. They are needed to calculate and fetch data for the report, but they are not needed in the report layout itself.

Report Structure Window

The report structure window displays a diagram of the report structure. Each report element—format, data groups, detail data, and the report itself—has a corresponding node in the structure:

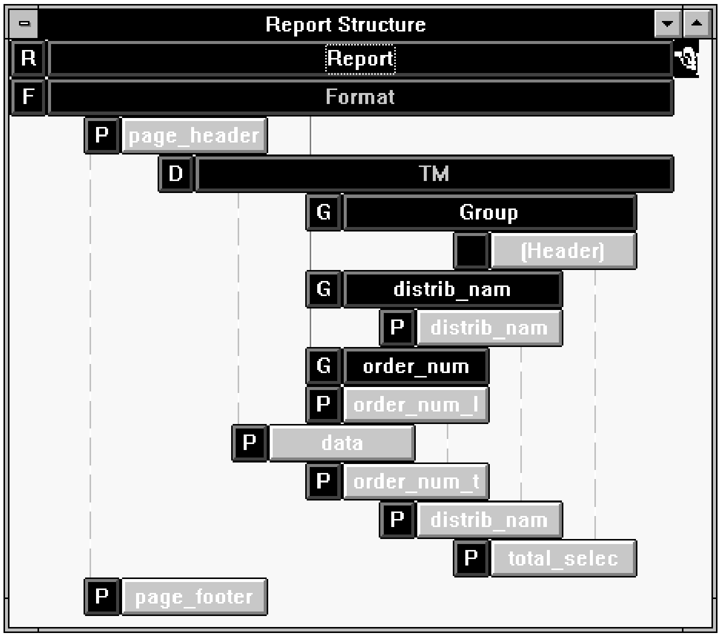


Figure 16. The report structure window schematically shows how a report executes.

The main report node appears at the top of the report structure; its page format node is directly below it. The detail node specifies how the data is fetched for the report. Each data group level has a corresponding group node whose footer performs the calculations for the group totals. The print nodes attached to the detail and group notes have associated layout areas to provide the report output.

Property Settings

A number of properties, both on the report file itself and on individual widgets, are automatically assigned values. This section briefly describes some of the properties that are set by the report wizard.

Report file

The report wizard defines and sets the following properties for the report file:

- Under Inclusions, the JPL Procedures property makes public the prototype JPL file `rwwizard.jpl`.
- Under Transaction, the Root property is set to the table view selected on the Table Selection dialog box.

Page orientation

If the width of the report exceeds 8.5 inches, the Orientation property is set to Landscape. Otherwise, the property value is set to Portrait.

Data fetching

For each detail node, the Data Source property is set to TM View. This indicates that the transaction manager executes `VIEW` and `CONTINUE` commands to fetch the report data. When the transaction manager executes these commands, data is fetched for any dynamic output widget that is a member of the current transaction.

Table views

For the root table view, the Sort Widgets property contains the names of widgets not included in the report totals. If you delete a widget from the report, you also must delete the widget from the Sort Widgets property. Otherwise, you get an error.

Data grouping

For each group node, the Break On property is set to a database column chosen in the Data Group Order dialog box. The hierarchy of group nodes corresponds to the order selected on that dialog.

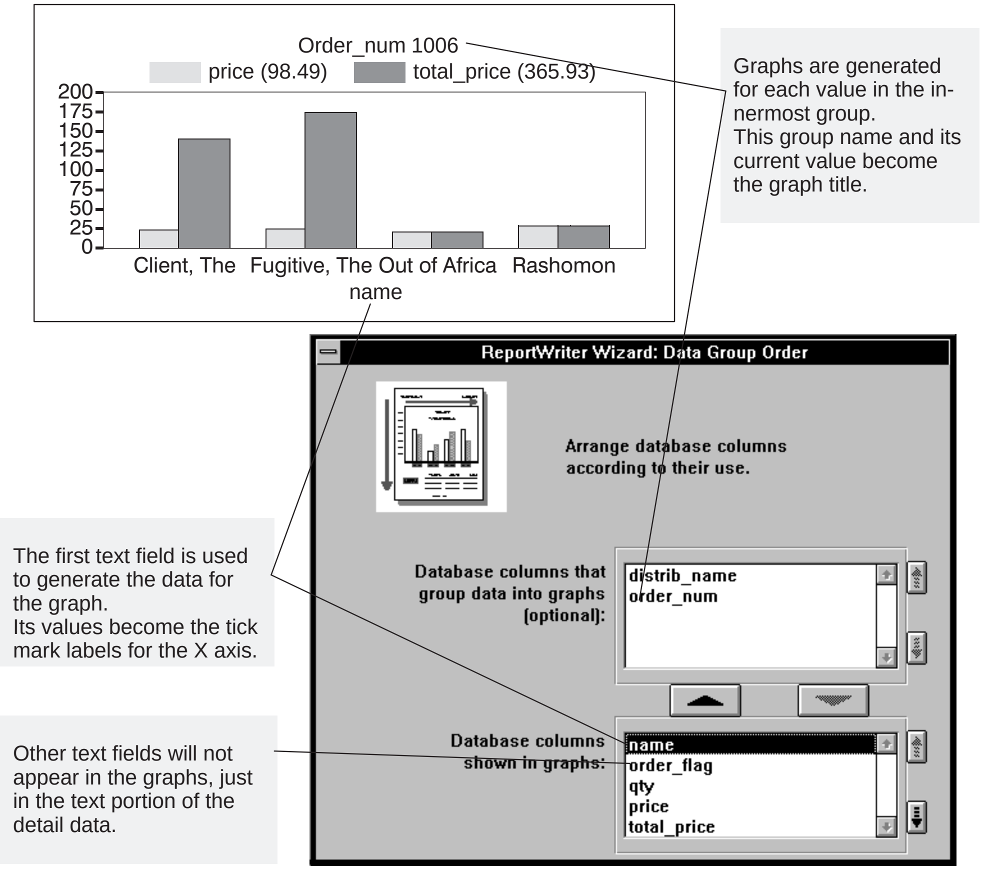
Output

For each print node, the Area property is set to the appropriate layout area.

Including Graphs

Numeric data in your report can be presented as a graph. Both pie and bar charts are available and can be specified in the Graphs and Running Totals dialog.

The selection of data groups determines which column values are selected to appear in the graph. The graph is created for the innermost data group.


Order_num 1006
price (98.49)
total_price (365.93)
200
175
150
125
100
75
50
25
0
Client, The
Fugitive, The
Out of Africa
Rashomon
name
ReportWriter Wizard: Data Group Order
Arrange database columns according to their use.
Database columns that group data into graphs (optional):
distrib_name
order_num
Database columns shown in graphs:
name
order_flag
qty
price
total_price

Graphs are generated for each value in the innermost group.
This group name and its current value become the graph title.

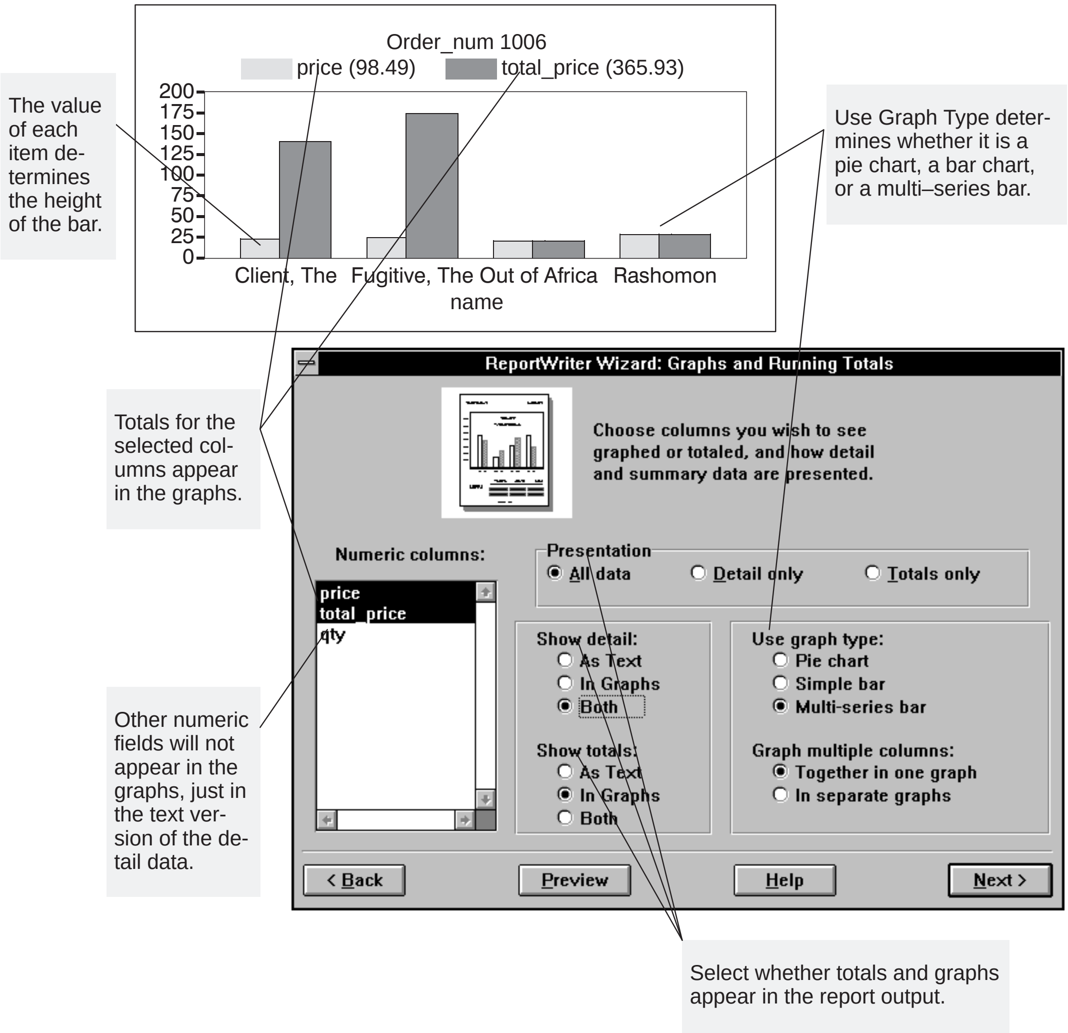
The first text field is used to generate the data for the graph.
Its values become the tick mark labels for the X axis.

Other text fields will not appear in the graphs, just in the text portion of the detail data.

Figure 17. This graph, which is for an order in the `vbizplus` *database, shows the film titles that are included in the order and the price and total price for each film title.*

Because graphs display numeric data, text columns in the data set are discarded if only graphing options are chosen for the detail data section.

If your report includes two or more numeric data columns, you have a choice of graphing options. You can choose to plot them on a single graph. Alternatively, you can graph each separately, and so plot the column's values for an entire group in one graph.



ReportWriter Wizard: Graphs and Running Totals

Choose columns you wish to see graphed or totaled, and how detail and summary data are presented.

Numeric columns:
- price
- total_price
- qty

Presentation: ● All data ○ Detail only ○ Totals only

Show detail:
○ As Text
○ In Graphs
● Both

Show totals:
○ As Text
● In Graphs
○ Both

Use graph type:
○ Pie chart
○ Simple bar
● Multi-series bar

Graph multiple columns:
● Together in one graph
○ In separate graphs

The value of each item determines the height of the bar.

Use Graph Type determines whether it is a pie chart, a bar chart, or a multi–series bar.

Totals for the selected columns appear in the graphs.

Other numeric fields will not appear in the graphs, just in the text version of the detail data.

Select whether totals and graphs appear in the report output.

Figure 18. In this report, the detail section of the report will contain text output and business graphs while the totals section will contain only business graphs

Note: *A report that runs on Macintosh or in in character mode omits display of graphs. However, a layout area is created for the graph, which displays on other platforms.*

Customizing Wizard Reports

There are a variety of things you can do to enhance the screen created by the report wizard. The following is a list of suggestions:

Modify repository prototype

The report wizard stores and uses a prototype in the repository—`smwizrw`. This report contains prototypes for two report areas, page header and page footer, and for widgets such as lines, labels, and graphs.

You can modify this report to customize the report wizard's output; these changes are propagated to new and existing reports through inheritance. Note that existing reports are affected only when you bring them into the editor and resave them.

Change fonts

The wizard sets the Font Name property to match `smwizrw`: for the report file, JAM Times Roman; and for widgets in page header and footer areas, JAM Helvetica. For widgets in all other layout areas, the Font Name property is set to Default.

By using JAM fonts, you can easily change their mapping to platform-specific fonts by simply editing the configuration file. For more information about report fonts and configuration file settings, refer to page 56.

Because font settings affect the placement and spacing of widgets in reports, you might want to change the Font Name setting in your `smwizrw` repository entry to use the fonts available in your environment.

Modify the JPL file

The report wizard's prototype JPL code resides in an editable ASCII file `rwwizard.jpl`, which is in the JAM `config` directory. For faster loading, compile this file with the `jpl2bin` utility (refer to page 11 in the *Language Reference*).

Customize repository widgets

Widgets in JAM repository entries obtain their original property settings during the importing process. Because wizard-created reports copy widgets from these repository entries, you might want to change the property values set by the importer to more useful values. For example:

- ❍ Edit the Label property of the static labels.
- ❍ Edit the Length property of dynamic output widgets.

- Change the Format properties for widgets with numeric and date/time output.
- Create a widget that contains a derived or computed value (refer to page 104).

Change the pixmap
To change the pixmap displayed in the page header of the reports, select the pixmap, open the Format/Display category, and change the name in the Active Pixmap property. The pixmap must be located either in the local directory or in a library that is open when the report executes.

Troubleshooting

Refer to this section for additional guidance in using the report wizard.

Creating the Repository

To create a report, the report wizard copies widgets from a repository. Consequently, the repository must be set correctly to ensure that the wizard works properly.

The repository must contain at least one table view that inherits from `@DATABASE`, specified in the objects' Inherit From property. This tells the wizard that widget definitions are from an existing database. If the wizard cannot find this setting, it displays an an error. For more on creating repositories, refer to page 55 in the *Editors Guide*.

You can import the desired database objects—tables, views, and synonyms—to the open repository. Each imported object becomes a separate entry in the repository. For more on importing, refer to page 59 in the *Editors Guide*.

Using Multiple Table Views

When you import the database objects, foreign key definitions are also imported to the repository as link widgets. If foreign keys are not defined for the database (or not supported by the DBMS), you must manually create the link widgets in each repository entry in order to have more than one table view in the report. For more information on link widgets, refer to page 343 in the *Editors Guide*.

Adding the Wizard Prototype to the Repository

When you run the report wizard for the first time, it automatically creates a prototype report—`smwizrw`—in your open repository. The report wizard uses this prototype to set inherited properties in reports.

If the prototype report in the repository is from an old version of the report wizard, you are asked whether to replace it. If you made changes or additions to the prototype screen, first back it up, then let the report wizard overwrite it. Later, you can restore your changes to the new prototype.

Note: *If the report wizard needs to add or update its prototype screen, make sure you can write to the open repository. If the repository already contains a valid prototype screen, write permissions are not necessary to run the report wizard.*

Using a Read-Only Repository

If a read-only repository does not contain a valid version of `smwizrw`, you cannot use it with the report wizard. You must have write permission for the repository in order to run the report wizard the first time. The person who administers the repository for the development team can run the report wizard after importing the database tables; thereafter, others can run the report wizard without having write permission.

3

Defining Report Layout

ReportWriter's layout window defines the format and content of report output. It typically contains one or more *layout areas*, named spaces whose contents are output at runtime. A layout area can only by invoked through a print node in the report structure; the timing and sequence of the layout area's output depends on execution of its print node.

A layout area outputs data through the widgets that populate it. Each widget's properties determine the source and format of its data; the widget's position within the layout area determines where that output appears in relation to other output from the same layout area.

A layout area is defined by a layout area widget, which set the area's boundary and name. The layout area spans the width of the layout window and extends to the layout area widget above it or to the window's border. The layout window also contains an unnamed area whose contents are not subject to output.

For example, the following layout window contains three layout areas for detail, page header, and page footer output, defined as `data`, `page_header`, and `page_footer`, respectively:

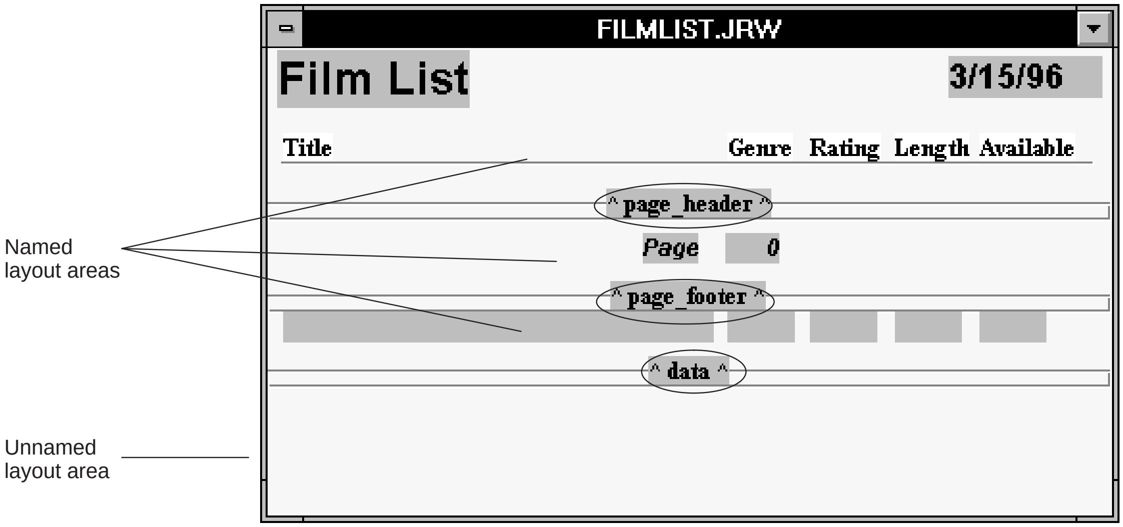


Figure 19. A layout window contains layout areas and an unnamed area.

The following report structure contains three print nodes that specify when to output the layout areas shown earlier:

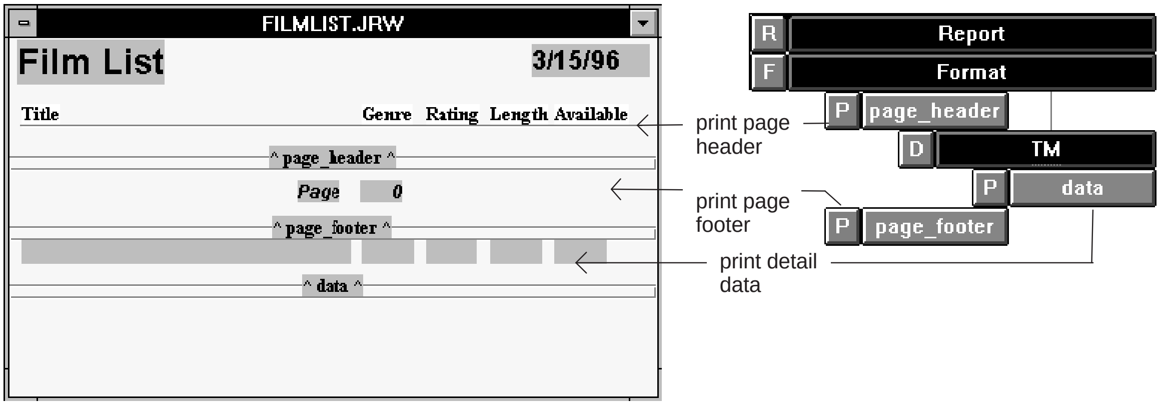


Figure 20. Print nodes specify which layout areas to output, and when.

Creating Layout Areas

You define a layout area by creating a layout area widget:

1. Give the layout window focus.
2. Choose Create⇒Layout Area or the corresponding toolbox button.
3. Point and click where you want the new area to appear in the layout window.

ReportWriter creates an unnamed layout area widget, which defines the area between it and the layout area widget above it.

Because layout areas can only be invoked by name, you should always set a new layout area's Name property (under Identity). Like other widgets, layout area names within the same report file must be unique. For information about JAM naming conventions, refer to page 299 in the *Editors Guide*.

Managing the Layout Window

You can modify the view of report layouts by collapsing some or all of the layout areas, or by changing their relative positions. This can help maximize the amount of working space available for a given area and lets you rearrange the window's contents in a way that makes sense to you. You can also manipulate a layout area widget in order to change the area's size.

To collapse and expand layout areas:

1. Select the layout areas that you want to collapse or expand.
2. From the Edit menu, bring up the Arrange submenu and choose the desired option:
   - Collapse — Reduces each selected area to a single line and hides its contents.
   - Expand — Restores the selected layout areas to their full dimensions.
   - Collapse All — Reduces all layout areas to a single line; the unnamed area at the window's bottom is unaffected.
   - Expand All — Restores all layout areas to their full dimensions.

Note: *When you save or preview a report, ReportWriter automatically restores all collapsed layout areas to full view.*

To move layout areas:

1. Select the desired layout area widgets.
2. From the Edit menu, bring up the Arrange submenu and choose the desired option:
   - Move Up moves the selected areas above the areas previously above them.
   - Move Down moves the selected areas below the areas previously below them.

Note: *Rearranging layout areas in the layout window has no effect on report output.*

To change layout area dimensions:

- ❍ To enlarge a layout area, select the layout area widget and drag it down as far as needed.
- ❍ To reduce a layout area, select the layout area and drag it up. You can move the layout area widget up only as far as the first widget inside that area. To reduce the area further, you must first move up the widgets inside it.

In both cases, changing the dimensions of one layout area has no effect on the others.

Note: *You can only change a layout area's height; all layout areas occupy the entire width of the report page.*

Editing Layout Area Properties

Each layout area has a set of properties that are described below. These properties set format and behavior parameters for the entire layout area. Select the layout area widget and bring the Properties window into focus.

Name

Only named layout areas can be specified for output in the report. Layout area names must be unique among all other named widgets and nodes in the report file. Refer to page 299 in the *Editors Guide* for more information about widget naming conventions and requirements.

Memo Text

Provides up to nine lines of text for comments or programmatic use. For information, refer to page 116 in the *Editors Guide*.

Inherit From

Defines the source of inheritance—the name of the repository report followed by the name of the parent object in this format:

repository_entry`!`*widget_name*

For more information about inheritance, refer to page 63 in the *Editors Guide*.

Start Row

Specifies the layout area widget's position relative to the layout window's grid. For more information, refer to page 47 in the *Editors Guide*.

Font properties

Sets the default font properties—Font Name, Point Size, and so on—for all widgets in this layout area. If the layout area's font properties are set to Default, at runtime JAM resolves which fonts to apply according to what is set at a higher level. For more information about how fonts are set, refer to page 56.

Vertical Anchor

Specifies how the layout area widget aligns itself to the layout window's grid when you choose Edit⇒Grid Align or Options⇒Snap To Grid:

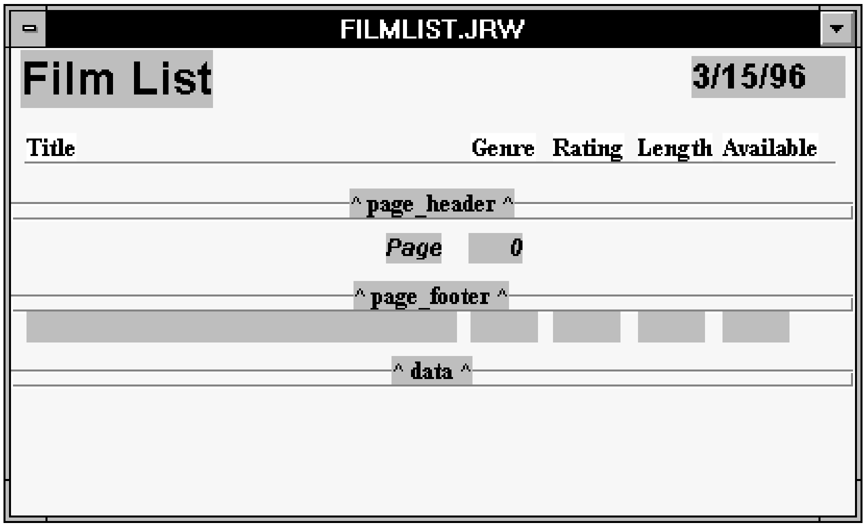
- Middle (default) — The layout area widget snaps to the middle of the nearest vertical coordinate.
- Top — The layout area snaps to the top of the nearest vertical coordinate.
- Bottom — The layout area snaps to the bottom of the nearest vertical coordinate.

Populating Layout Areas

In general, ReportWriter outputs the data of all widgets in an invoked layout area. Two widget types are responsible for most report output:

- *dynamic output* widgets get their data at runtime—for example, from a database or from other widgets.
- *static output* widgets have their data set in the editor; their data remains constant.

For example, the following layout window contains three layout areas—`page_header`, `page_footer`, and `data`:

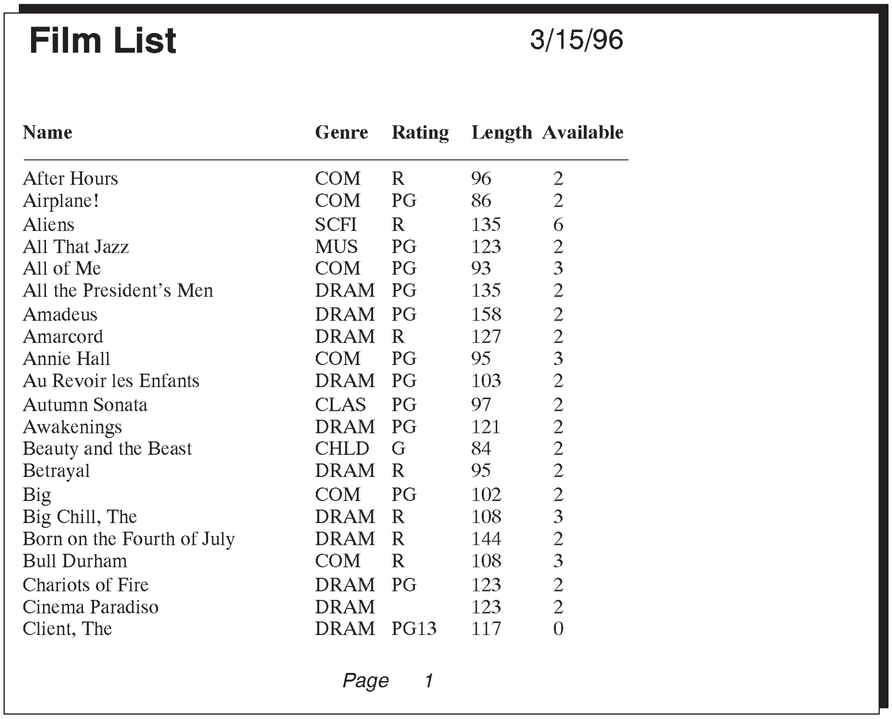


- `page_header` is output as the report's page header. It contains two dynamic output widgets for the report title and system date. It also contains static output widgets that at runtime appear over columns of data; their values are fixed.

- `page_footer` is output as the report's page footer. It includes a widget that outputs page numbers.

- `data` contains dynamic output widgets that at runtime display the film data—name, genre, and so on; these widgets are initially empty and get their data at runtime, usually from a database.

A report that uses this layout might yield the following output:



Film List 3/15/96

Name	Genre	Rating	Length	Available
After Hours	COM	R	96	2
Airplane!	COM	PG	86	2
Aliens	SCFI	R	135	6
All That Jazz	MUS	PG	123	2
All of Me	COM	PG	93	3
All the President’s Men	DRAM	PG	135	2
Amadeus	DRAM	PG	158	2
Amarcord	DRAM	R	127	2
Annie Hall	COM	PG	95	3
Au Revoir les Enfants	DRAM	PG	103	2
Autumn Sonata	CLAS	PG	97	2
Awakenings	DRAM	PG	121	2
Beauty and the Beast	CHLD	G	84	2
Betrayal	DRAM	R	95	2
Big	COM	PG	102	2
Big Chill, The	DRAM	R	108	3
Born on the Fourth of July	DRAM	R	144	2
Bull Durham	COM	R	108	3
Chariots of Fire	DRAM	PG	123	2
Cinema Paradiso	DRAM		123	2
Client, The	DRAM	PG13	117	0

Page 1

white space

A layout area’s output includes white space. The height of the layout area determines the amount of space the layout area uses. White space is reduced or eliminated only in the case of shrinkage (page 122) and consolidation (page 126).

A layout area can also include other widgets: graph widgets let you present data as a pie or bar graph; box and line widgets can be used to enhance a report’s appearance. For example, `page_header` contains a horizontal line that underlines the column labels in the actual report.

Refer to the *Editors Guide* for more information on these widget types: page 259 for lines and boxes; page 121 for graph widgets. For information about presenting report data in a graph, refer to page 101.

Copying Widgets to a Layout Window

You can create widgets directly from the Create menu or toolbox; or you can copy them from the layout window of another open report or a JAM screen. When ReportWriter copies widgets from a JAM screen, it checks their type and makes the following adjustments:

- Line, box, and graph widgets are copied directly to the target layout window with no changes.

- Static labels are converted to static output widgets.
- Grid frame widgets cannot be copied to a report.
- All other widget types are converted to dynamic output widgets.

inheritance

When you copy widgets from a repository entry, ReportWriter sets the widget’s Inherit From property and applies JAM’s rules of inheritance to the copied widgets. A widget that is copied directly from a repository entry inherits the properties of its source. If you copy an object from a screen or from another report, any properties that it inherits are propagated to the copy.

Inheritance helps ensure a widget is consistent in its appearance and behavior wherever it appears in the report or elsewhere in the application. For example, you might want all monetary values to conform to the same format. Setting the appropriate properties in repository objects lets you propagate this format to output widgets in your reports.

For more information about inheritance, refer to Chapter 4 in the *Editors Guide*.

Positioning Widgets

By default, a widget’s position is fixed in relation to its layout area. If widgets that are on the same line are wider than the data that they actually output, you can eliminate the unused space and thereby make their data contiguous. Set their Placement property (under Composition) to one of these values:

- Float Left — Trim the width of the previous widget to its actual output; float this widget’s start position left by the number of trimmed characters.
- Float Right — Trim leading blanks from the next widget; float this widget’s start position right by the number of trimmed characters.

ReportWriter floats widgets only to the extent that their own widths are trimmed. The actual amount of white space between widgets remains unaffected.

Note: *To make sure that widgets float towards each other, align them horizontally so that their Start Row properties have the same value. If Start Row properties are not the same, ReportWriter might use their absolute (fixed) positions.*

JAM’s screen editor provides a number of options for aligning and spacing widgets within the layout window—for example, Edit⇒Align and Options⇒Snap to Grid. For more information about arranging widgets, refer to page 46 in the *Editors Guide*.

Using Fonts

You can format all widget output by applying different fonts and font sizes and setting one or more attributes—bold, italics, and underlined. Fonts can be chosen from these categories, listed in descending order of universality:

- PostScript fonts, either by name or through the aliases that are defined in the configuration file's `[Postscript Fonts]` section. PostScript fonts are available to all platforms.
- TrueType fonts, available only to reports run on Windows and Macintosh.
- All Macintosh fonts, available only to reports run on Macintosh.

If you specify a font that is undefined for a given environment, the device driver determines the actual output.

Although the editor lets you choose any GUI-resident font and font alias, you should use only those fonts that are valid for report output —or *printable fonts*. If you intend to run reports on multiple platforms, use font aliases to ensure the desired output in each environment.

Setting Fonts

When you create a report, a default font is applied to all layout areas and output widgets. This default is derived from the configuration file. Under the Font heading in the Properties window, all font-specific properties are initially set to Default for the report file, layout areas, and output widgets. Only objects that inherit their font properties from a repository or were copied from another source have their own font assignments.

You can set fonts at several levels in a report, listed below in order of precedence:

1. Individual widgets — Each output widget can have its own font settings. These override any defaults that are set at a higher level. The widget adjusts in size according to the font specification.
2. Layout area — The font properties set for the layout area are used by all widgets in that area that lack their own settings.
3. Report file — The font properties set for the report file are used by all layout areas that lack their own settings. To access report file font properties, give focus to the layout window and bring the Properties window into view.
4. Application — The applicable section in the configuration file sets default printing font properties for all reports generated by this executable.

Defining Font Aliases

You can define GUI-independent printing font aliases in the configuration file with the following format:

```
alias-name  [ (font-qualifier...)  ]      =  font-spec
            [ [ (font-qualifier...) ]     =  font-spec  ]...
```

The default configuration file defines four aliases:

```
JAM Courier
JAM Times Roman
JAM Helvetica
JAM Symbol
```

The default configuration file maps each of these aliases to a valid font for a given platform. For example, the default configuration file defines font alias `JAM Times Roman` in the `[Windows]`/`[Macintosh]`, `[Postscript Fonts]`, and `[Text Fonts]` sections:

```
[Windows Fonts]
# JAM Font Name   Qualifiers   Windows font
# -------------   ----------   ------------
JAM Times Roman                = Times New Roman


[PostScript Fonts]
# JAM Font Name   Qualifiers    PostScript font
# -------------   ----------    ---------------
JAM Times Roman   (italic bold) = Times-BoldItalic
                  (italic)      = Times-Italic
                  (bold)        = Times-Bold
                                = Times-Roman
[Text Fonts]
# JAM Font Name   Qualifiers    Text Font
# -------------   ----------    ---------
JAM Times Roman                 = Times-Roman
```

Given this definition, JAM can resolve usage of JAM Times Roman in a report, regardless of the platform that it runs on. For more information on how JAM resolves font aliases, refer to page 148 in the *Configuration Guide*.

Note: *All aliases that are defined in the Text Fonts section are resolved via the device configuration file. For more information*, refer to page 175.

4

Building the Report

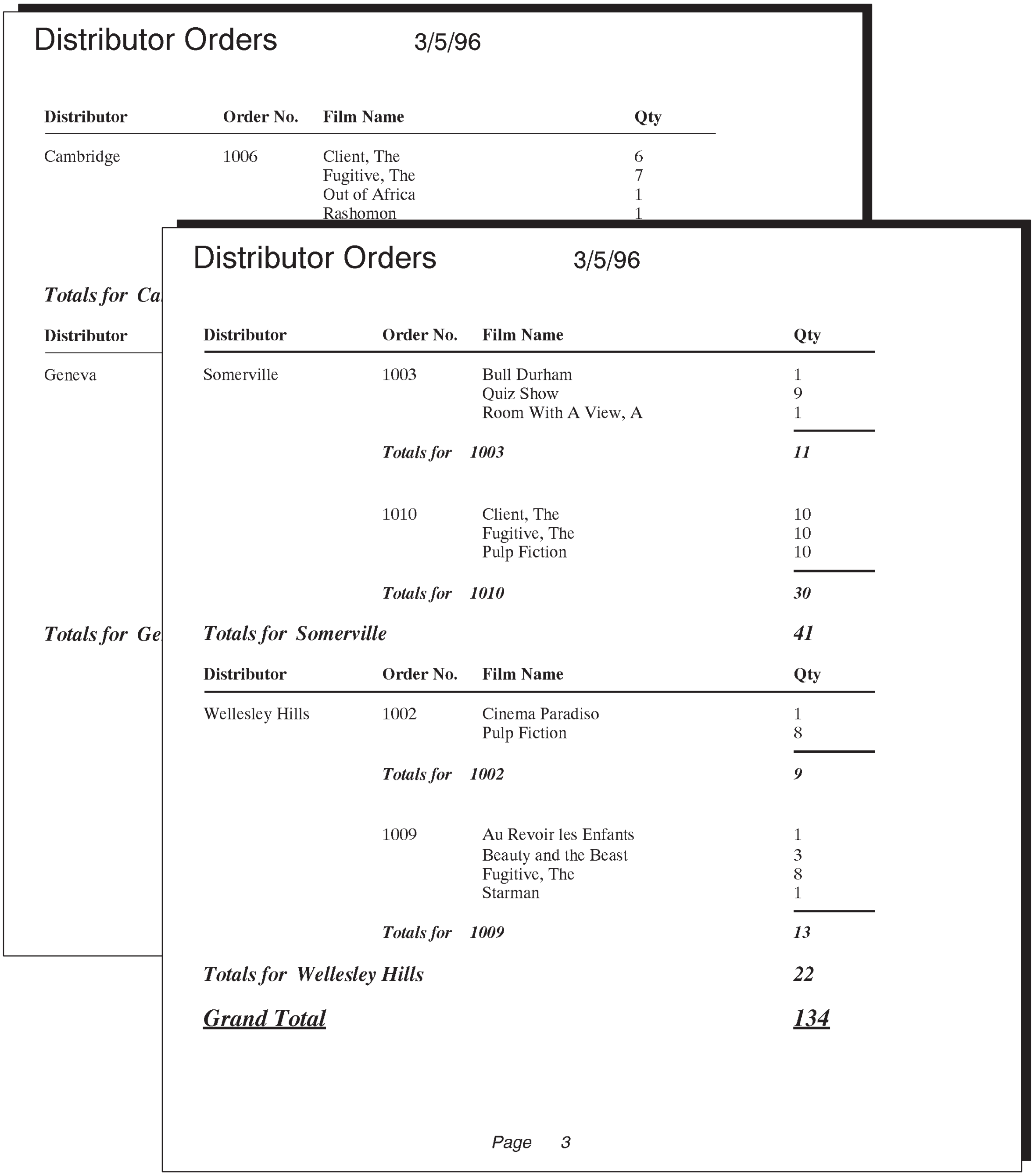
ReportWriter offers two views into a report file: the physical layouts used by a report are shown in the layout window, while specifications for report generation are schematically rendered in the report structure window.

This chapter focuses on report structure components and their properties, and how to manipulate these to modify report execution.

Report Structure Window

The report structure window schematically shows how a report executes. Stages of report execution are depicted in a vertical hierarchy composed of *nodes*. The topmost node defines a report; when this report runs, all nodes below and subordinate to it execute in top-down order.

For example, the following column report sorts orders from video store outlets by distributor name and order number:



Distributor Orders 3/5/96

Distributor	Order No.	Film Name	Qty
Cambridge	1006	Client, The	6
		Fugitive, The	7
		Out of Africa	1
		Rashomon	1

Totals for Ca

Distributor

Geneva

Totals for Ge

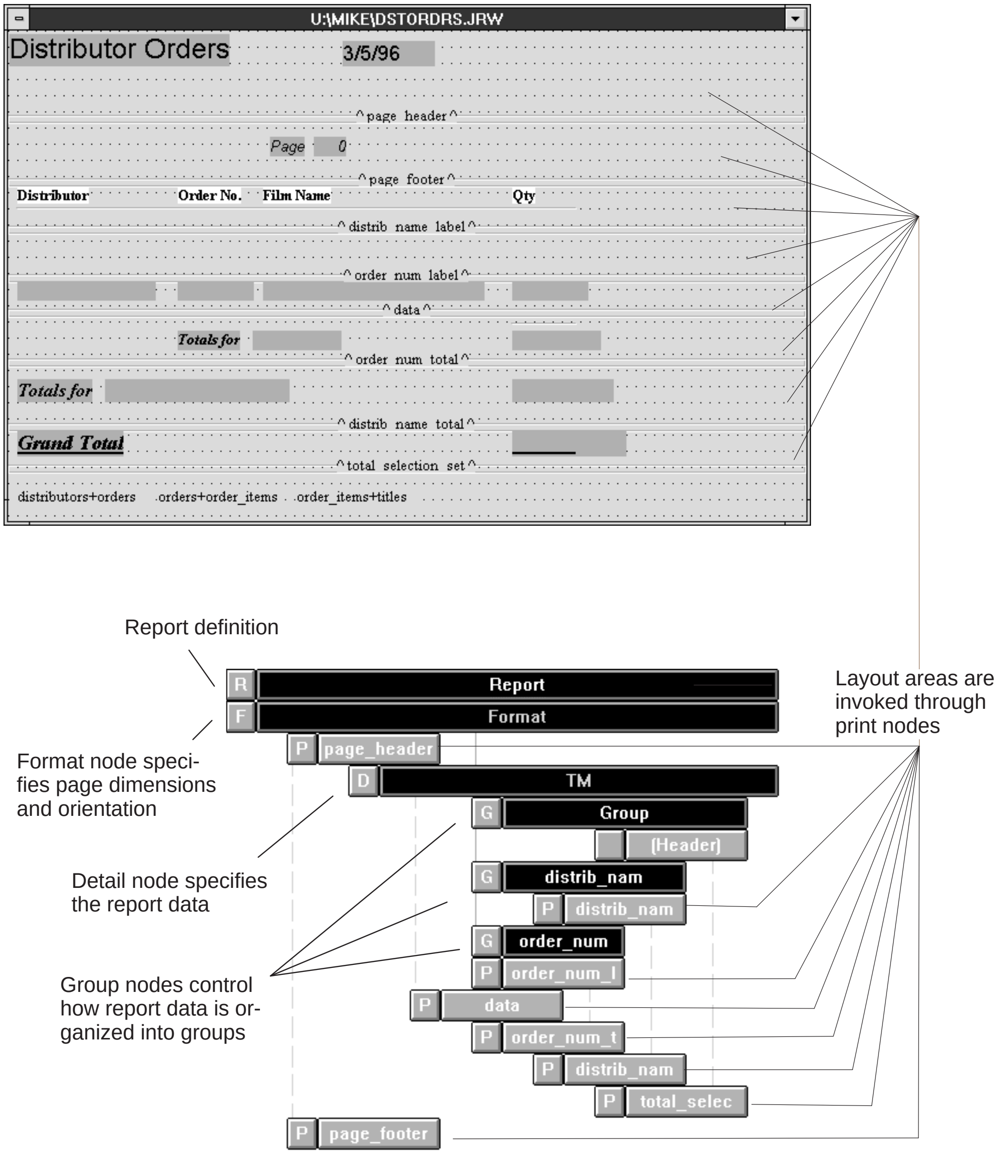
Distributor Orders 3/5/96

Distributor	Order No.	Film Name	Qty
Somerville	1003	Bull Durham	1
		Quiz Show	9
		Room With A View, A	1
	Totals for 1003		***11***
	1010	Client, The	10
		Fugitive, The	10
		Pulp Fiction	10
	Totals for 1010		***30***
Totals for Somerville			***41***

Distributor	Order No.	Film Name	Qty
Wellesley Hills	1002	Cinema Paradiso	1
		Pulp Fiction	8
	Totals for 1002		***9***
	1009	Au Revoir les Enfants	1
		Beauty and the Beast	3
		Fugitive, The	8
		Starman	1
	Totals for 1009		***13***
Totals for Wellesley Hills			***22***
Grand Total			***134***

Page 3

Figure 21 shows how execution of this report is viewed through the report structure window and its corresponding layout window. The two windows offer complementary views into the same report definition:



Report definition

Format node specifies page dimensions and orientation

Detail node specifies the report data

Group nodes control how report data is organized into groups

Layout areas are invoked through print nodes

Figure 21. Structure of report that outputs video outlet orders.

Figure 21 shows a report structure composed of these node types:

- The topmost unnamed node is a report node; it represents the desired report; all nodes attached to it comprise the report's definition. The report structure window can contain multiple reports, each beginning with a report node and representing separate report definitions. Of these, only the first report can be unnamed.
- A page format node follows the report node; it specifies page dimensions and orientation. Attached to it are two print nodes, which specify the layout areas to use for page header and footer output—`page_header` and `page_footer`, respectively. Because page headers and footers are above and below all other page output, these two nodes enclose the detail and group nodes.
- A detail node specifies the source of report data. In this case, the report relies upon JAM's transaction manager to use the table views and link widgets stored with this report to fetch the required data. Attached to this node is a single print node, which outputs each instance of the detail data.
- Three group nodes are attached to the detail node; these tell ReportWriter how to organize data in the report. Each group node has a print node in its footer section, which outputs subtotals for its group.

Node Classes and Types

Nodes belong to one of two classes—structure nodes and action nodes:

- Each structure node determines an aspect of the report definition—for example, a page format node sets page dimensions and orientation, while a detail node determines how the report gets its data.
- Action nodes are attached to a structure node and determine report output and processing for that structure node—for example, a print node that is attached to a group node's footer section specifies the layout area to print after a break occurs within that group; a call node specifies a function to call; and so on.

Table 1 lists the different node types and summarizes their purpose:

Table 1. Report structure node types.

Type	Purpose	More...
Structure nodes:		
Report	Defines a report.	p. 69
Page Format	Specifies page format and orientation.	p. 70

Type	Purpose	More...
Instance	Provides an all-purpose hook for one-time execution of the attached action nodes.	p. 89
Detail	Specifies the source of report data and actions to perform at each fetch.	p. 73
Group	Specifies how to group data and the processing to perform when a new group is generated.	p. 77
Action nodes:		
Print	Outputs a layout area.	p. 77
Call	Calls a function.	p. 88
Subreport	Invokes a subreport.	p. 84
End Page	Forces a page break and starts a new page.	p. 118

You can attach any number and type of action nodes to all structure nodes except a report node. Action nodes can be attached in any order.

Node Hierarchy

The order of a report’s structure nodes is summarized in this formula:

$$\text{Report}\left\{\text{Page Format}\left[\begin{array}{l}\text{Instance}\\ \text{Detail [Group]...}\end{array}\right]\ldots\right\}\ldots$$

A report structure conforms to these rules:

- The first node in a structure must be a report node.
- A report node must be followed by a page format node, whose properties set page dimensions and orientation. A report can have multiple page format nodes; this lets you change a report’s format at different stages of execution.
- After the page format node, a report can contain one or more instance and detail nodes in any order. Each detail node can have one or more group nodes. Typically, a report uses one set of data and has only one detail node.
- Each structure node except a report node can have one or more action nodes attached to it. Action nodes can be of any type and placed in any order.

The flexibility offered by report structures lets you create almost any flavor of report with as many levels of complexity as needed.

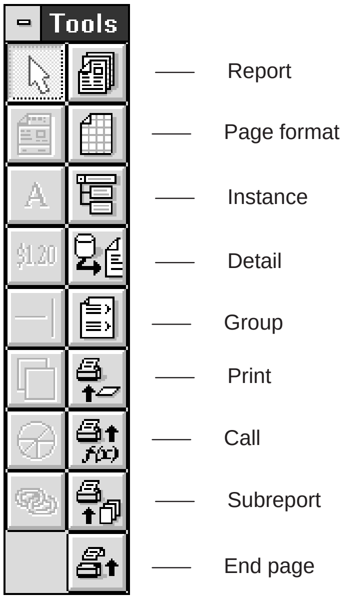
Editing the Report Structure

You can modify the report structure by adding, removing, and moving nodes. You can also control the amount of detail shown by collapsing and expanding portions of the structure's hierarchy.

To add a node:

1. Select the node type from either of the following:

    - Create menu — Select a node type from the list.
    - Tool box — Select the desired icon.



2. Select the node that is above the position desired for the new node.

To delete or remove a node:

Select a node. Do either of the following:

❍ Choose Edit⇒Cut or the Cut button from the toolbar.

This places the removed node and its attachments in the structure's clipboard. You can then paste the node wherever its placement is valid in the report structure. The cut node remains in the clipboard until it is overwritten by the next Cut or Copy operation.

❍ Choose Edit⇒Delete or press the Delete key.

The selected node and its attachments are removed from the structure and from memory; they can only be restored by choosing Undo.

To copy a node:

Select the node to copy and choose Edit⇒Copy or the Copy button from the toolbar.

This places a copy of the node and its attachments in the structure's clipboard. You can then paste the copy anywhere its placement is valid in the report structure. The copy remains in the clipboard until it is overwritten by the next Cut or Copy operation.

To paste a node:

1. Choose Edit⇒Paste or the Paste button from the toolbar.

2. Select the node that is above the position desired for the pasted node.

To move a node:

Select the node that you want to move. Do either of the following:

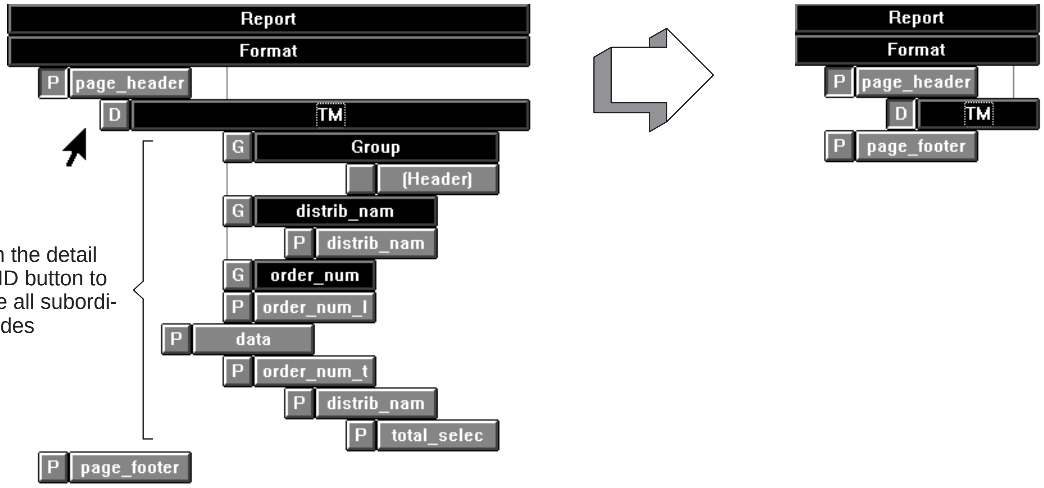
❍ From the Edit menu, choose Arrange Nodes⇒Move Up or Arrange Nodes⇒Move Down.

❍ Press the Move Up or Move Down key.

Note: *You can move a node up and down only within its current level in the report structure hierarchy. For example, a call node that is subordinate to an instance node can be moved above or below other action nodes that belong to the same instance node; it cannot be moved anywhere else in the report structure. To move a node elsewhere in the hierarchy, use Cut and Paste.*

To collapse and expand the structure view:

Click on a node's type ID button to toggle all attached nodes in and out of view; or from the Edit menu, choose Arrange⇒Expand or Arrange⇒Contract. Figure 22 shows how you can collapse and expand portions of the report structure.



Click on the detail node's ID button to collapse all subordinate nodes

Figure 22. Click on a node's type ID button to collapse and expand the view of its subordinate nodes.

You can also collapse or expand the entire structure. From the Edit menu, bring up the Arrange submenu and choose the desired option:

- Collapse All — Hides all nodes in the structure window except report nodes.
- Expand All — Restores view of all nodes in the structure window.

Editing Node Properties

Through the Properties window, you can edit the properties of any selected node. To bring the Properties window into focus for the current node, give focus to the node. An arrow marks the selection; you can bring the Properties window forward by double-clicking on this arrow.

Each node type has one or more properties that are unique to it; these are discussed in later sections. Nodes also share a common set of Identity properties; these are discussed below.

Name

A node's Name property identifies it for programmatic access. Named nodes can be accessed at runtime in order to change their properties. For example, naming a print node lets you access its `area` property and set it according to the runtime context:

```
if amount < 0
{
   detail_output->area = ″debit_entry″
}
else
{
   detail_output->area = ″credit_entry″
}
```

Report nodes must be named in order to be explicitly invoked—for example, by another report or from the command line.

Refer to page 299 in the *Editors Guide* for more information about widget naming conventions and requirements.

Inherit From

Defines the source of inheritance for this report—the name of the repository report followed by the name of the parent object in this format:

repository-entry`!`*widget-name*

A node can inherit any properties that are set in its source, such as Break On and Orientation. However, nodes cannot inherit their relationships to each other as specified by their relative order in the repository entry’s report structure.

For more information about inheritance, refer to page 63 in the *Editors Guide*.

Comments

By including a brief description of a node, you can save information about this node and its purpose:

1. Select the node.
2. Under Identity, select the Comments property. The Comments dialog box opens.
3. Enter or edit text through one of these actions:
   - Type the text directly in the text area.
   - Choose Editor to access your local editor as defined by setup variable `SMEDITOR` (page 17 in the *Configuration Guide*).
   - Choose Read File to read in an external file located on your system.
   - Choose Save File to save the comments to an external file.
4. Choose OK save the comments and return to the Properties window.

Viewing Node Links

You can review dependencies between a node and widgets in the layout window by selecting that node and choosing Report⇒Show Property Links. ReportWriter displays the Property Links dialog, which contains two types of entries:

- *prop-name* –> *widget-name* shows that this node's *prop-name* property is set to *widget-name*. For example, the links for a print node might include this entry:

  ```
  Area –> distrib_name_footer
  ```

  This entry shows that the print node's `Area` property specifies to output layout area `distrib_name_footer`.

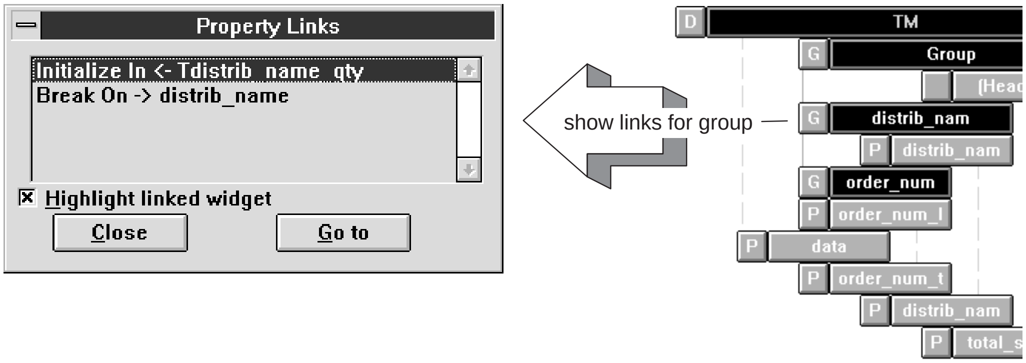
- *prop-name* <– *widget* shows that a widget in the layout window specifies this node in its *prop-name* property. If *widget* is an unnamed widget, it is identified as `Field #`*n*, where *n* is its field number. For example, the links for a detail node might include this entry:

  ```
  Update In <– Tdistrib_name_qty
  ```

  This entry shows that the detail node is referenced by total widget `Tdistrib_name_qty`'s Update In property.

  Note that group and detail nodes can be linked to a total or history widget's Initialize In and Update In properties even when those properties are blank. ReportWriter infers the default for these properties from the node in which the widget is output.

For example, this dialog shows the links for a group node:



Initialize In <- Tdistrib_name_qty
Break On -> distrib_name

Figure 23. Property Links dialog shows the widgets that are specified in a node's properties.

***Note:** The Property Links dialog only shows links to existing widgets; if a property specifies a widget or node that does not exist—either because it is not yet created or it is included at runtime—the dialog omits this link.*

Going to a Property Link

When you display the Property Links dialog, you can give focus to the widget or node specified in the selected entry by choosing Go To, or by double-clicking on the entry. If the widget is in the layout window, ReportWriter brings this window forward.

Defining a Report

A report file contains one or more reports. Each report begins with a report node, which can name the report and specify its parameters. This node and its descendants comprise a single report.

Named and Unnamed Reports

The report structure window can contain multiple reports. Of these, only the first can be unnamed. Unnamed reports can be invoked through the name of the report file. For example, this `runreport` command runs the first report in report file `expense_report`, without regard to its Name property:

```
runreport expense_report
```

Alternatively, the following `runreport` command runs report `summary_rpt` in report file `expense_reports`:

```
runreport expense_reports!summary_rpt
```

An unnamed report can serve only as a main report; other reports in the same report file are typically invoked as subreports by the main report and by each other. Named reports can also be invoked externally—as main reports, or as subreports by reports in other files.

Report Parameters

A report can define one or more parameters, which allows its caller to supply a corresponding number of arguments. This can be useful for setting a report’s properties at runtime. For example, an argument that sets the WHERE clause in a report’s SQL statement lets you modify database access each time you run the report. Or you can alternate different layout areas for the report’s title page, where the desired layout area is set through a command-line argument.

When you select the Parameters property, a dialog box opens. Enter the names of the parameters, one per line. You can enter the name of any dynamic output widget in the layout window or JPL global variable that is declared in the report file’s unnamed procedure. Each parameter so named receives an argument when the report is run. The order in which parameters are specified determines the order in which arguments must be passed to the report.

If the report receives fewer arguments than the the number of defined parameters, the values of the leftover parameters remain unchanged. If the report receives more arguments than the number of defined parameters, the extra arguments are ignored.

Setting Page Format

Page layout properties—the size and orientation of the page and footer placement—are set by the page format node. The layout areas for the page header and footer are output through two print nodes that are attached to the page format node’s header and footer sections, respectively.

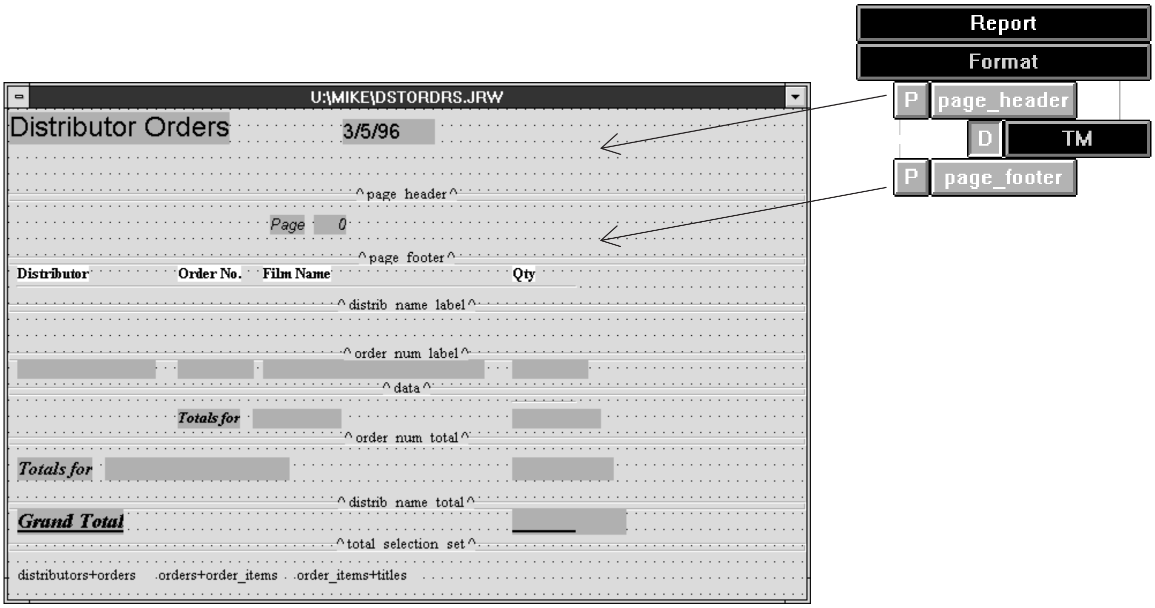


Figure 24. Specify page header and footer output through page format print nodes.

Page Format Properties

A page format node has three properties:

Floating Footer
Specifies whether the page footer's position is fixed at the bottom of the page, or allowed to float directly below the output of each page.

Orientation
Sets the report's page orientation to portrait or landscape. If you set this property to Default, ReportWriter uses the orientation that is set in the report invocation string (via the supplied option), the Page Setup dialog, or (on Windows) by the print driver.

Page Size
Specify the dimensions of the output area on report pages. If these properties are left blank, ReportWriter uses the page size that is set in the report invocation string (via the supplied option) or in the Page Setup dialog. You can select one of the predefined page sizes from the drop-down list, or specify your own page size in this format:

width `x` *height [unit-spec]*

Units of measurement (*unit-spec*) can be specified in inches (`in`), millimeters (`mm`), or characters (`c`—the default). For example, `7 x 9 in` specifies a page width of 7 inches and a length of 9 inches. `80 x 40` specifies a page that is 80 characters wide by 40 lines long.

If you specify dimensions in character (`c`) units, ReportWriter uses the average character size in the report's default font to calculate a character unit's width and height.

Resetting Page Format

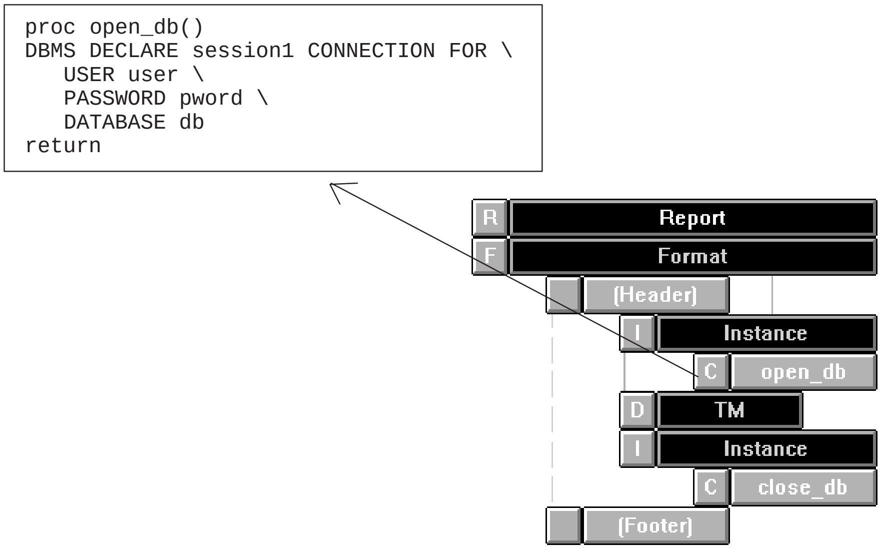
You can change the report's page format at any stage of report execution by inserting a new page format node in the report structure. The new node's properties supersede the previous page format property settings. These properties take effect on the first page to output after execution of the new page format node. To enforce new page format properties immediately, insert an end page node after the page format node.

Note: *Execution of a new page format node does not by itself force a page break. If the node is executed before the current page is full, ReportWriter continues to use the previous page properties through completion of that page.*

Connecting to a Database

If the report gets its data from a database, a database connection must be established before the report's detail node is executed. How the connection is made depends on the report is run. A stand-alone report should specify the desired connection in the report structure through a call node that executes before detail processing begins. Alternatively, if the report is run through a JAM application, the application can require users to specify the connection before it lets them run the report.

Figure 25 shows a typical stand-alone report, which sets the database connection through a call node attached to the the main report's page format node; this call node's function contains the `DBMS DECLARE CONNECTION` command. Required arguments for the connection are supplied to the report when it is invoked and made accessible to the procedure through report parameters `user`, `pword`, and `db`. The call node is attached to an instance node, so it is called only once:



```
proc open_db()
DBMS DECLARE session1 CONNECTION FOR \
   USER user \
   PASSWORD pword \
   DATABASE db
return
```

Figure 25. A stand-alone report connects to a database through a call node that executes before detail processing begins.

disconnecting from a database

If a report sets its own report connection, it should also close the connection before terminating. The report structure shown in Figure 25 closes its database connection through a call node that is called at the report's end. This call node also is attached to an instance node, so it is called only once.

Fetching Report Data

The detail node's Data Source property specifies how to fetch report data. Data can come from one of the following sources:

- One of the databases that JAM supports (Sybase, Oracle, Informix, and others). Wizard-created reports can access the data in any of these databases through transaction manager-generated SQL.
- A `send` bundle created in the JAM application that invokes the report.
- Your own function.

Using Database Data

Reports typically get their data from a database. ReportWriter uses one of JAM's database drivers to connect to the desired database and fetch report data. The layout window should contain a dynamic output widget for each database column whose data you wish to use in the report—either for output or for calculations. When the query is executed, the database driver pairs the report widgets to database columns and reads data into the report widgets accordingly.

If no widget corresponds to a fetched database column, the data in that column is ignored. No error is reported. For more information on how JAM pairs database columns to its own widgets, refer to page 224 in the *Application Development Guide*.

A report can fetch database data in one of three ways:

- Execution of `VIEW` by the transaction manager.
- A SQL query that you specify.
- Execution of a previously declared database cursor.

Transaction Manager

If you set Data Source to TM View, ReportWriter uses the transaction manager to generate the SQL statements required to fetch data. The transaction manager fetches report data through its `VIEW` and `CONTINUE` commands. When `VIEW` executes, the transaction manager fetches data for any widget belonging to a table view that is included in the transaction. `CONTINUE` is repeatedly executed until all data is fetched for the report.

Wizard-created reports rely on the transaction manager to fetch data. The report wizard automatically creates the widgets, links, and table views that might be needed.

If you create a report manually and want the transaction manager to supply the report's data, copy some or all of the layout widgets from a repository entry that is imported from a database. Using repository components automatically sets the properties needed by the transaction manager. Specifically, copy these objects:

- ❍ All widgets to output fetched data. Copy these to the detail's layout area. (ReportWriter automatically changes the widgets to dynamic output widgets.)
- ❍ Link widgets that describe the relationship between required table views, if the query includes more than one database table. Copy these links to the unnamed area of the layout window. The transaction manager uses these links to determine which table views are a part of the current transaction. You might also need to change the Type or Parent/Child property settings of the copied link widgets. Server links are always suitable for use in reports.

To fetch data using the transaction manager:

1. Select a detail node.
2. Under Identity, set the Data Source property to TM View. The Root subproperty is displayed.
3. Set the Root subproperty to the name of the root table view. The root table view forms the basis of the transaction manager transaction and is the parent of any links. If this property is blank, ReportWriter gets the root table view from one of these two sources (in this order):
   - The report file's Root property (accessible when the layout window has focus).
   - The link widgets' Parent/Child properties.

For more information about table views, refer to page 354 in the *Application Development Guide*. For general information about the transaction manager, refer to page 309.

SQL SELECT Statement

Set Data Source to SQL Query if you want to use your own SQL to fetch report data. The SQL statement must conform to the syntax of the database in use.

In order to set Data Source to SQL Query, the report must contain a dynamic output widget for each database column in the query, and the widget and column names should be identical. If the database column and the JAM widget have different names, your SQL statement must map between the two; otherwise, you can write your own SQL statement only by setting Data Source to Predefined Cursor (refer to page 75).

To fetch data using a SQL SELECT statement:

1. Select a detail node.

2. Under Identity, set the Data Source property to SQL Query. Related subproperties are displayed.

3. In the SQL Statement subproperty, enter the SQL string.

4. Set the Default Cursor subproperty:

    - Yes — Execute the statement on the default select cursor.

    - No — Open a dedicated cursor to execute the statement so that any database operation on the default cursor is not interrupted.

For example, this SQL statement generates an order list for the specified customer:

```
SELECT orders.order_num, order_date, title_id, price, qty
   FROM orders, order_num
   WHERE distrib_id = :+distrib_id
   ORDER BY order_num, title_id
```

ReportWriter executes this statement, then repeatedly executes `DBMS CONTINUE` until all the data is fetched into the report.

For Sybase, you can also fetch report data using the `EXEC` command to execute a stored procedure. For example, this statement executes procedure `fetch_order`, passing the value in `order_num` as an argument to that procedure.

```
EXEC fetch_order :+order_num
```

For more information about how JAM's database drivers fetch data from a database, refer to Chapters 14 and 15 in the *Application Development Guide*.

Named Cursor

Named cursors are useful for fetching data to subreports that are invoked repeatedly. The cursor is declared just once—typically, at startup of the application or the parent report—then executed each time a subreport is invoked. Because the statement is already parsed, ReportWriter can execute it faster than an equivalent SQL query.

For this option, a cursor is declared and associated with the SQL `SELECT` statement that fetches the report data. For more information about declaring cursors in JAM, refer to Chapter 13 in the *Application Development Guide*.

To fetch data using a named cursor:

1. Declare the cursor before detail output begins, either in the report itself or in the calling application. To declare the cursor within the report:

    - Write a JPL procedure that uses `DBMS DECLARE CURSOR` to declare the cursor with bind parameters, if any. If any column and widget names do not match, the procedure must also explicitly map the relationship between all non-matching pairs with `DBMS ALIAS`.

- In the report's structure, insert an instance node that executes before the detail node, and attach a call node to it. This call node's function calls the desired JPL procedure.

2. Select a detail node.

3. Set the Identity⇒Data Source property to Predefined Cursor. The Cursor and Using subproperty is displayed.

4. Set Cursor and Using to the name of the cursor, followed by the JAM variables corresponding to bind parameters, if any. At runtime, ReportWriter executes `DBMS EXECUTE` and appends the bind variable names to the command's `USING` clause. `DBMS CONTINUE` is repeatedly executed until all data is fetched for the report.

⚠ ReportWriter reserves the following naming convention for its own use:

`_RWC_`*xxxx*

xxxx is a four-digit hexadecimal number that uniquely identifies the cursor.

Fetching Bundle Data

JPL's `receive` command can process bundle data transmitted by a `send` command previously executed in the calling JAM application. When the detail node first executes, ReportWriter fetches as much bundle data as the receiving widgets require; each successive fetch continues where the last fetch left off, until no more bundle data remains to be read. This method is appropriate for transferring limited amounts of data from a screen to a report.

To fetch bundle data:

1. Select a detail node.

2. Under Identity, set the Data Source property to Receive Bundle. Related subproperties are displayed.

3. Set the Receive Widgets subproperty to the names of the widgets to receive the data. The order of the widget names must match the order in which data is specified in the `send` command.

4. Set the Bundle Name subproperty to the bundle name specified in the `send` command. If no bundle name was specified, leave this property blank; ReportWriter uses the unnamed bundle.

For more information on `send` and `receive`, refer to page 191 in the *Application Development Guide*.

Using a Custom Function

If you set the detail node's Data Source property to TM View or SQL Query, ReportWriter uses JAM's database interface to fetch data. If your report requires data from flat files, wire services, or other sources not directly supported by JAM, you might need to set Data Source to Custom Function and specify your own function to fetch data. This function must fetch the desired data and place the data into the target widgets in layout window.

You can write a custom function in either JPL or C. If you call a C function, it must be installed on the prototyped function list and linked into ReportWriter. For information on installing prototyped functions, refer to page 121 in the *Application Development Guide*.

For information about ReportWriter return codes, refer to page 109.

To fetch data using a JPL procedure or C function:

1. Select a detail node.
2. Under Identity, set the Data Source property to Custom Function. The Function Call subproperty is displayed.
3. In the Function Call subproperty, enter the function or procedure name.

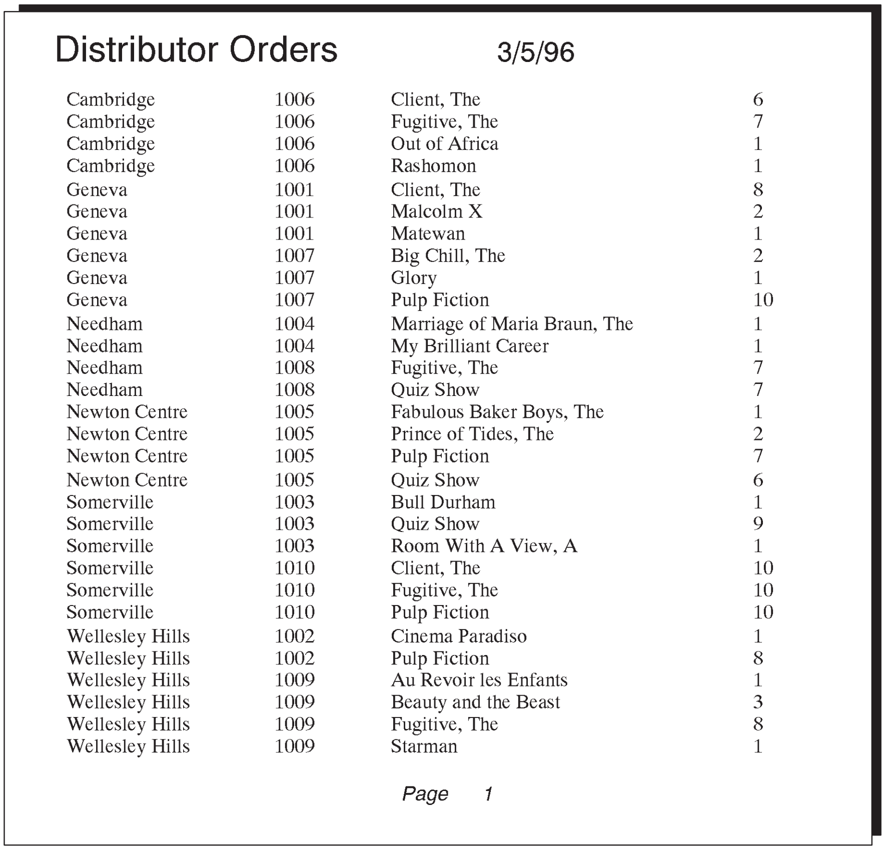
Outputting Detail Data

The print nodes that you attach to a detail node are executed after every fetch; the specified layout determines the format and content of detail data. You can avoid outputting detail data by omitting this print node—for example, to produce reports that contain summary data only (refer to page 83). For information about print node composition properties, refer to Chapter 6.

Creating Groups

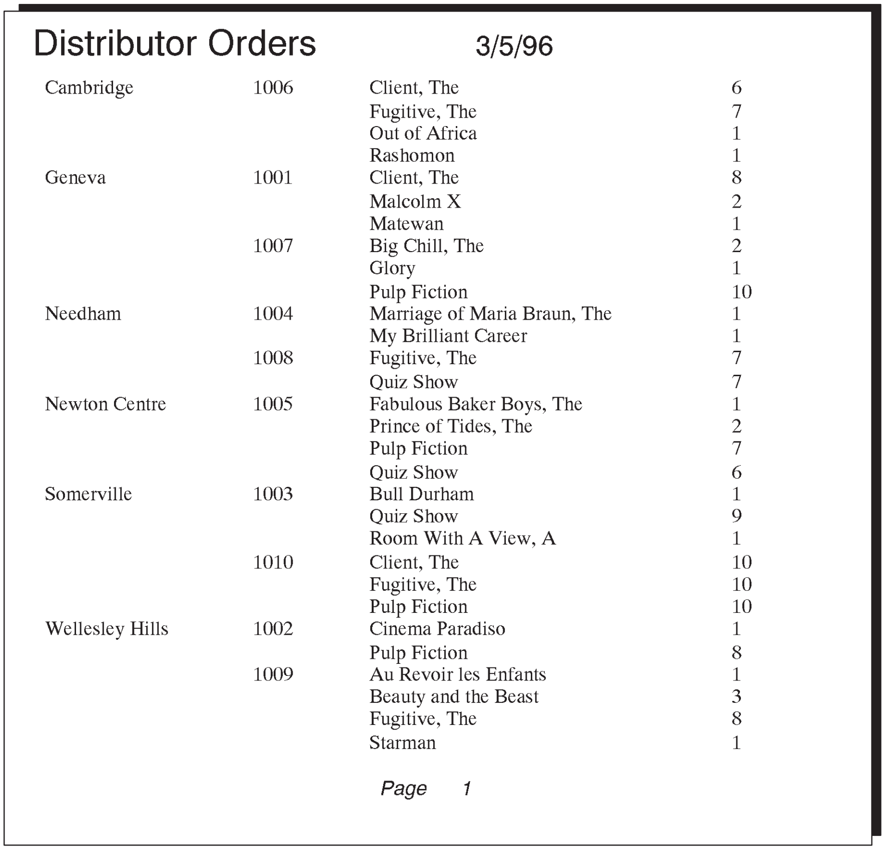
Reports can visually separate groups of detail output and display introductory and summary data for each group. Groups can be nested; their summary data, such as totals and averages, can be used in graphs and displayed as bar or pie charts.

Report data can be grouped according to the values of any sort widget. If report data is sorted on more than one widget, you can group data at multiple levels. For example, the following report is sorted on distributor IDs and order numbers:



```
Distributor Orders              3/5/96

Cambridge         1006     Client, The                     6
Cambridge         1006     Fugitive, The                   7
Cambridge         1006     Out of Africa                   1
Cambridge         1006     Rashomon                        1
Geneva            1001     Client, The                     8
Geneva            1001     Malcolm X                       2
Geneva            1001     Matewan                         1
Geneva            1007     Big Chill, The                  2
Geneva            1007     Glory                           1
Geneva            1007     Pulp Fiction                    10
Needham           1004     Marriage of Maria Braun, The    1
Needham           1004     My Brilliant Career             1
Needham           1008     Fugitive, The                   7
Needham           1008     Quiz Show                       7
Newton Centre     1005     Fabulous Baker Boys, The        1
Newton Centre     1005     Prince of Tides, The            2
Newton Centre     1005     Pulp Fiction                    7
Newton Centre     1005     Quiz Show                       6
Somerville        1003     Bull Durham                     1
Somerville        1003     Quiz Show                       9
Somerville        1003     Room With A View, A             1
Somerville        1010     Client, The                     10
Somerville        1010     Fugitive, The                   10
Somerville        1010     Pulp Fiction                    10
Wellesley Hills   1002     Cinema Paradiso                 1
Wellesley Hills   1002     Pulp Fiction                    8
Wellesley Hills   1009     Au Revoir les Enfants           1
Wellesley Hills   1009     Beauty and the Beast            3
Wellesley Hills   1009     Fugitive, The                   8
Wellesley Hills   1009     Starman                         1

                           Page    1
```

Repeating values under the `distrib_name` and `order_num` columns show how data in this report might be grouped. The next report eliminates repeating values to clarify groupings:



```
Distributor Orders              3/5/96
Cambridge          1006    Client, The                        6
                           Fugitive, The                      7
                           Out of Africa                      1
                           Rashomon                           1
Geneva             1001    Client, The                        8
                           Malcolm X                          2
                           Matewan                            1
                   1007    Big Chill, The                     2
                           Glory                              1
                           Pulp Fiction                       10
Needham            1004    Marriage of Maria Braun, The       1
                           My Brilliant Career                1
                   1008    Fugitive, The                      7
                           Quiz Show                          7
Newton Centre      1005    Fabulous Baker Boys, The           1
                           Prince of Tides, The               2
                           Pulp Fiction                       7
                           Quiz Show                          6
Somerville         1003    Bull Durham                        1
                           Quiz Show                          9
                           Room With A View, A                1
                   1010    Client, The                        10
                           Fugitive, The                      10
                           Pulp Fiction                       10
Wellesley Hills    1002    Cinema Paradiso                    1
                           Pulp Fiction                       8
                   1009    Au Revoir les Enfants              1
                           Beauty and the Beast               3
                           Fugitive, The                      8
                           Starman                            1

                            Page    1
```

Sorting Data

Data should be grouped in the same order in which it is sorted. For example, the root table view for the previous report specifies its sort widgets as `distrib_name`, `order_num`, and `name`, in that order. Given this sort order, data can be grouped first by distributor IDs, then by order numbers, and finally by film names.

If you rely on the transaction manager to fetch data, you can sort report data through the root table's Sort Widgets property. For information about this property, refer to page 352 in the *Editors Guide*. A SELECT statement can also sort widgets as desired. If the report fetches data from a `send`/`receive` bundle or through your own function, make sure that the data you supply is appropriately sorted.

Defining Groups

A report structure's group nodes and their respective Break On properties determine how data is grouped. The Break On property specifies a named dynamic output widget, or *break field*, whose values define a single level of data grouping. Each grouping can contain data from one or more detail fetches; the group grows as long as the break field's value remains unchanged. When the break field's value changes, ReportWriter ends the previous grouping and begins a new one.

On each fetch, before outputting the current detail, ReportWriter checks whether this widget's current value differs from the previous fetch; if it does, ReportWriter begins break group processing with these actions:

1. Executes the footer actions for this group and all lower-level groups, beginning with the lowest-level group.

2. Executes the header actions for this group and all lower-level groups, beginning with this group.

3. Resumes detail processing.

Insert group nodes below the detail node in the desired order. For example, the report structure in Figure 26 yields the output shown earlier:

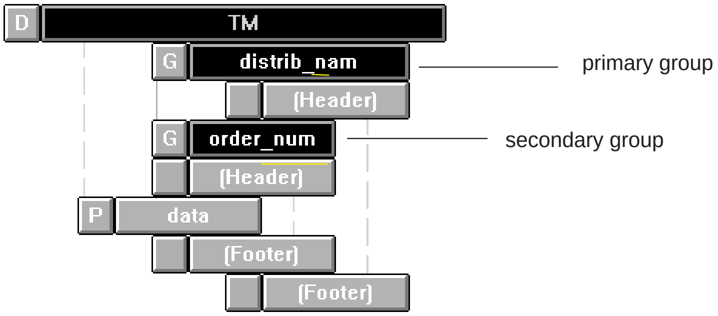


Figure 26. Group nodes in a report structure specify how to group report data.

The first group node's Break On property is set to `distrib_name`; for the second, it is set to `order_num`. In order to suppress repeating values within sort columns, both group nodes have their Print Break Value property set to First and Page Top.

A report that groups data can also generate totals and other calculated values from numeric columns; for more information, refer to Chapter 5.

Controlling Break Field Output

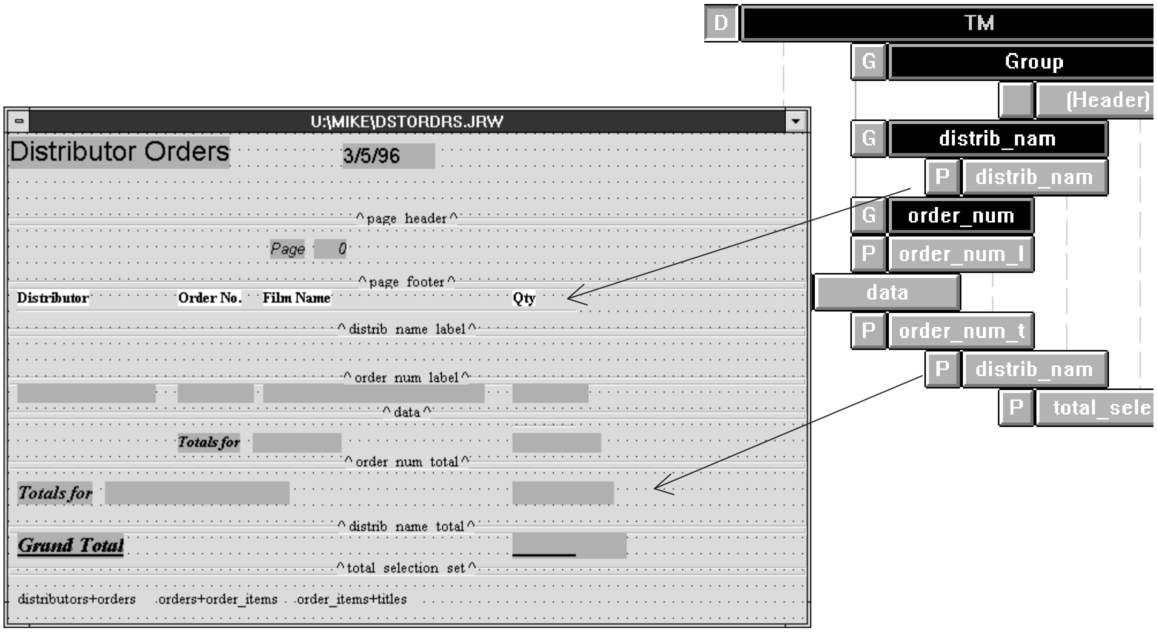
By default, ReportWriter outputs the values in all dynamic output widgets with each detail fetch. You can control frequency of output in a group's break field by setting the group's Print Break Value to one of these values:

- Each Use — Output the break field’s values on each detail line of the group.
- First Use Only — Output the break field’s value only in the first detail line of the group.
- First and Page Top (default) — Output the break field’s value in the first detail line of the group; if the group output spans multiple pages, reproduce the break field’s value in the first detail line of each new page.

Outputting Group Headers and Footers

When you create a group node, ReportWriter automatically inserts header and footer sections to indicate all processing and output that takes place before and after a group’s detail output. Group’s header and footer sections can contain print nodes that output text at the beginning and end of the group’s detail output. Typical header output in a column report includes column titles and the break field’s current value; footer output often contains group totals for one or more columns.

For example, in the following report structure, group node `distrib_name`’s header and footer sections each contain a print node that specifies a layout area, `distrib_name_label` and `distrib_name_total`, respectively. The footer layout area includes dynamic output widget `Tdistrib_name_qty`, which totals values in column `qty` for the current group:



When this report runs, ReportWriter outputs the two layout areas at the beginning and end of each group of video distributors. Total widget `Tdistrib_name_qty` contains the subtotal of `qty` values for each group. For more information about totaling group values, refer to page 96.

Several group node properties control how ReportWriter outputs its headers and footers:

Running Header

If a group's output spans one or more page breaks, you can ensure that header output is repeated at the top of each new page by setting this property to Yes. The default for this property is No.

Note: *Setting this property has no effect on printing group headers and footers when a break occurs in the group.*

Local Header Only

This property determines whether to suppress header output for a group when the next-higher group also outputs its own header:

- Yes — Output this group's header only if no break occurs in the next-higher group; if this group and the next one above it break simultaneously, ReportWriter outputs only the higher group's header.

  For example, you might set this property to Yes when column labels in the lower group's header are a subset of those in the group above it. In this case, it is redundant to output headers from both groups.

- No (default) — Output this group's header together with the higher group's header.

One-Detail Footer

Determines whether to suppress footer output if a group contains only one line of detail or none:

- Yes (default) — Always output the group footer regardless of the detail content. Be sure to set this property to Yes for summary-only reports.
- No — Output the group footer only when the group contains more than one detail line; suppress footer output for groups that contain only one or no detail line. If footer output is suppressed, ReportWriter uses the value in the Custom Space subproperty to determine how much space to substitute, if any.

Custom Space

If footer output is suppressed because the group contains only one or no detail, this property sets the amount of blank space to output in place of the footer. If this property is left blank (the default), ReportWriter inserts an amount of white space that is equivalent to the white space output by the footer.

You can set the amount of space in inches (`in`), millimeters (`mm`), or characters (`c`—the default). For example, `1in` inserts 1 inch of blank space after a one-detail group, while `2` specifies to insert the height of 2 characters. If the value is in character (`c`) units, ReportWriter uses the average character size in the report's default font to calculate a character unit's height.

Outputting Summary Data Only

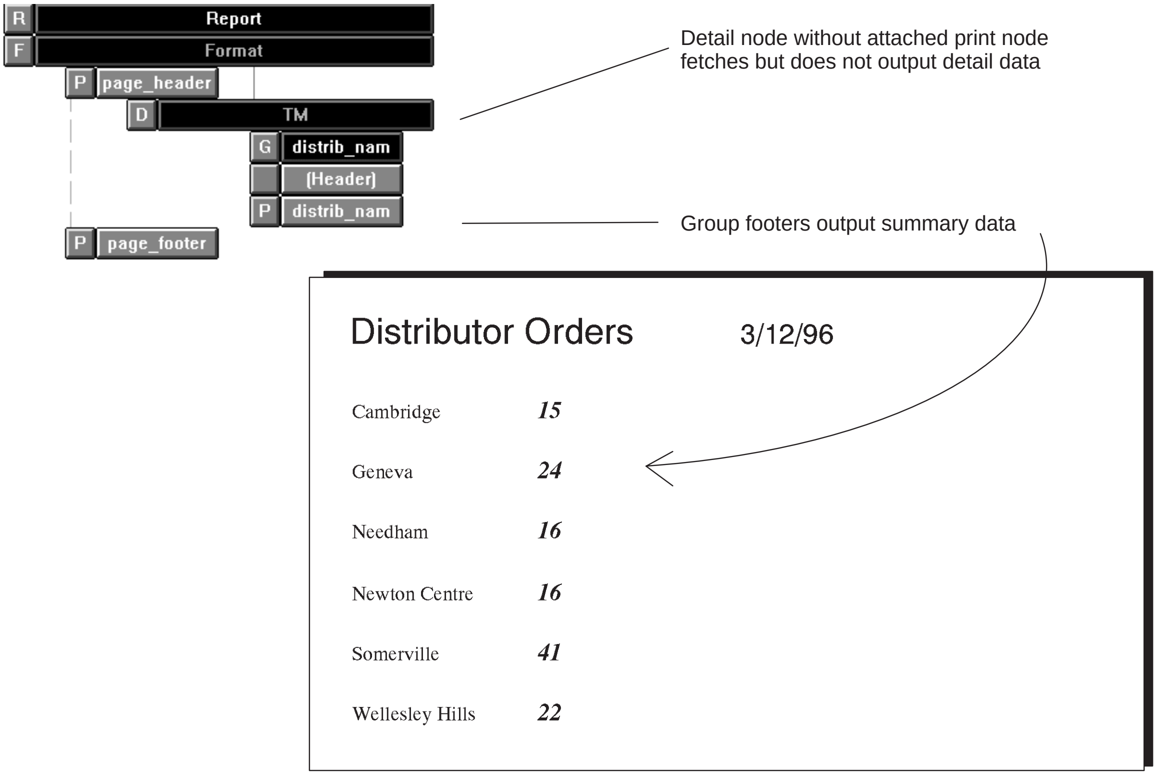
A report that outputs summary data in group footers can easily be modified to omit detail data. To create a summary-only report:

1. Remove or omit from the report structure the detail's print node. Usually, it makes sense also to remove or omit any group header print nodes whose output label the detail columns.

2. Set the One-Detail Footer property to Yes for all groups that output summary data.

Note: *Whether or not the report outputs detail data, the layout window must contain dynamic output widgets that receive the fetched data, either in an unused layout area or in the layout window's unnamed area.*

Figure 27 shows the structure of a summary-only report and its output:



Detail node without attached print node fetches but does not output detail data

Group footers output summary data

Distributor Orders 3/12/96

Cambridge	*15*
Geneva	*24*
Needham	*16*
Newton Centre	*16*
Somerville	*41*
Wellesley Hills	*22*

Figure 27. Summary-only report structure and its output.

Grouping All Report Data

In order to process data for the entire report—for example, to calculate grand totals—ReportWriter can view all detail output as a single group. To give this group primacy over all other groups, its group node must be topmost in the report structure. No breaks occur in this group, so its Break On property is left empty. For more information on generating grand totals, refer to page 96.

Invoking Subreports

ReportWriter lets you invoke a subreport at any stage of report execution. In general, you can use any named report as a subreport. How you use a report can determine where you store it. If a report is intended to be invoked as a subreport by different reports, it probably makes sense to define it in a separate file where it can be accessed equally by all its users. Alternatively, if a subreport is invoked only by one report, you can define both in the same file.

To invoke a subreport:

1. Create a subreport node in the report structure. Attach the subreport node at the point in the structure where you want its output to appear.
2. Identify the desired report in the subreport node’s Report Invocation property. The name that you enter must be defined in a report node’s Name property. The report to invoke can be in one of three locations:
   - The current report file — A named report in the current file can be invoked by any other report in the same file.
   - An included report file — If the report file’s Inclusions property specifies other files, any named report from the included files can be invoked as a subreport.
   - Another report file — To invoke an *external report*—that is, a report defined in a non-included file—qualify the report name with the name of the file that stores it:

     report-filename`!`*report-name*

For more information about invoking external and included subreports, refer to page 90.

Setting Subreport Composition

Three Composition properties control how ReportWriter outputs the contents of a subreport:

Indent

Shifts right all subreport output from the caller’s leftmost margin by the amount of space specified. Relative spacing within the subreport’s output remains unchanged. Specify the amount of space to indent in this format:

indent-space [unit-spec]

Units of measurement (*unit-spec*) can be specified in inches (`in`), millimeters (`mm`), or characters (`c`—the default). If you specify dimensions in character (`c`) units, ReportWriter uses the average character size in the report’s default font to calculate a character unit’s width.

Keep on Page

Determines whether subreports are split across pages. If you set this property to Yes, ReportWriter keeps all output of the specified subreport together and unbroken. If the subreport can fit in the space remaining on the current page, it is output immediately; otherwise, ReportWriter starts a new page.

If Keep on Page is set to No (the default), subreport output begins on the current page and continues on the following pages if needed.

Reserve Space

Sets the amount of space that the subreport occupies. This property must be set for subreports invoked from page footers; it is optional otherwise. This property is especially useful for printing on forms with a fixed-length space for the subreport data.

The default value is blank, which specifies to use only the amount of space required to output the entire report. You can set the amount of reserved space in inches (`in`), millimeters (`mm`), or characters (`c`—the default). For example, `5in` reserves 5 inches to output the report, while `40` specifies to reserve 40 characters. If the value is in character (`c`) units, ReportWriter uses the average character size in the report's default font to calculate a character unit's height.

If Reserve Space is set to a value, ReportWriter assumes that the subreport occupies the stated area and computes page breaks accordingly. If the subreport contains less area than specified, ReportWriter appends blank output to achieve the required size. If the subreport contains more than the specified number of lines, the output is truncated and an appropriate warning message is issued.

If Reserve Space is blank, ReportWriter consolidates a subreport's leading and trailing blank lines with the adjacent output. Refer to page 126 for an explanation of blank line consolidation.

Note: *Use Reserve Space only for page footer subreports and fixed-length subreports. To keep a subreport from being split across pages, set Keep on Page to Yes.*

Using Caller Settings

Two properties, Use Caller Groups and Use Caller Format, let you decide whether the subreport should use the calling report's group and page format node settings.

Note: *A subreport that lacks its own group nodes always uses the caller's settings.*

Use Caller Groups

Controls whether a subreport uses the caller's groups or its own:

- No (default) — The subreport begins with no groups in effect; the first group node encountered in the subreport begins a new group hierarchy.
- Yes — Groups set in the calling report remain in effect for the subreport; groups specified in the subreport are added to the existing hierarchy. The subreport also inherits the parent report's current break level to prevent duplication of break headers in the subreport.

Execution of a detail node normally forces an initial break for all groups. If a subreport has a detail node and detail processing is already underway in the calling report, you can avoid redundant break processing by setting Use Caller Groups for the subreport node to Yes. The subreport inherits the caller's break hierarchy and the current break level. When the first fetch of a detail node is executed in the subreport, initial breaks are forced only for break levels subordinate to the calling report's current level.

The report structure location of a subreport node that uses its caller's groups determines which group levels are affected:

Invoked from:	Affected group levels in subreport:
detail node	None.
group header/ footer	Lower levels only.
page format/ instance node	The subreport inherits the break level current in the parent when the calling subreport node is processed. The subreport's detail node forces initial breaks at all lower levels. If no break level is active, the detail statement forces breaks at all levels.

Use Caller Format

Controls whether a subreport uses the caller's page format node or its own:

- No (default) — The subreport uses the property settings of its own page format node—Page Size, Orientation, and Floating Footer; it also uses the actions attached to its own page header and footer sections. If any page format properties in the subreport are left empty, the subreport uses the caller's settings.
- Yes — ReportWriter uses the caller's page format node and ignores the one in the subreport.

When User Caller Format is set to No, ReportWriter processes page footer actions differently for internal and external subreports:

- If the subreport is internal—that is, stored in the caller’s report file or included at runtime—ReportWriter executes the subreport’s page footer actions on the current page.
- If the subreport is external, ReportWriter continues to execute the caller’s page footer actions for the current page; the subreport’s page headers and footers execute only on the first new page that the subreport itself generates. When control returns to the caller, ReportWriter restores its page header and footer actions only when the current page ends.

To ensure that ReportWriter processes an external subreport’s page header and footer actions immediately after it is invoked, force a page break before the subreport starts. To restore its caller’s page header and footer actions right after the subreport ends, force a page break before the caller resumes control.

Note: *A subreport that is invoked from a report’s page header or footer always uses the primary report’s Page Size and Orientation properties. The subreport’s header and footer sections and their actions are ignored.*

Calling Functions

Insert a call node wherever you want to execute a function written in JPL or C. A call node specifies the desired function through its Function Call property. A call node can invoke a JPL procedure or installed C function. For a description of specific uses of call nodes, refer to page 105.

The string that you enter in the Function Call property must use the syntax of JPL’s `call` command (page 47 in the *Language Reference*). For example, a call node can invoke function `myproc` and pass string arguments `hello` and `world` by setting its Function Call property to this value:

```
myproc( ”hello”, ”world” )
```

You can call any JPL procedure that is in the current report file, an included report file, or a public file—that is, a file whose procedures are made accessible via the `public` command (refer to page 68 in the *Language Reference*).

You can also call a C function if it is installed on the prototyped function list and linked into ReportWriter. For information on installing prototyped functions, refer to page 121 in the *Application Development Guide*.

To create or edit JPL procedures in a report file:

1. Give focus to the report’s layout window.

2. Under Inclusions, select the JPL Procedures property. The JPL Program Text dialog box opens.

3. Enter or edit the JPL procedures with one of these methods:

   - Type the code directly in the JPL Program Text window.

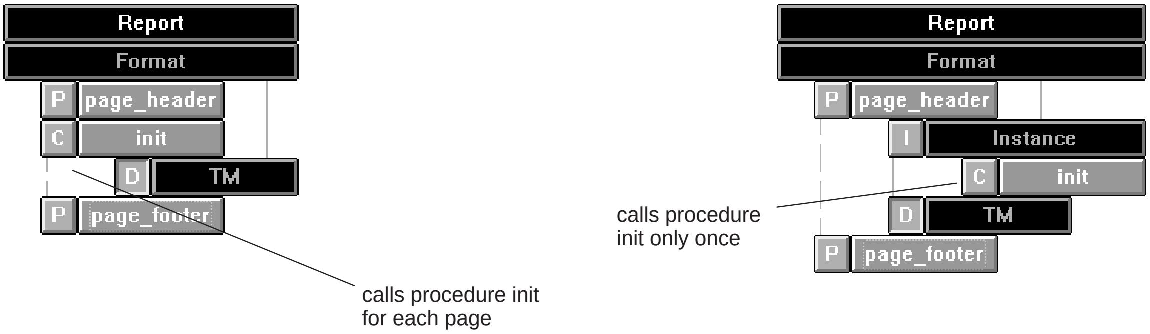
   - Choose the Editor button to invoke your local text editor.

   - Read in an existing file by choosing the Read File button. The Select File dialog box opens where you can identify the file.

4. Choose OK to accept your changes and exit the window. Or discard your changes by choosing Cancel.

Processing One-Time Events

An instance node provides an all-purpose hook for one-time execution of the attached action nodes. For example, compare the page format nodes shown in Figure 28. The header sections of both contain a call node that calls function `init`. However, in one case, the call node is attached directly to the page format node; at runtime, the ReportWriter calls `init` before each page is output, including the first one. In the second, the call node is attached to an instance node; in this case, ReportWriter calls `init` only once, before the first page is created:



calls procedure init for each page

calls procedure init only once

Figure 28. An instance node's actions are executed only once.

You can attach any number and type of action nodes to an instance node; ReportWriter performs these actions in order of appearance. Instance nodes are typically used to perform these tasks:

- Invoke initialization and clean-up procedures such as opening and closing a database connection (page 72).
- Produce a title and trailer pages (page 118).

Building Modular Reports

You can build a report whose components are partially or entirely derived from other report files. This is particularly useful for objects that have the potential for being reused by different reports—for example, layouts for title and trailing pages or inserted reports.

A report can access the contents of another report file in two ways:

- A subreport node can invoke a report that is defined in another file by specifying the file name and the name of the report—for example, `title.jrw!main`.
- A report can specify report files to be included through its Report Files property. All report definitions, layout areas, and widgets in the included files are accessible to the entire report.

Invoking External Report Files

A report can invoke a subreport that is stored in another file; the subreport node's Report Invocation property names the desired report with this syntax:

report-filename `!`*report-name*

ReportWriter looks for the specified file in the following locations in this order:

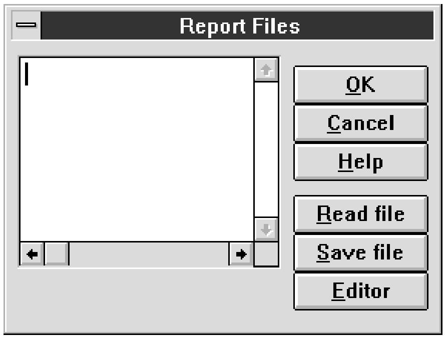
1. The application's memory-resident list; Refer to page 522 in the *Application Development Guide* for more information about memory-resident files.
2. All open libraries.
3. On disk in the current directory.
4. Along the path supplied to `sm_initcrt`.
5. Along all paths in the setup variable `SMPATH`. If any path exceeds 80 characters, it is skipped.

Including Report Files

A report file can specify to include one or more other files. At runtime, Report-Writer views the file and its inclusions as a single virtual file. Consequently, reports in one file can reference components in any of the other files. For example, if report file `x.jrw` specifies to include `y.jrw`, at runtime ReportWriter views the two files as one. Thus, subreport nodes in `x.jrw` can specify reports in `y.jrw`; similarly, print nodes in `y.jrw` can name layout areas that are in `x.jrw`.

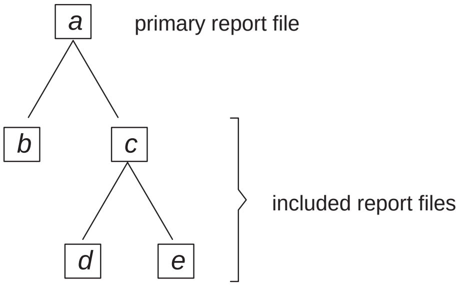
To include report files:

1. Give focus to the layout window of the primary report file.
2. Under Inclusions, select the Report Files property. ReportWriter displays the Report Files dialog box:



3. In the dialog box, enter the names of the report files you want to include. Each inclusion must be on a separate line. You can either type in the names directly or read in the contents of another file by pressing Read File.

An included file can itself specify to include other files. For example, the following diagram illustrates a possible inclusions structure, where file `a` includes `b` and `c`, and `c` includes files `d` and `e`:



resolution of name ambiguity

All names that might be cross-referenced among the primary file and its inclusions—for example, layout areas, reports, and nodes—should be unique; if redundant names occur, ReportWriter resolves the ambiguity by using the first component that it finds among the child report files. For example, if files `d`, `c` and `a` all contain layout areas named `page_header`, ReportWriter uses the layout area in `d`.

⚠ Multiple nesting levels contain the potential for circular references. A file should never include one of its predecessors.

Included Versus External Files

ReportWriter opens and closes external subreports as needed at runtime, so you can invoke as many subreports from the main report as desired. The maximum number of inclusions for a single report is limited to 254 lines in the primary and included reports.

5

Calculating Report Data

At runtime, ReportWriter can automatically capture data as it is output and use it to calculate totals. It can also copy output to other locations in the report; or use an array to collect a series of output values. You can also write your own functions to process output—for example, to calculate summary data such as averages, maximums, and minimums, or to control report execution by changing node properties at runtime.

Outputting Derived Values

A dynamic output widget's Value property controls how it gets its output value. You can set this property so that a widget gets its value from previous report output—for example, totals of column values within a given group. You can reset the widget's Value property to one of these options:

- Total — Get the total of all output in a widget. Totaled values are taken from a segment of report output—for example, all values in a given group.

- Copy — Get the same value as another dynamic output widget. Use this setting to repeat pertinent information—for example, the current value in a group's break field alongside subtotals for that group.

- History — Collect output from a widget. Values are collected from a specified segment of output—for example, all values in a group. You can use this data to supply the data series required by a graph widget.
- Page Number — Get the current page number, typically for output in page headers or footers.

Unless set to Page Number, the Value property has one or more subproperties:

Value Source

Specifies the dynamic output widget whose values you want to total, copy, or collect. The property's combo box lists all named dynamic output widgets in the current layout window. You can choose or type in one of these; or you can enter the name of a widget to be created later.

Initialize In

Available only when the widget's Value property is set to History or Total, this subproperty is set to the name of a group or detail node. ReportWriter initializes the widget's contents each time the specified node executes. The property's combo box lists all named group and detail nodes in the structure window. You can choose or type in one of these; or you can enter the name of a node to be created later.

Usually, you should leave this property blank. ReportWriter sets it to the same group node that outputs the widget's layout area.

Set this property only in two cases:

- The widget is not output—either because it is in an unused layout area or in the layout window's unnamed area.
- The widget's layout area is specified for the print node at runtime.

Update In

Available only when the widget's Value property is set to History or Total, this subproperty is set to the name of a group or detail node. ReportWriter updates the widget's contents each time the specified node executes. The property's combo box lists all named group and detail nodes in the structure window. You can choose or type in one of these; or you can enter the name of a node to be created later.

Usually, you should leave this property blank. ReportWriter updates the widget as follows:

- For total widgets, ReportWriter updates the widget's cumulative value from each detail output.

- For history widgets, it updates the collected values from the group immediately below the group in which the widget is initialized. If no lower group node exists, it updates the widget's collected values with each detail.

Set this property only in two cases: the widget is not output—either because it is in an unused layout area or in the layout window's unnamed area; or the widget's layout area is specified for the print node at runtime.

Context

If Context remains set to Default, ReportWriter uses the value that is consistent with the context of the widget's output—that is, the page, group, or detail in which it appears. For example, a total widget that is output in a group footer gets the totals for that group; while a total widget that is in a page footer gets the totals for the page.

Reset this property in these cases:

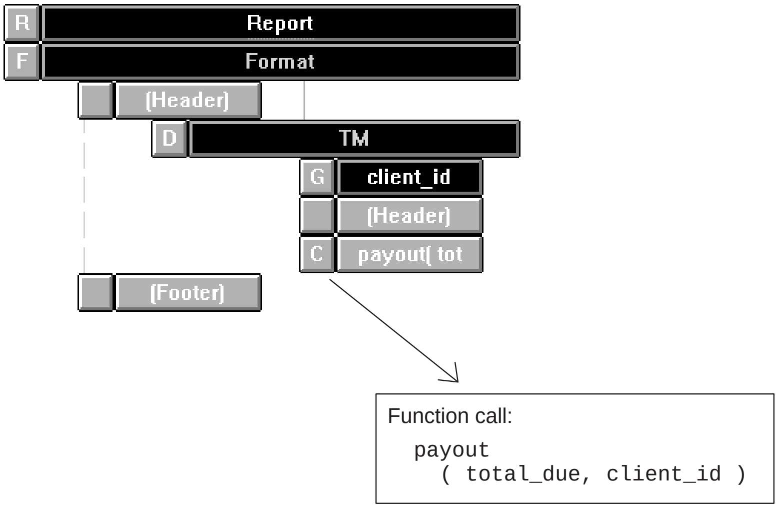
- The widget is not output—either because it is in an unused layout area or in the layout window's unnamed area.
- The widget's layout area is specified for the print node at runtime.
- The output context defined by a print node is different from the one that is actually required. This can occur when a subreport performs group footer output. In this case, each output widget's Context property should be set to Group.

In either case, ReportWriter might be unable to determine the widget's default context, so you should explicitly set its context with one of these values:

- Current — Use the latest value as this widget's context. Because ReportWriter buffers as much output as necessary in order to determine when to generate a new group or page, the latest output and current value for a widget are sometimes different.
- Group — Use the latest group output as this widget's context.
- Page — Use the latest page output as this widget's context.

example of setting context

For example, an investment firm might use a report to calculate total payments due to a set of clients. The report then calls a program that transfers the money electronically. This report has no actual output of its own; the group node that breaks on clients has a footer section that only has a call node. This calls a function that is passed two arguments that identify the client and amount due:



Function call:

```
payout
  ( total_due, client_id )
```

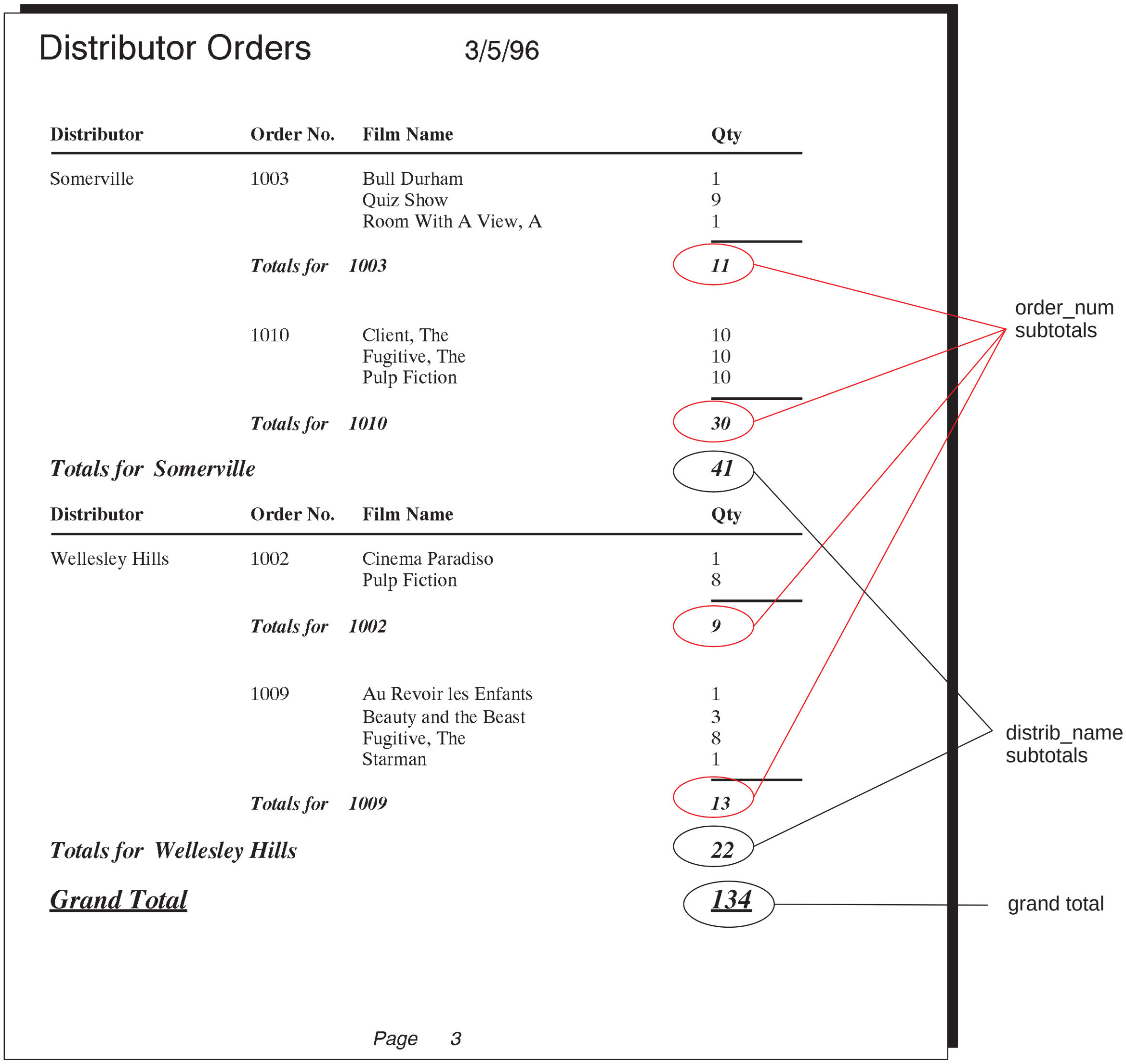
Given this structure, the report requires its layout window to contain three widgets: `client_id`, `payment_dtl`, and `total_due`. Values in `payment_dtl` are totaled in `total_due`, whose Value property is set to Total and its Value Source property to `payment_dtl`. The report has no output of its own, so it requires no layout area. Because `client_id` and `total_due` have no output context in the report structure, ReportWriter needs to be told explicitly their correct context. Thus, both widgets have their Context property set to Group.

Totaling

You can create widgets that automatically calculate and display totals for a column's values. These totals can be for a given group, for a specific page, or for the entire report.

Group and Grand Totals

Reports typically generate subtotals for one or more columns within each group, and a grand total for all values. Figure 29 shows a report whose output is grouped on `distrib_name` and `order_num`. Each group ends with a subtotal of `qty` values; the end of the report also contains a grand total for `qty`:



```
Distributor Orders                3/5/96

Distributor          Order No.   Film Name                     Qty
Somerville           1003        Bull Durham                    1
                                 Quiz Show                      9
                                 Room With A View, A            1

                     Totals for  1003                          11

                     1010        Client, The                   10
                                 Fugitive, The                 10
                                 Pulp Fiction                  10

                     Totals for  1010                          30
Totals for  Somerville                                         41
Distributor          Order No.   Film Name                     Qty
Wellesley Hills      1002        Cinema Paradiso                1
                                 Pulp Fiction                   8

                     Totals for  1002                           9

                     1009        Au Revoir les Enfants          1
                                 Beauty and the Beast           3
                                 Fugitive, The                  8
                                 Starman                        1

                     Totals for  1009                          13
Totals for  Wellesley Hills                                    22
Grand Total                                                   134

                          Page    3
```

Figure 29. Report that generates group subtotals and grand total.

To generate this output, the report relies on three components:

- Group nodes for each group that requires totaling. These include a group node that defines all detail output as a single group and enables grand totals.
- Layout areas for each group that outputs totals. Each layout area contains a *total widget*—that is, a dynamic output widget whose Value property is set to

Total. The Value Source subproperty of each total widget is set to the name of the widget whose values are totaled.

- A print node in each group node's footer section for outputting group totals; this print node specifies the appropriate layout area.

Figure 30 shows how to define a report that generates the output shown earlier:

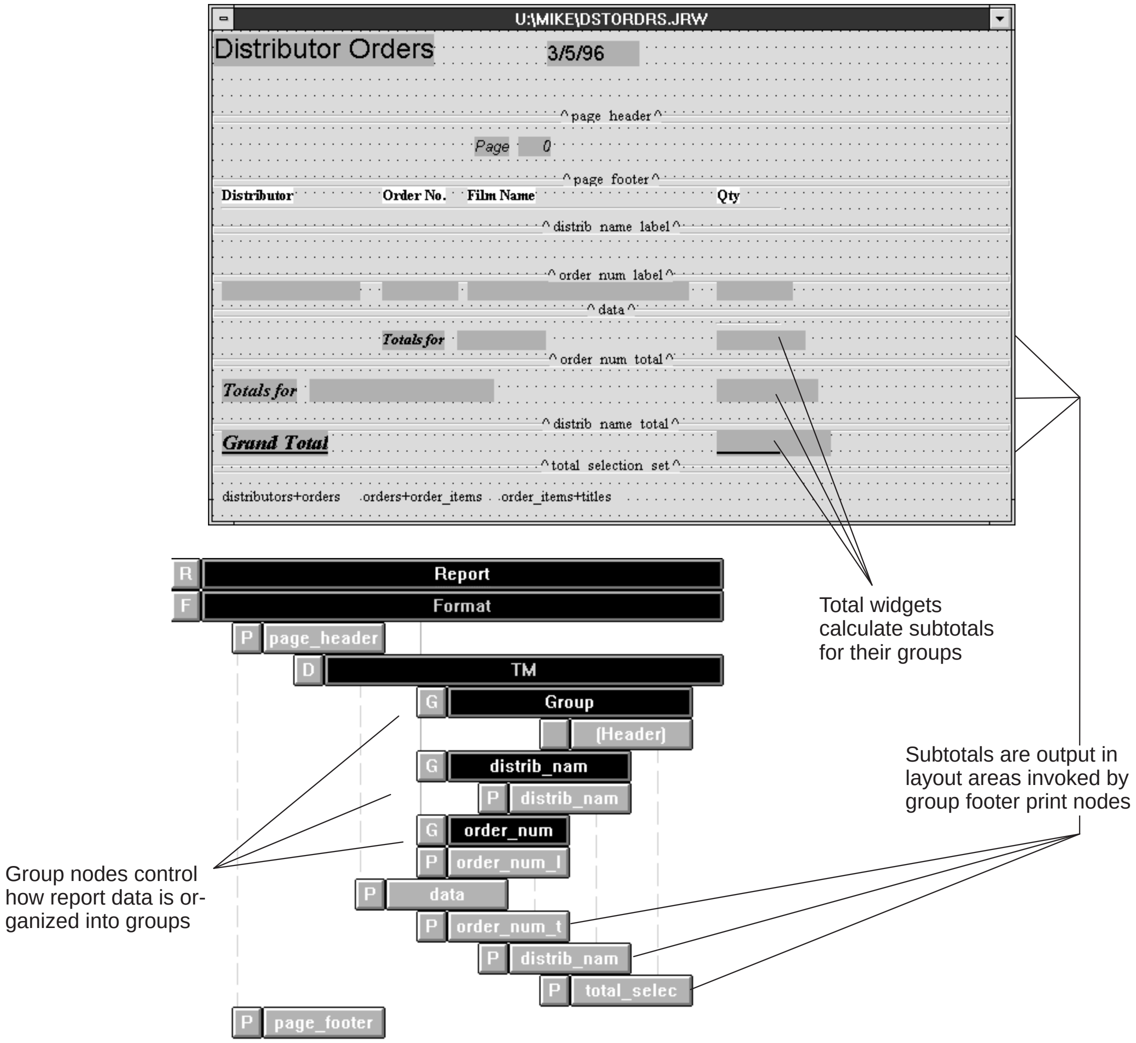


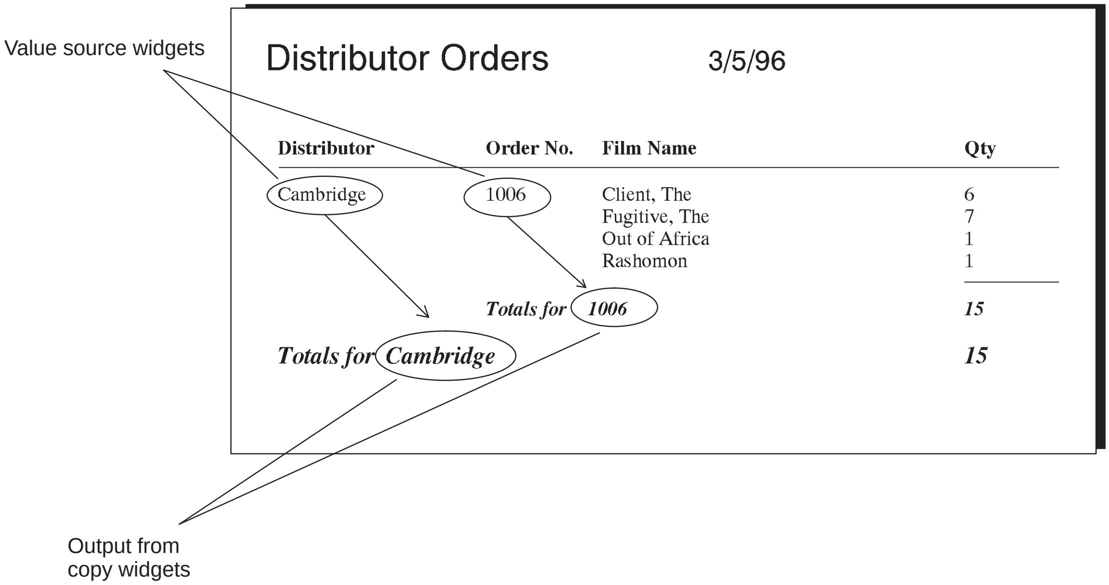
Figure 30. Layout areas and report structure for generating group and grand totals.

Page Totals

You can also generate totals at the page level: create a layout that includes a total widget and output this layout from the page format node's footer section.

Copying

Reports sometimes need to reproduce data that has already been output. For example, it can be helpful to identify a group's totals by reproducing the output of the widget on which the group broke. Figure 31 shows a report fragment where subtotals for distributors and orders are each accompanied by a dynamic output widget; these widgets each repeat the output from their respective break fields:

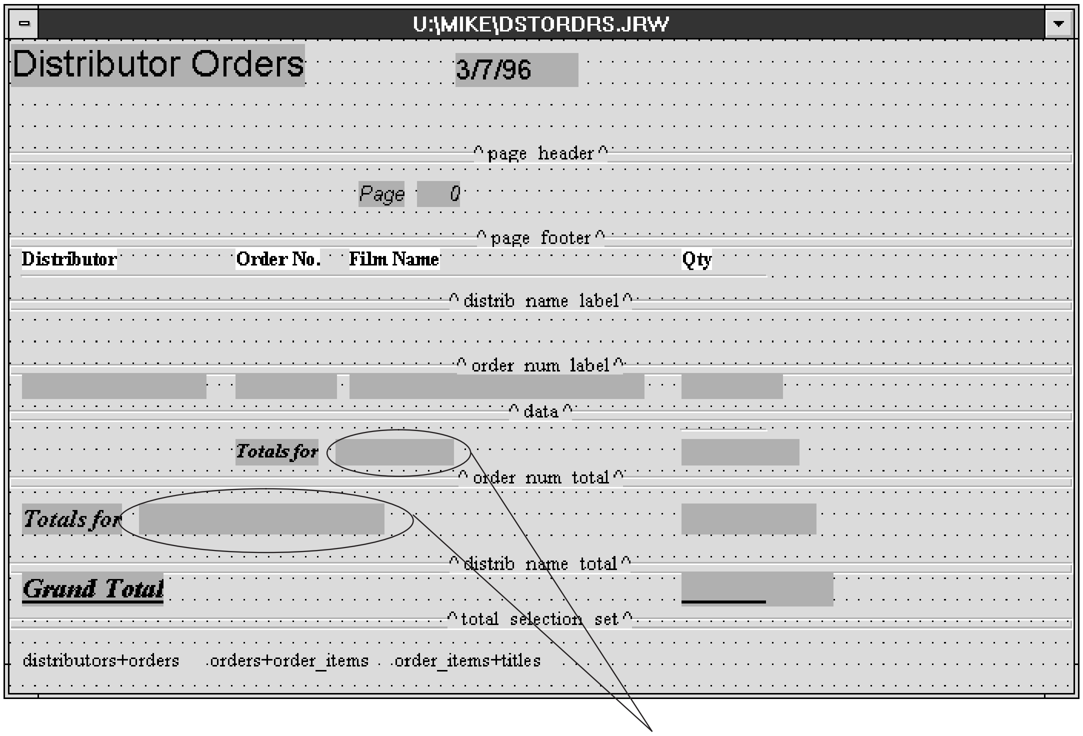


Distributor Orders 3/5/96

Distributor	Order No.	Film Name	Qty
Cambridge	1006	Client, The	6
		Fugitive, The	7
		Out of Africa	1
		Rashomon	1
	Totals for 1006		***15***
Totals for Cambridge			***15***

Figure 31. Use copy widgets to repeat output from other widgets.

To reproduce output from one widget in another, create a *copy widget*—that is, a dynamic output widget whose Value property is set to Copy. The Value Source subproperty must be set to the name of the widget whose values you wish to reproduce. For example, the layout areas for the group footers shown earlier contain copy widgets whose Value Source is set to the groups' break fields, `distrib_name` and `order_num`:



Copy widgets

The previous example copies group-level output. You can also copy data into page headers and footers. For example, you might want the page header to show the first value that is output by a group's break field, or the footer to show the last value that is output by that field.

Collecting

You can set a dynamic output widget to collect the values that are output in another widget. A widget that collects data from another—or *history widget*—can be used, for example, to collect totals for a given group and present these in a graph.

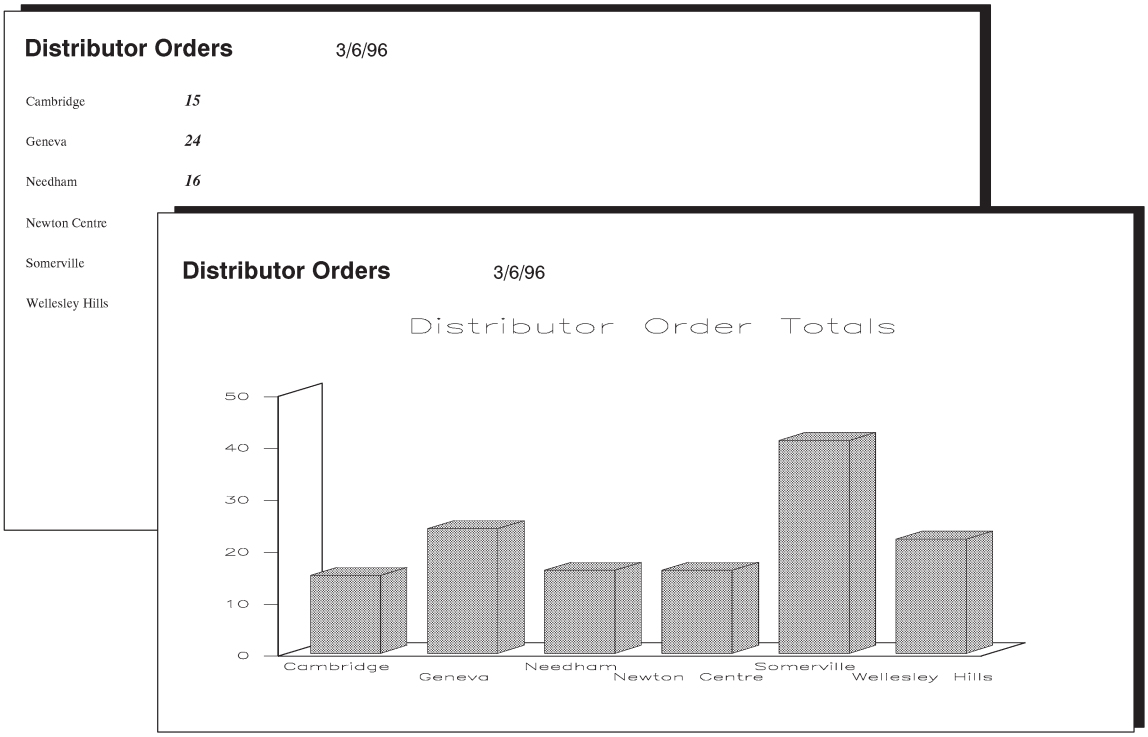
To create a history widget:

1. Create a dynamic output widget in the unnamed area of the layout window.
2. Under Geometry, make the widget a scrolling array by setting these properties:
   - Scrolling = Yes
   - Max Occurrences = blank (unlimited number of occurrences)
3. Under Composition, set these properties:

- Value = History
- Value Source = *widget-name* — The name of the widget whose values you want to collect.
- Initialize In = *node-name* — The name of a group node for which data is collected. When ReportWriter executes this node's header section, it clears all occurrences in the history array before it executes any of the group's header actions.
- Update In = *node-name* — The name of the group or detail node at which the Value Source widget's value is appended to the history widget. When ReportWriter executes *node-name*, it allocates a new occurrence in the history array and fills it with the current data in the Value Source widget. If *node-name* is a group node, the update occurs immediately after the group's footer section execute.

Using Historical Data in Graphs

Figure 32 shows a report that calculates and displays totals for each distributor in the `vbizplus` database. It then displays these totals in a graph:



Distributor Orders 3/6/96

Cambridge	*15*
Geneva	*24*
Needham	*16*
Newton Centre	
Somerville	
Wellesley Hills	

Distributor Orders 3/6/96

Figure 32. Outputting report values in a bar graph.

The data required to draw this graph is derived from two history widgets:

- `Tdistrib_qty_hist` collects the output from `Tdistrib_qty`, which totals `qty` values for each group of distributors.

- `distrib_name_copy_hist` collects the output from `distrib_name_copy`, a copy widget that gets `distrib_name`'s value for each group.

Two graph properties are set to the two history widgets: the Value Source property for Data Series #1, which populates the bar data, is set from `Tdistrib_qty_hist`; and the Label Source property for X Tick Marks, which sets the labels under each bar along the X axis, is set by `distrib_name_copy_hist`.

Displaying Page Numbers

To display page numbers in a report:

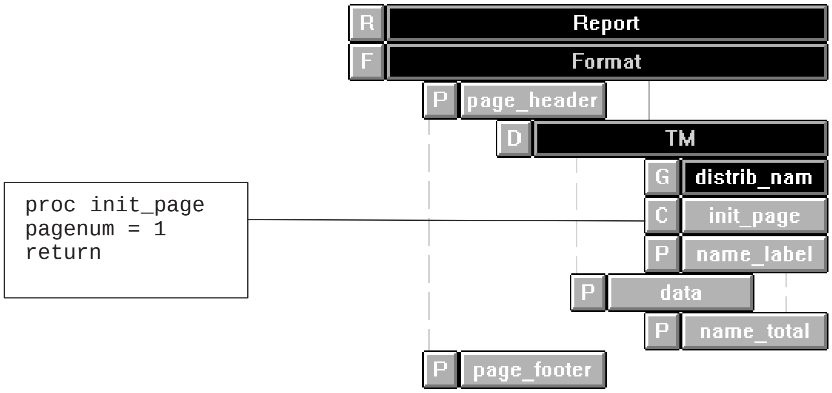
1. Create a dynamic output widget in the desired layout area—typically, an area that is output as the page header or footer.

2. Set this widget's Value property to Page Number.

By default, ReportWriter sets a page number widget to 1 at the beginning of a report and increments its value at each page break. Page numbering is continuous throughout a report. You can ensure that a subreport inherits the calling report's page number by setting the subreport node's Use Caller Format property to Yes. Otherwise, page numbering restarts for that subreport.

You can programmatically reset a page number widget's value at any stage of report execution. Simply insert a call node in the portion of the report structure where you want to restart page numbering. This call node's function resets the page number widget to the desired value.

For example, Figure 33 shows the structure of a report with a single group node; the report restarts page numbering for each iteration of that group. The group's header section contains a call node whose function `init_page` initializes page number widget `pagenum` to 1. So, each time the report processes a break in this group, `pagenum` is reinitialized to 1; successive page breaks automatically increment `pagenum`'s value until the next group break occurs:



```
proc init_page
pagenum = 1
return
```

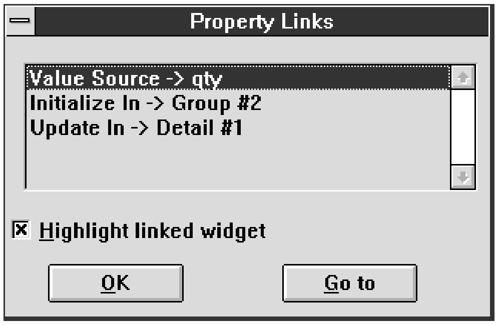
Figure 33. Report that restarts page numbering on each group break.

Viewing Property Links

To review dependencies between a dynamic output widget that processes another widget’s data, and other widgets and nodes:

1. Select the widget.
2. Choose Report⇒Show Property Links.

ReportWriter displays the Property Links dialog:



Property Links

```
Value Source -> qty
Initialize In -> Group #2
Update In -> Detail #1
```

☒ Highlight linked widget

The Property Links dialog can contain two types of entries:

- *prop-name* –> *obj-name* shows that this widget’s *prop-name* property is set to *obj-name*, either another widget or a node. If *obj-name* is an unnamed node, it is identified by its node type. For example, the links for a total widget might include these entries:

```
Value Source –> qty
Initialize In –> Group #2
Update In –> Detail #1
```

These entries shows that this widget totals values in `qty` and updates the accumulated total on each detail fetch.

Note that group and detail nodes are linked to a total or history widget's Initialize In and Update In properties even when these properties are blank. ReportWriter infers the default for these properties from the node in which the widget is output.

- *prop-name* `<–` *obj-name* shows that a node or another widget specifies this widget in its *prop-name* property. If *obj-name* is an unnamed node, it is identified by its node type. For example, the links for a widget node might include these entries:

```
Value Source <– distrib_name_copy
Break On <– Group #2
```

This entry shows that copy widget `distrib_name_copy` uses this widget as its data source; and an unnamed group node specifies it as the group's break field.

Note: *The Property Links dialog only shows links to existing widgets; if a property specifies a widget or node that does not exist—either because it is not yet created or it is included at runtime—the dialog omits this link.*

Going to a Property Link

When you display the Property Links dialog, you can give focus to the widget or node specified in the selected entry by choosing Go To, or by double-clicking on the entry. If the widget is in the structure window, ReportWriter brings this window forward; if the structure window is closed, ReportWriter opens it.

Adding Computed Values

You can take advantage of your database's aggregate or mathematical functions to calculate column values. The transaction manager fetches these values as part of the report data. To do this, create a dynamic output widget and set its properties as described below. You can create this widget either in the report itself or in a repository entry. If you add the widget to your repository, it is available for selection when you create reports through the report wizard.

Set these widget properties:

- If you are creating the widget in a repository entry, set its Name property. This value identifies the widget as a column name in report wizard dialogs.

- If the widget contains numeric values, change C Type to the appropriate value, such as Int or Float.
- In the Database category, change Use In Select to Yes.
- Set Use In Select to a database expression that calculates the widget’s value.

After you set the widget’s properties, update the members of the table view to include the new widget.

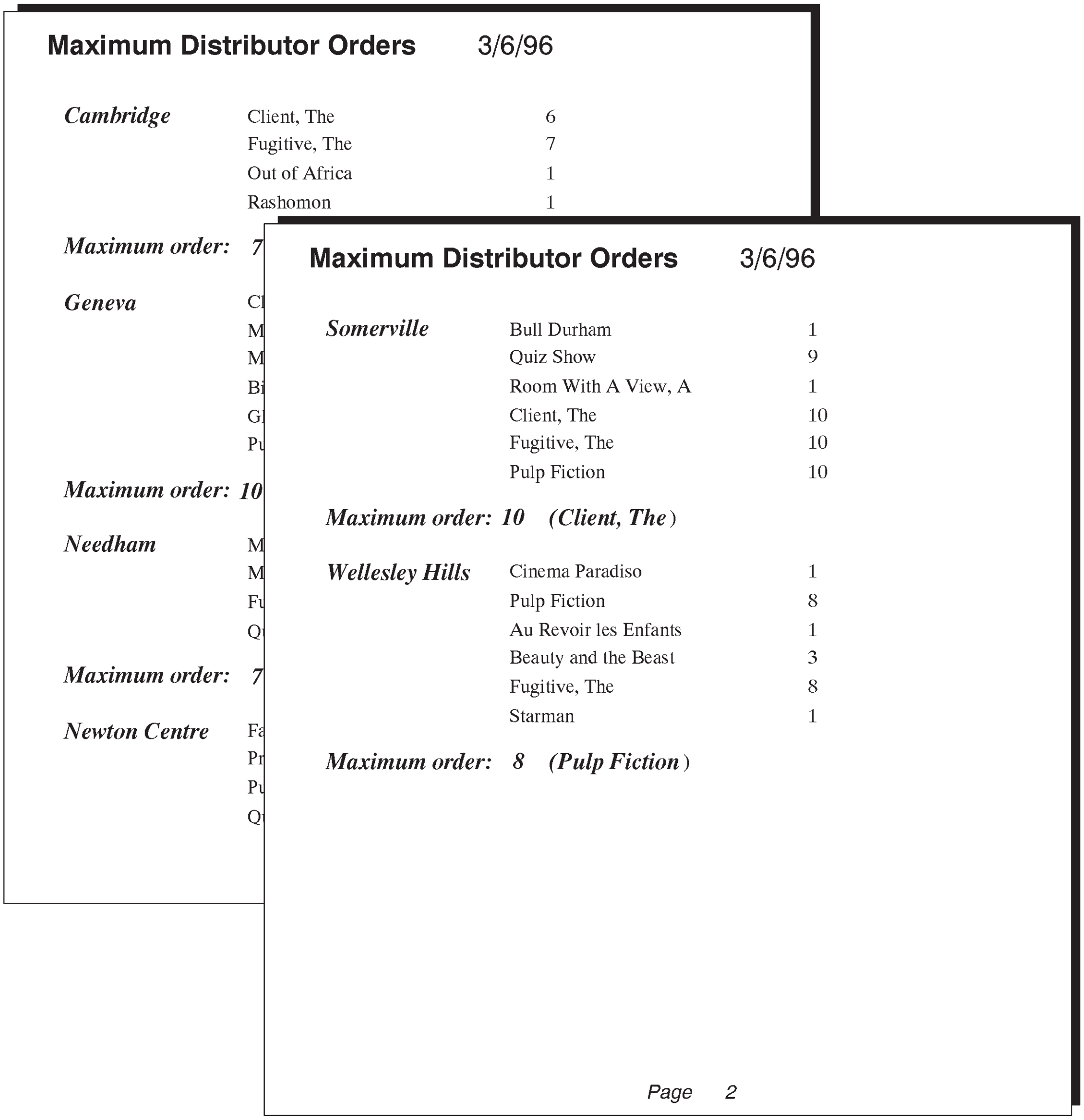
Note: *Add the widget to a repository entry in order to make it available for selection when you create reports through the report wizard.*

Using Call Nodes

By using call nodes to call your own functions, there is no limit to the types of operations that you can perform on report data. You can insert a call node anywhere in the report structure, giving you access to data at every level of report execution. The functions that you write can be written in either C or JPL; you can store them in the report file itself or on disk in a file or library.

Calculating Widget Output

You can call functions that compute typical summary data such as average, minimum, or maximum values. Summaries can be calculated for any given group, for each page, and for the entire report. The following report shows the largest video order from each distributor:



```
Maximum Distributor Orders        3/6/96

Cambridge        Client, The                6
                 Fugitive, The              7
                 Out of Africa              1
                 Rashomon                   1

Maximum order:  7

Geneva           Cl
                 M
                 M
                 Bi
                 Gl
                 Pu

Maximum order: 10

Needham          M
                 M
                 Fu
                 Q

Maximum order:  7

Newton Centre    Fa
                 Pr
                 Pu
                 Q
```

```
Maximum Distributor Orders        3/6/96

Somerville       Bull Durham                1
                 Quiz Show                  9
                 Room With A View, A        1
                 Client, The               10
                 Fugitive, The             10
                 Pulp Fiction              10

Maximum order: 10    (Client, The )

Wellesley Hills  Cinema Paradiso            1
                 Pulp Fiction               8
                 Au Revoir les Enfants      1
                 Beauty and the Beast       3
                 Fugitive, The              8
                 Starman                    1

Maximum order:  8    (Pulp Fiction )

                              Page     2
```

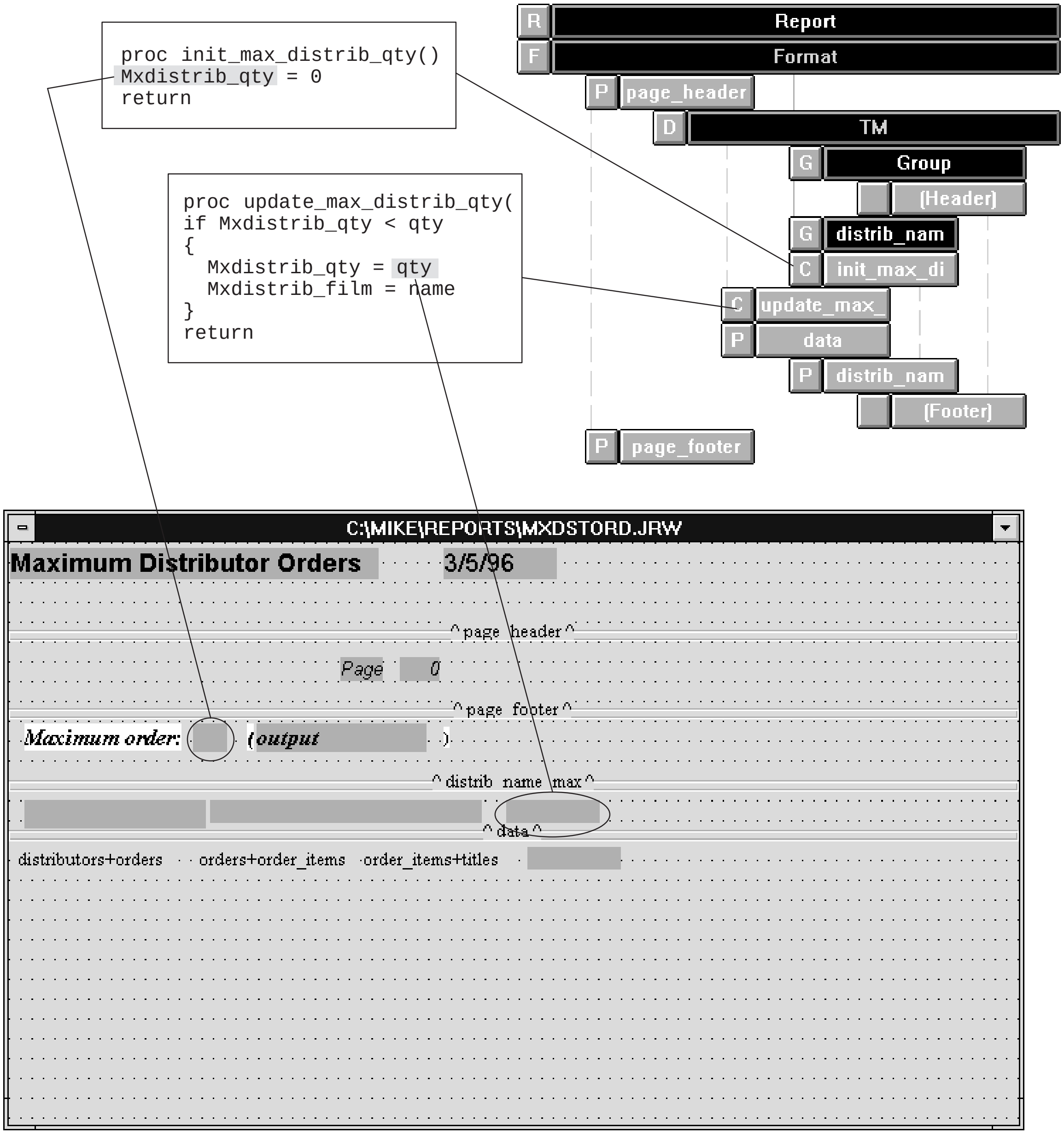
Figure 34. Maximum order quantities for each distributor.

A report can generate and display calculated data through three components:

- A *calculation widget* that collects and optionally displays the calculated data. Use a dynamic output widget that has its Value property set to Normal, and place it in the layout area where you want the calculated data to appear. For example, to display a column’s minimum value within a group, create a calculation widget in the layout area used for that group’s footer.

- A call node that initializes the calculation widget. Position this node so that it executes before calculations for the widget data begin. For example, if you want to calculate the average value of a column within a given group, insert the call node directly below the group node as a header action. Thus positioned, the call node executes on each iteration of the group and reinitializes the calculation widget before the first fetch of group data.
- A call node that updates the calculated data. Place this node below the report’s detail node as a detail action. ReportWriter calls this function each time it fetches a new row.

A report that generates the kind of output shown earlier might use the following layout area and report structure:



```
proc init_max_distrib_qty()
Mxdistrib_qty = 0
return
```

```
proc update_max_distrib_qty(
if Mxdistrib_qty < qty
{
  Mxdistrib_qty = qty
  Mxdistrib_film = name
}
return
```

Figure 35. Report that calculates maximum output.

Creating a Calculation Widget

A calculation widget acquires and optionally outputs calculated data. To show calculated data, place the calculation widget in a layout area that is output. To hide calculation widget data, place the widget in the layout window's unnamed area (below all layout areas), or set its Hidden property to Yes or Always.

The report defined in in Figure 35 defines a single group whose break field is `distrib_name`; the layout area for the group's footer contains calculation widget `Mxdistrib_qty` . During detail processing, this widget get the maximum `qty` value that is calculated for the group; when the group breaks and its footer area is output, `Mxdistrib_qty` contains this value.

Initializing Calculated Data

Before any detail data is output, the function called from the group node's header section initializes `Mxdistrib_qty` to 0:

```
proc init_max_distrib_qty()
Mxdistrib_qty = 0
return
```

This initialization function is called each time a break in `distrib_name` generates a new group.

Updating Calculated Data

After fetching and outputting each row of data, ReportWriter executes the call node that is attached to the detail node. This call node's function compares the fetched data in `qty` and compares it to `Mxdistrib_qty`; if the data in `qty` is greater, the function puts this value into `Mxdistrib_qty`; it also updates the value in the group footer widget `Mxdistrib_film`:

```
proc update_max_distrib_qty()
if Mxdistrib_qty < qty
{
  Mxdistrib_qty = qty
  /* if updating max, also get film name */
  Mxdistrib_film = name
}
return
```

Controlling Report Execution

You can programmatically control report execution through JPL return codes that affect subsequent processing; you can also modify execution by setting report node properties at runtime—for example, a print node's Area property.

Using JPL Return Codes

The JPL procedures that you call during report execution can return with values that influence subsequent processing. Table 2 lists these return codes and how they affect report execution.

Table 2. ReportWriter JPL return codes

Return code	Action
`SM_RW_OK`	Continue normal report processing.
`SM_RW_SKIP`	Skip over all remaining action nodes that are attached to the current structure node. If the call node is attached to a detail node, begin the next fetch.
`SM_RW_ENDDETAIL`	Stop fetching data and skip over all nodes attached to the current detail node.
`SM_RW_ENDREPORT`	Silently terminate current report. If this is the main report, finish outputting the current page before quitting.
`SM_RW_ENDRUN`	Silently terminate main report. Finish outputting the current page if necessary.
`SM_RW_ERROR`	Immediately stop execution of all reports and post an error message.

For example, you can conditionally execute a report’s title page, generated by a print node and end page node, if both are preceded by a call node that calls this procedure:

```
proc set_area

if draft_arg == 1        /* draft distribution */
   @widget(”title_pg_print”)–> area = ”draft_title”

else if draft_arg == 0   /* final distribution title */
   @widget(”title_pg_print”)–> area = ”final_title”

else                     /* no distribution, no title page */
   return SM_RW_SKIP     /* skip print/end page nodes */

return SM_RW_OK          /* process print node */
```

Setting Node Properties at Runtime

The previous example shows JPL code that sets a print node’s Area property at runtime. A number of node properties, listed in Table 3, are accessible at runtime. You can read and change these properties using JPL or equivalent calls to JAM library functions. For example, this JPL statement gets the name of the report invoked by subreport node `qtr_summary`:

```
vars sub_rpt
sub_rpt = @widget(”qtr_summary”)–>report_invocation
```

Only named nodes are accessible at runtime; their object class is `@widget`. For more information about accessing widget properties, refer to page 28 in the *Language Reference*.

Table 3. ReportWriter properties that are accessible at runtime.

JPL property mnemonic	Values	More...	Constraints
Detail node:			
`root`	str — root table view	73	Data Source = TM
`sql_statement`	str — `SELECT` statement	74	Data Source = SQL Query
`cursor_and_using`	str — cursor name and arguments	75	Data Source = Predefined Cursor
Group node:			
`break_on`	str — break field name	79	
Call node:			
`function_call`	str — call string	88	
Print node:			
`area`	str — layout area name	52	
Subreport node:			
`report_invocation`	str — invocation string	85	

Transferring Data from a Report

You can use JAM’s `send` command to send data back to its caller. If the caller is a JAM application, it can retrieve this data through a corresponding `receive` command; otherwise, it can use JAM library functions such as `sm_receive` or `sm_get_bundle_data`. The `send` command can send data from report widgets and global variables, or any valid expression.

The call node whose function contains the `send` command can be placed anywhere in the report structure; so you can send data back to the caller at any stage of report execution. Use `send` and `receive` as vehicles for return values when the report returns to its caller; or for sending data during report execution—for example, to save status or error messages.

Grouping Data on Computed Break Fields

In previous examples, groups are defined through break fields that obtain their values from the report's data source—typically, a database. You can also set a group node's Break Field property to a widget whose value is computed by a function written in JPL or C.

Computed breaks can be used to group data in a number of ways. For example:

- Start a new group every *n* detail lines and output a group footer that consists of blank lines, in order to improve readability of the report.
- Start a new group whenever data in the fetched row meet certain criteria—for example, an invoice value exceeds a specified amount—and output a warning.
- Group fetched rows into categories derived from the fetched data, such as dated transactions grouped into three-month periods.

In each of these cases, the group's break field is a widget or global variable whose value is computed. The function that sets the break field's value must be set in the report detail node's Pre-Break Call property. ReportWriter performs this function after it fetches the detail data but before it checks for break field changes and begins break group processing.

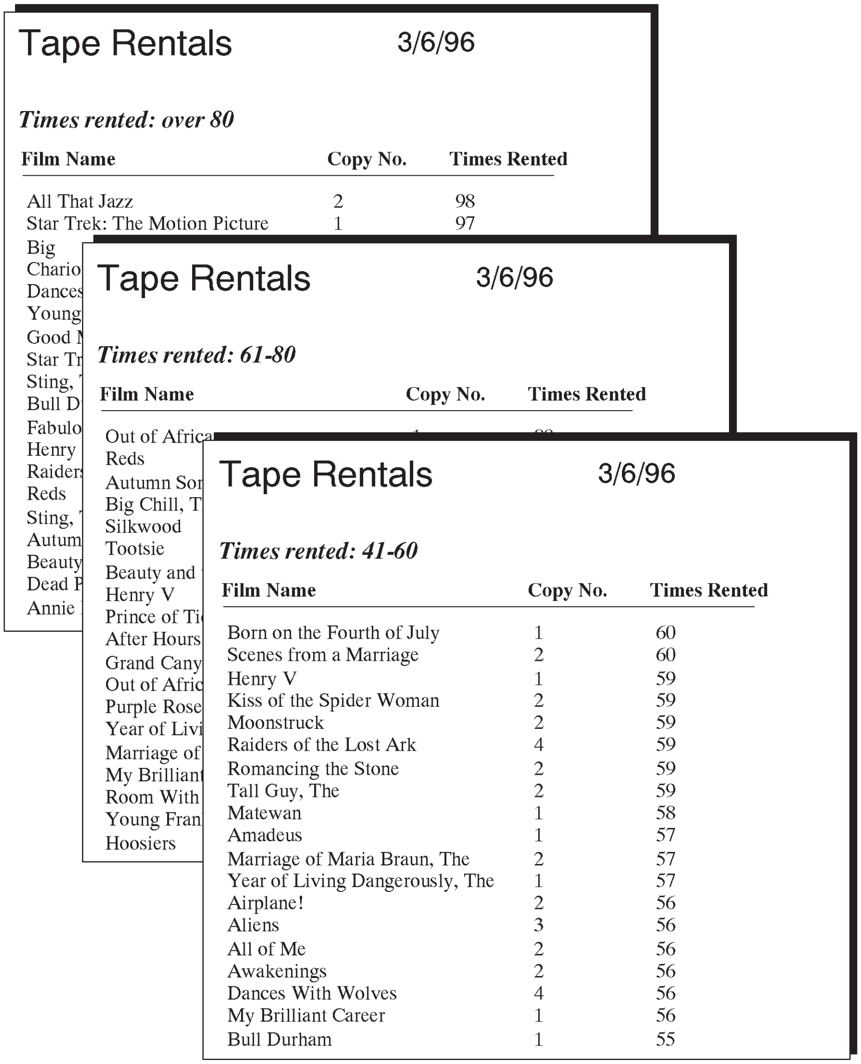
In the following example, the break field's value is derived from fetched data. The report fetches tape rental data and groups it in ranges of times rented —over 80, between 61 and 80, and 41 and 60. The report must order the fetched rows by the `times_rented` field in descending order, and contains these components:

- In the layout window's unnamed area, dynamic output widget `rental_group`. This widget serves as the group's computed break field.
- A layout area that serves as a group header; this area contains dynamic output `rental_range_label`. This widget's content is set at runtime to show the range of rentals in the current group.
- A JPL procedure that computes the value of break field `rental_group` for each fetched row; the report's detail node must specify this function in its Pre-Break Call property.
- In the structure window, a single group node whose Break Field property specifies `rental_group`.

When you run this report, ReportWriter executes `set_rental_group` each time a row is fetched:

```
proc set_rental_group

/* check the fetched value in widget times_rented
 * and determine which group this record belongs
 * in, set widgets rental_group and rental_range_label
 * accordingly.
 *
 * the ranges are:  1: 41 – 60
 *                  2: 61 – 80
 *                  3: over 80
 *
 * the variable rental_group is a non–output widget
 * in the layout window’s unnamed area
 */

if times_rented >= 81
{
   rental_group = 3
   rental_range_label = ”Number of rentals: over 80”
}
else if times_rented >= 61
{
   rental_group  = 2
   rental_range_label = ”Number of rentals: 61 – 80”
}
else if times_rented >= 41
{
   rental_group = 1
   rental_range_label = ”Number of rentals: 41 – 60”
}
else if times_rented < 41
/* don’t bother to check any further */
{
   return SM_RW_ENDDETAIL
}
```

This procedure sets the break field `rental_group` before break group checking occurs, so ReportWriter can correctly determine whether to start a new group. When you run this report, it yields this output:



Tape Rentals 3/6/96

Times rented: over 80

Film Name	Copy No.	Times Rented
All That Jazz	2	98
Star Trek: The Motion Picture	1	97
Big		
Chario		
Dances		
Young		
Good M		
Star Tr		
Sting, T		
Bull D		
Fabulo		
Henry		
Raiders		
Reds		
Sting, T		
Autum		
Beauty		
Dead P		
Annie		

Tape Rentals 3/6/96

Times rented: 61-80

Film Name	Copy No.	Times Rented
Out of Africa		
Reds		
Autumn Son		
Big Chill, T		
Silkwood		
Tootsie		
Beauty and		
Henry V		
Prince of Ti		
After Hours		
Grand Cany		
Out of Afric		
Purple Rose		
Year of Livi		
Marriage of		
My Brilliant		
Room With		
Young Fran		
Hoosiers		

Tape Rentals 3/6/96

Times rented: 41-60

Film Name	Copy No.	Times Rented
Born on the Fourth of July	1	60
Scenes from a Marriage	2	60
Henry V	1	59
Kiss of the Spider Woman	2	59
Moonstruck	2	59
Raiders of the Lost Ark	4	59
Romancing the Stone	2	59
Tall Guy, The	2	59
Matewan	1	58
Amadeus	1	57
Marriage of Maria Braun, The	2	57
Year of Living Dangerously, The	1	57
Airplane!	2	56
Aliens	3	56
All of Me	2	56
Awakenings	2	56
Dances With Wolves	4	56
My Brilliant Career	1	56
Bull Durham	1	55

Figure 36. Report with groups generated by computed break field.

6

Refining the Look

Usually, you want a report to do more than just present data; you also want it to organize and present information efficiently and attractively. For example, you might want page numbers along with descriptive text to appear at the top or bottom of each page. Reports often begin with a cover page that displays a title, the date, and other introductory material.

This chapter shows how to improve your report's appearance by:

- Controlling pagination.
- Inserting title and trailing pages.
- Managing the format of group output.
- Setting justification and indentation within a layout area.
- Eliminating extra white space.

Controlling Pagination

By default, ReportWriter allows page breaks wherever it can do so without splitting output of a layout area. If the next area to be output can fit on the current page, ReportWriter outputs it to that page; otherwise, ReportWriter begins a new page.

You can override this behavior and control page breaks in several ways:

- Allow page breaks within or between detail rows.
- Control pagination within a group.
- Keep all output within a subreport on the same page.
- Force page breaks at specific stages of report execution.
- Make page breaks conditional.

Allowing Detail Page Breaks

By default, detail print nodes have their Keep on Page property set to Yes. This keeps the output for each detail row on the same page; it also ensures that group header and footer output are accompanied with at least one detail row.

If the detail's print node has Keep on Page set to No, output for a detail row can be split across pages. Setting this property to No also allows ReportWriter to output group headers and footers without being accompanied by a detail row. So, group headers can print at the bottom of a page, before the group data is output; and group footers can be at the top of a new page.

You can also print each detail on its own page by setting the detail node's One per Page property to Yes.

Controlling Group Pagination

By default, ReportWriter prints as many groups on a single page as space allows, and splits groups across pages. Group headers and footers are printed intact and with at least one detail row. You can override this behavior in several ways:

- Keep group output together on a single page.
- Start a group on a new page.
- Allow page breaks between a group's detail output and its headers and footers.

To keep group data together on the same page:

1. Give focus to the group node.
2. In the Properties window, set Keep on Page to Yes.

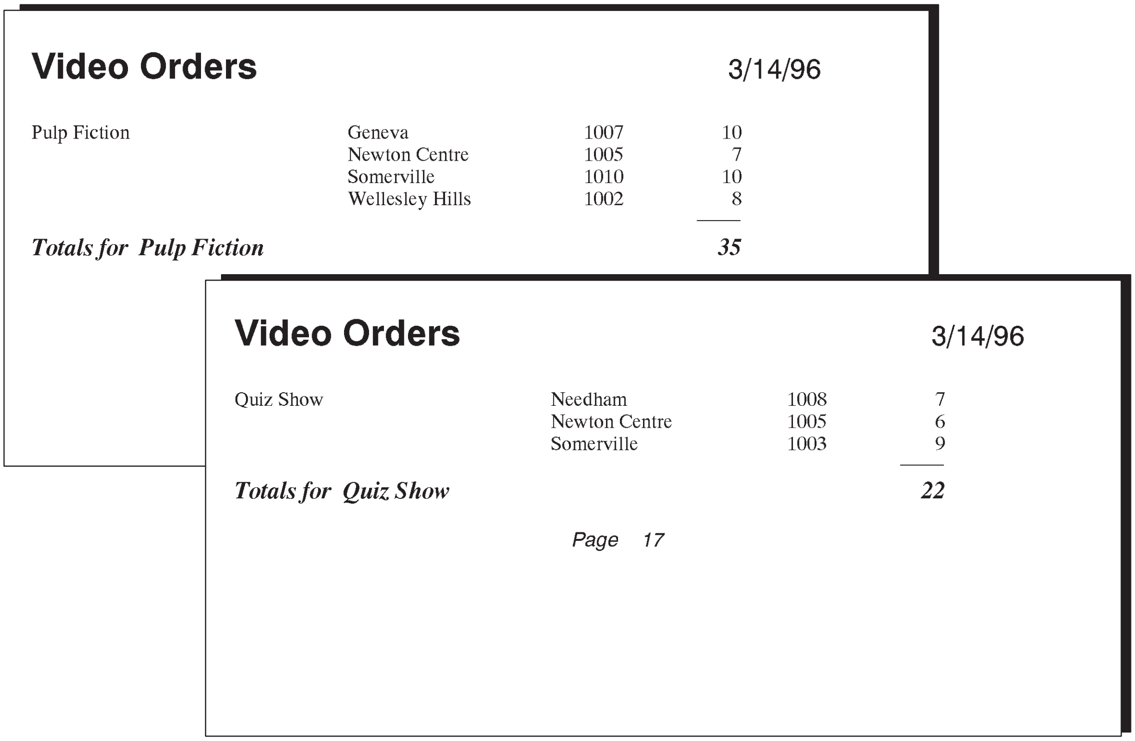
When you set a group node's Keep on Page property to Yes, ReportWriter keeps the specified group's header, detail, and footer data on the same page, if possible. If the entire group, including its header and footer, can fit on the current page, it is output immediately; otherwise, ReportWriter starts a new page.

To start a group on its own page:

1. Give focus to the group node that identifies the desired group.
2. In the Properties window, set One per Page to Yes.

When you set a group node's One per Page property to Yes, ReportWriter starts a new page for each group generated by that node. The new page begins after ReportWriter outputs the footers of the previous group.

For example, given a report that shows video orders grouped by film names, distributors, and order numbers, you can force a page break after each break on `name`; set One Per Page to Yes for `name`'s group node. This yields the following output:



```
Video Orders                                              3/14/96

Pulp Fiction                 Geneva            1007        10
                             Newton Centre     1005         7
                             Somerville        1010        10
                             Wellesley Hills   1002         8
                                                         ----
Totals for  Pulp Fiction                                   35
```

```
Video Orders                                              3/14/96

Quiz Show                    Needham           1008         7
                             Newton Centre     1005         6
                             Somerville        1003         9
                                                         ----
Totals for  Quiz Show                                      22

                          Page    17
```

To allow breaks in group headers and footers:

By default, the print nodes that output group headers and footers have Keep on Page set to Yes. This prevents page breaks within the header or footer output; it also ensures that group headers and footers are output with at least one detail row from the group. So, if a new group begins and the current page lacks enough space to fit all of the header output and at least one detail row, a new page begins before the header data. Similarly, ReportWriter reserves enough space to print on the same page all footer output along with one detail row.

If Keep on Page to set to No for the print node of a group header or footer, its output can end, span, or begin a page.

Keeping Subreport Output Together

To keep subreport output on the same page, set the subreport node's Keep on Page property to Yes. If the subreport can fit in the space remaining on the current page, it is output immediately; otherwise, ReportWriter starts a new page.

If Keep on Page is set to No (the default), subreport output begins on the current page and continues on the following pages if needed.

Forcing Page Breaks

Inserting an end page node in the report structure forces a page break at that stage of report execution. A forced page break closes the current page and executes the page footer nodes; if more output follows, ReportWriter begins a new page and executes the page header nodes. Typical uses include outputting title and trailer pages, separating sections within a report, and generating blank pages.

Note: *To start group or detail output on a new page, use the One per Page property for their corresponding nodes.*

Inserting Blank Pages

Execution of an end page node forces a page break whether or not there is output on the current page. Thus, you can insert successive end page nodes to create blank pages.

Blank pages are not always entirely blank—they typically carry over any page headers and footers that are in effect. To output a page that is truly blank, clear the existing page specifications by inserting a new page format node before blank page output begins; if no layout areas are specified for the page header and footer, the end page nodes that follow generate text-free pages.

Generating Title and Trailer Pages

Reports often begin with a cover, or title page; they can also end with a separate page that contains summary information. Reports that contain multiple sections also might require title pages for each section. You can add each of these components by creating a layout area and modifying the report structure.

To create a title page:

1. In the layout window, create a layout area that contains the output required for the title page.

2. In the report structure window, attach an instance node to the report structure's page format node.

3. Below the instance node, attach a print node and specify the title page's layout area.

4. Below the print node, attach an end page node. This forces a page break between the title page and detail data.

Note: *If you want the title page to omit the header and footer output of the report's main body, insert a new Page Format node immediately below the Report node; attach to it the instance node and title page's print node.*

To create a trailer page:

1. In the layout window, create a layout area that contains the output required for the trailing page.

2. In the report structure window, create a page format node below the last page format node.

3. Below the new page format node, attach an instance node.

4. Below the instance node, attach an end page node. This forces a page break between the detail data and the trailing page.

5. Attach a print node to the instance node and specify the trailing page's layout area.

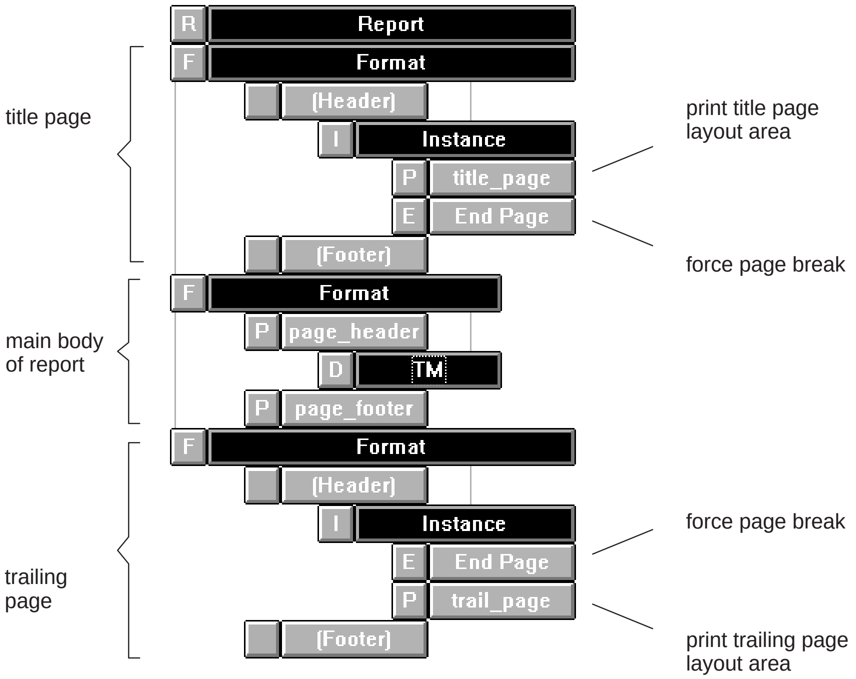


Figure 37. Report structure that specifies title and trailing pages.

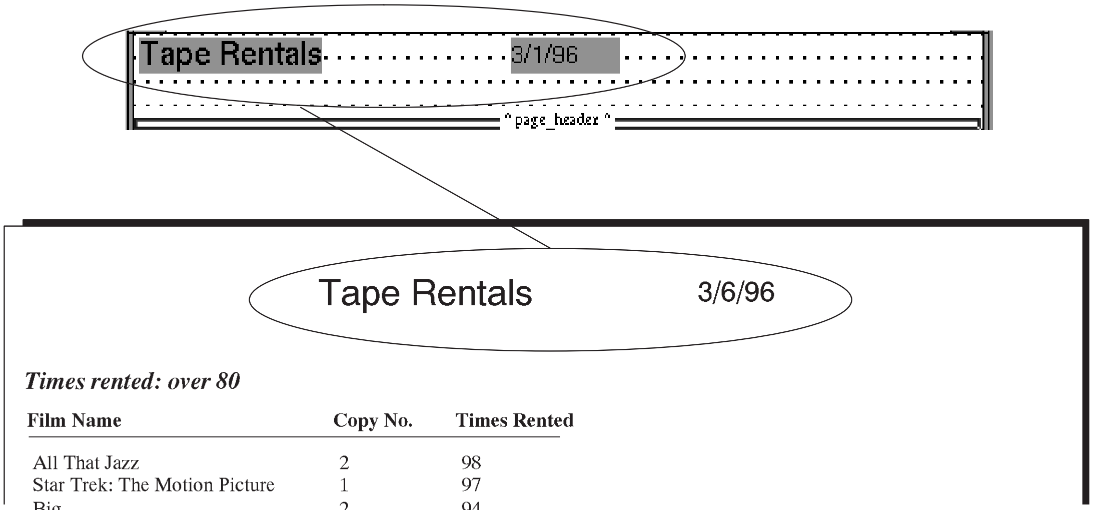
Positioning Output

Ordinarily, the layout of a print node's output is determined by the position of the widgets within its layout area. A print node has two Composition properties, Justification and Indent, that generally override widget positions and determine where to place the print node's output at runtime.

Justification

Setting a print node's Justification property to Center, Left, or Right shifts all widgets in the print node's layout area as specified. Spacing between widgets remains unchanged. If set to Normal (the default), ReportWriter uses each widget's Start Row and Start Column positions to output its data.

For example, Figure 38 shows layout area `page_header`, which is specified by the report header's print node. In the layout window, the first widget's position is flush left. You can center the actual header output by setting its print node's Justification property to Center:



Times rented: over 80

Film Name	Copy No.	Times Rented
All That Jazz	2	98
Star Trek: The Motion Picture	1	97
Big	2	94

Figure 38. To center the contents of a print node's layout area, set the print node's Justification property to Center.

Indent

Shifts right all widgets in the print node's layout by the amount of space specified. Spacing between widgets remains unchanged. Units of measurement (*unit-spec*) can be specified in inches (`in`), millimeters (`mm`), or characters (`c`—the default). If you specify dimensions in character (`c`) units, ReportWriter uses the average character size in the report's default font to calculate a character unit's width.

Reducing White Space

You can optimize use of the specified page dimensions in two ways:

- Eliminate, or shrink out, blank lines within a layout area.
- Consolidate white space that lies between contiguous layout areas.

Note that ReportWriter always outputs vertical white space between widgets, whether they are empty or not.

Shrinking Layout Areas

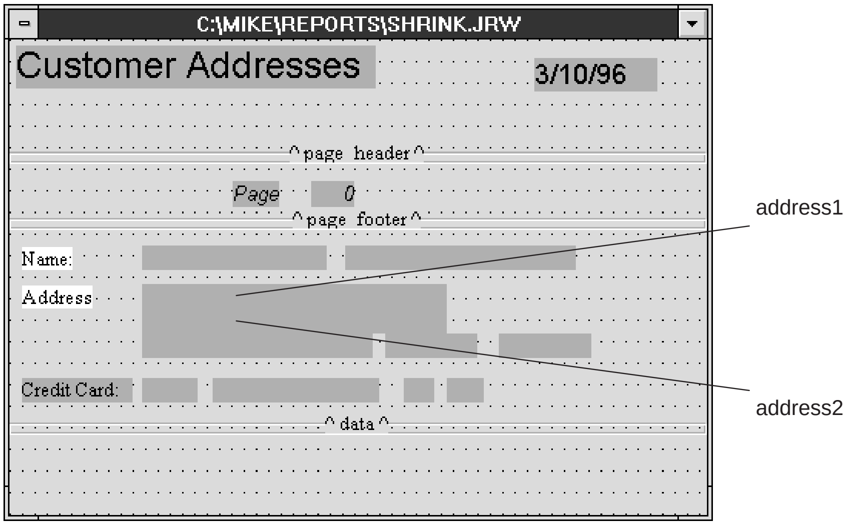
Sometimes, a layout area must anticipate variable amounts of data to output. This typically happens for two reasons:

- One line in a detail's output is blank because all fields in the corresponding record are empty.
- The layout area contains an array with a fixed number of elements, but the number of these that actually contain data varies at runtime.

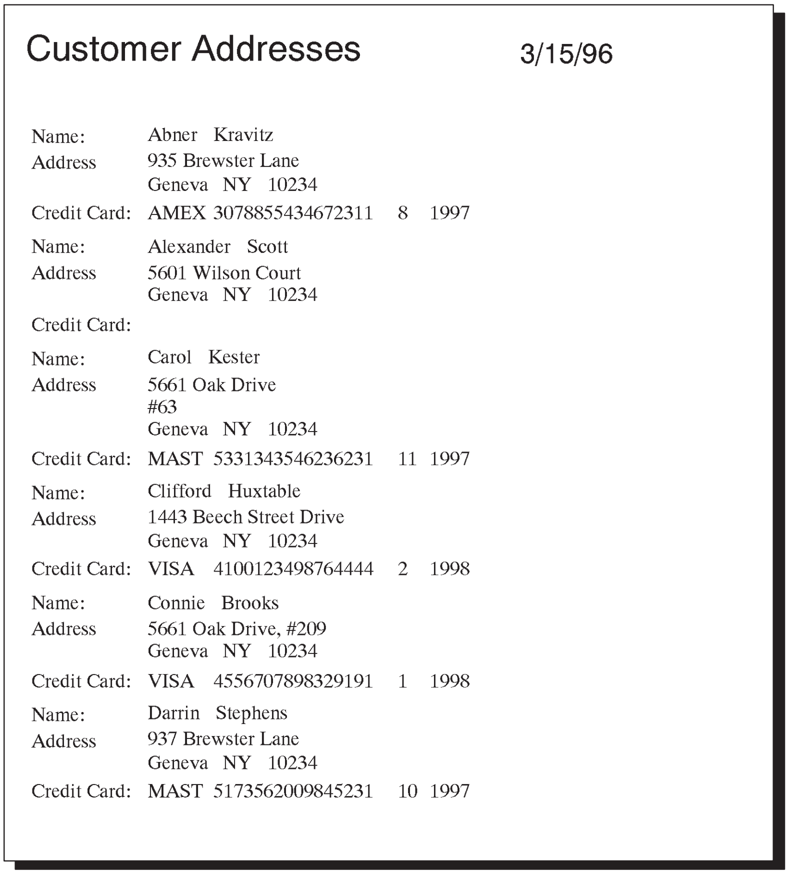
You can eliminate this white space by setting a print node's Shrink property to Yes. This tells ReportWriter to condense the layout area vertically when some of the allocated fields and array elements are empty.

Shrinking Out Empty Lines

A detail's layout area can include fields that are sometimes empty, depending on the data in a given record. For example, a report that lists customer addresses contains two fields from the `customers` table that stores street and apartment/floor numbers: `address1` and `address2`. The report's layout screen looks like this:



`address1` always contains data; `address2` is frequently empty. To close up the empty space otherwise created by `address2`, the print node that calls layout area `data` should have its Shrink property set to Yes. This yields the following output:



```
Customer Addresses                    3/15/96

Name:         Abner   Kravitz
Address       935 Brewster Lane
              Geneva   NY   10234
Credit Card:  AMEX 3078855434672311     8    1997
Name:         Alexander   Scott
Address       5601 Wilson Court
              Geneva   NY   10234
Credit Card:
Name:         Carol   Kester
Address       5661 Oak Drive
              #63
              Geneva   NY   10234
Credit Card:  MAST  5331343546236231    11   1997
Name:         Clifford   Huxtable
Address       1443 Beech Street Drive
              Geneva   NY   10234
Credit Card:  VISA    4100123498764444    2    1998
Name:         Connie   Brooks
Address       5661 Oak Drive, #209
              Geneva   NY   10234
Credit Card:  VISA    4556707898329191    1    1998
Name:         Darrin   Stephens
Address       937 Brewster Lane
              Geneva   NY   10234
Credit Card:  MAST  5173562009845231    10   1997
```

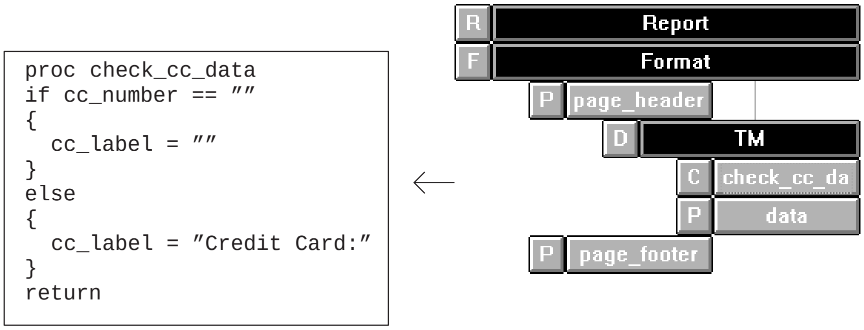
ReportWriter eliminates only lines that contain empty widgets, and only if all widgets on a line are empty.

Note: *To make sure that that all adjoining widgets are on the same line, align them horizontally so that their Start Row properties have the same value.*

The detail layout used in the earlier output contains a line for credit card information. This line contains dynamic output widget `cc_label`, which precedes the customer's credit card data—`cc_code` and `cc_number`. Not all customer records contain this information; but because `cc_label`'s Label property is set to `Credit Card`, ReportWriter outputs this line whether or not the rest of the line is empty. To output this line only when credit card data exists, modify the report to call a function that checks whether data exists in `cc_number`; if not, it sets `cc_label` to an empty string:

```
proc check_cc_data
if cc_number == ””
{
  cc_label = ””
}
return
```

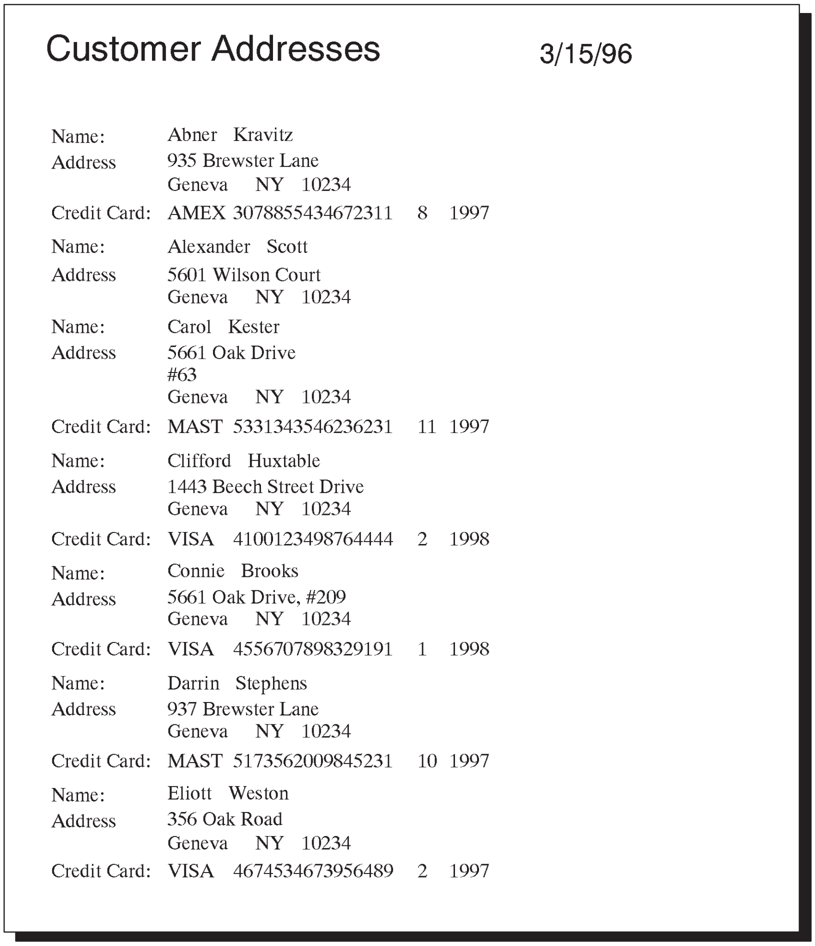
Figure 39 shows how this function is called through a call node that precedes the detail’s print node:



```
proc check_cc_data
if cc_number == ””
{
  cc_label = ””
}
else
{
  cc_label = ”Credit Card:”
}
return
```

Figure 39. Shrink out a line from the detail output by setting widgets on that line to an empty string.

When `cc_label` is set to an empty string, ReportWriter perceives the entire line as empty and removes it from the detail output. The call node that follows the print node resets `cc_label` to its previous content. When you run this report, it yields the following output:



```
Customer Addresses                    3/15/96

Name:         Abner   Kravitz
Address       935 Brewster Lane
              Geneva    NY   10234
Credit Card:  AMEX 3078855434672311    8    1997
Name:         Alexander   Scott
Address       5601 Wilson Court
              Geneva    NY   10234
Name:         Carol   Kester
Address       5661 Oak Drive
              #63
              Geneva    NY   10234
Credit Card:  MAST  5331343546236231    11  1997
Name:         Clifford   Huxtable
Address       1443 Beech Street Drive
              Geneva    NY   10234
Credit Card:  VISA    4100123498764444    2    1998
Name:         Connie   Brooks
Address       5661 Oak Drive, #209
              Geneva    NY   10234
Credit Card:  VISA    4556707898329191    1    1998
Name:         Darrin   Stephens
Address       937 Brewster Lane
              Geneva    NY   10234
Credit Card:  MAST  5173562009845231    10  1997
Name:         Eliott   Weston
Address       356 Oak Road
              Geneva    NY   10234
Credit Card:  VISA    4674534673956489    2    1997
```

Note: *Setting the Shrink property has no effect on blank space—that is, lines that contain no widgets. For information about consolidation of blank space, refer to page 126.*

Shrinking Arrays

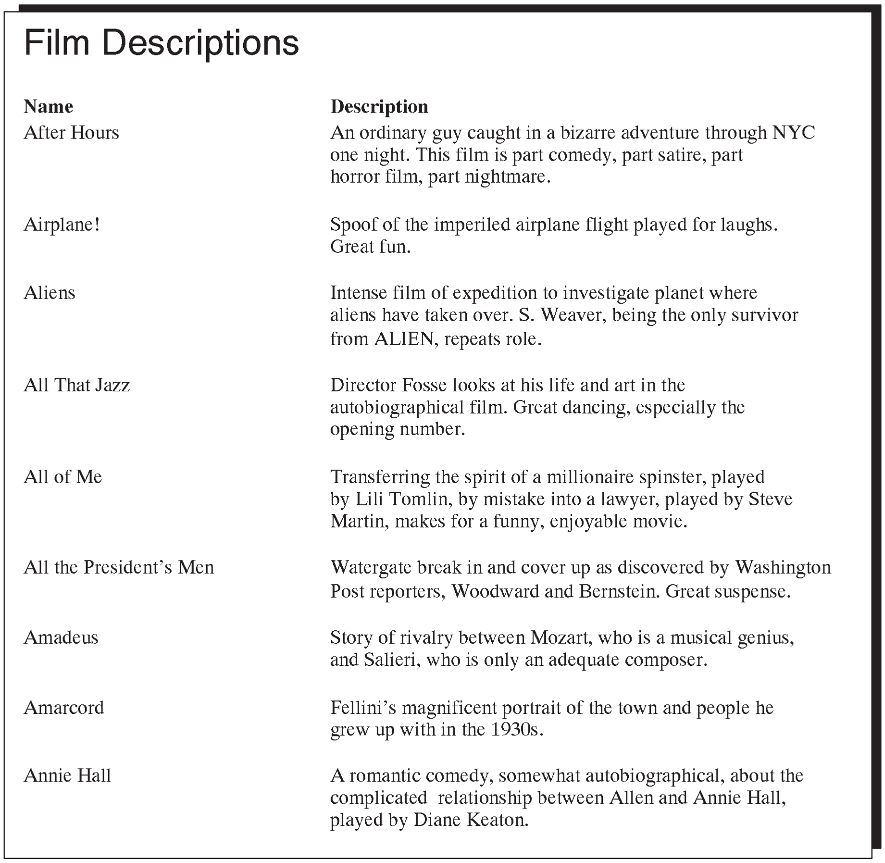
Reports can also contain arrays that are not always fully populated each time they are output. For example, you might produce an report that uses an array to show film descriptions. The amount of descriptive data varies for each film, so one or more occurrences in this array are often empty .

To ensure that the report shows only the populated rows in this array and shrinks out the remaining elements, follow these guidelines:

- Set the array’s Array Size property so there are enough elements (on-screen rows) for the longest possible description.

- Do not put any other fields on the same lines as array elements that might be empty. The presence of any non-empty field forces ReportWriter to output a line that it might otherwise shrink out.
- In the print node that outputs the array, set the Shrink property to Yes.

When the report is generated, the empty array occurrences are shrunk out, so the amount of space between all film descriptions is consistent:



Film Descriptions

Name	Description
After Hours	An ordinary guy caught in a bizarre adventure through NYC one night. This film is part comedy, part satire, part horror film, part nightmare.
Airplane!	Spoof of the imperiled airplane flight played for laughs. Great fun.
Aliens	Intense film of expedition to investigate planet where aliens have taken over. S. Weaver, being the only survivor from ALIEN, repeats role.
All That Jazz	Director Fosse looks at his life and art in the autobiographical film. Great dancing, especially the opening number.
All of Me	Transferring the spirit of a millionaire spinster, played by Lili Tomlin, by mistake into a lawyer, played by Steve Martin, makes for a funny, enjoyable movie.
All the President's Men	Watergate break in and cover up as discovered by Washington Post reporters, Woodward and Bernstein. Great suspense.
Amadeus	Story of rivalry between Mozart, who is a musical genius, and Salieri, who is only an adequate composer.
Amarcord	Fellini's magnificent portrait of the town and people he grew up with in the 1930s.
Annie Hall	A romantic comedy, somewhat autobiographical, about the complicated relationship between Allen and Annie Hall, played by Diane Keaton.

Figure 40. Eliminate empty occurrences in an array by setting the print node's Shrink property to Yes.

Note: *Setting the Shrink property to Yes removes null fields only if the null character is a blank space.*

Consolidating Blank Space

ReportWriter consolidates trailing and leading blank space of adjacent layout areas. The amount of blank space that is actually output equals the amount of trailing blank space in the first layout area or leading blank space in the second area, whichever is greater.

Note: *ReportWriter recognizes an area as blank only if it is completely empty—that is, unoccupied by widgets of any kind. An area that contains empty fields is affected only by the Shrink property.*

This behavior is especially useful in reports where group headers and footers can appear anywhere on the page—that is, the group nodes have their One Per Page property set to No. Headers can include leading blank lines to achieve natural spacing when printed immediately after the previous group footer. Blank space consolidation ensures that these blank lines do not exaggerate the amount of blank space produced when the group headers appear after a page break and immediately follow the page header. So, if the page header ends with two blank lines, and the group header begins with a single blank line. ReportWriter consolidates the amount of blank space between these areas to two lines only.

7

Running Reports

While you are developing your report, you can preview its output at any time. You can either view reports at your terminal through the editor's viewer; or you can send output to a file or printer. In graphical environments such as Windows, the viewer accurately shows the output you can expect from printed output. Because the viewer is integrated within the editor, you can respond immediately to the viewer output, then run the report to the viewer again to see how your changes look.

You can also use JAM's debugger to track down problems while you run reports in the editor—for example, trace the value of a widget in your report layout window.

When the report is complete, you can run it either from the command line or a JAM application. You can also set up a report so that users can influence report output.

ReportWriter supports these options for directing output:

- PostScript — The report is generated in Adobe standard PostScript.
- Macintosh/Windows — The report is generated using the printer driver for any installed printer. This is the default output driver when running Macintosh or Windows.
- Text — The report is generated in ASCII text.
- Metafile — The report is generated in ReportWriter's metafile format for display in the report viewer.

Running Reports from the Report Menu

The editor's Report menu contains several options that let you generate output for any report currently displayed in the editor. This menu is also available in a JAM application that uses the default JAM menu bar. You can send output to one of these destinations:

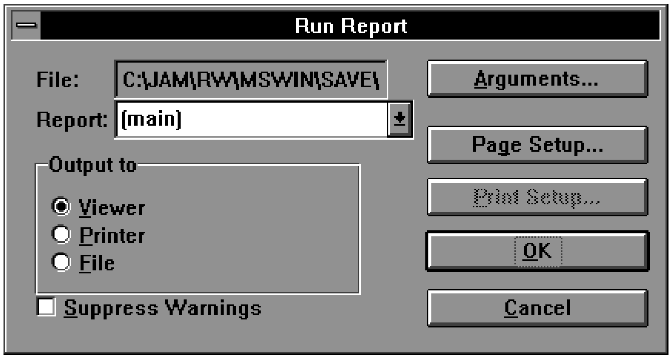
- The report viewer
- The printer specified in the Page Setup dialog
- A file

Viewing Reports Onscreen

While developing a report, you can preview its output through the report viewer:

1. If necessary, open the database required by this report: from the File menu choose Open⇒Database and select the desired database.

2. From the Report menu, choose Preview Report (from the editor) or Run Report (from a JAM application). The Run Report dialog box opens.



Run Report

File: C:\JAM\RW\MSWIN\SAVE\

Report: (main)

Output to

● Viewer
○ Printer
○ File

□ Suppress Warnings

Figure 41. The Run Report dialog box lets you view reports onscreen or send reports to a printer or file.

3. If the report you want to run is not the first one in the report structure window, select the name of the desired report from the Report option menu.

4. To inhibit display of ReportWriter warnings, select the Suppress Warnings check box.

5. Choose Arguments to enter argument values for predefined report parameters. For more information, refer to page 139.

6. Choose Page Setup to change the the page size, the page orientation, or the page margins. Choose OK when finished.

7. Choose OK to generate the report and display it in the report viewer.

As each page on the report is complete, it is available for display in the report viewer.

The report viewer has its own menu and toolbar. The View menu lets you navigate through the report and determine the size of the report display. The File menu lets you print the current output; it also lets you save the output to a metafile for later viewing.

Saving Viewer Output

While in the viewer, you can save report output to metafile format by choosing File⇒Save. After you enter a filename, choose OK. The report is saved to a file with a default `.rwm` extension in ReportWriter's metafile format. The output saved to this file can be displayed at any time through the report viewer.

To view report output saved in a metafile:

1. From the Report menu, choose Output Viewer.

2. From the viewer's File menu, choose Open. The file browser dialog displays available files.

3. Select or enter the name of the desired metafile and choose OK. The viewer displays the output saved to this file.

4. When your viewing session is complete, return to the editor by choosing File⇒Exit.

Printing Reports

You can use the Report menu to send output to a printer or file:

1. From the Report menu, choose Preview Report (from the editor) or Run Report (from a JAM application).

2. Choose the File or Printer option in the Output To box.

3. If the report you want to run is not the first report in the report structure window, select the name of the desired report from the Report option menu.

4. To inhibit display of ReportWriter warnings, select the Suppress Warnings check box.

5. Choose Arguments to enter argument values for predefined report parameters. For more information, refer to page 139.

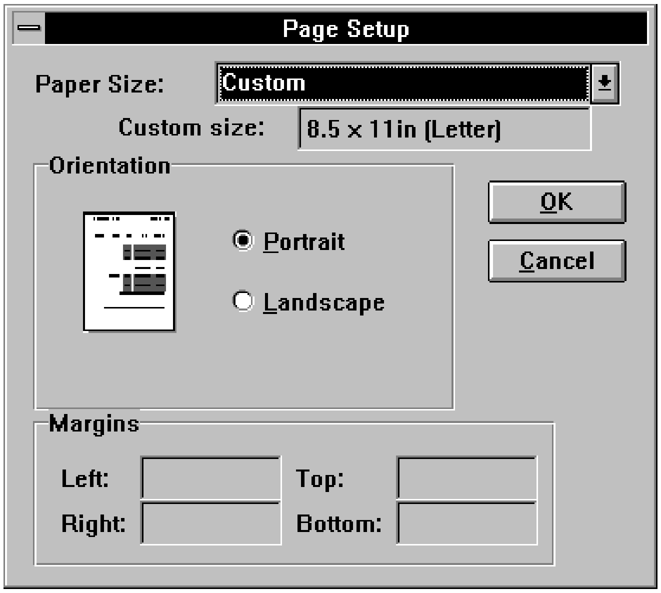
6. Choose Print Setup to change the printing options. The Print Setup dialog is different for each platform. Variations of this dialog and its options are described later.

7. Choose Page Setup to change the the page size, the page orientation, or the page margins. This dialog's options are described later. If there is a conflict between the settings in the report's properties and in the Page Setup dialog box, the settings in the report have precedence.

8. Choose OK.

Page Setup

The Page Setup dialog specifies page size, orientation, and margins. On UNIX systems, the initial settings for the Page Setup dialog box come from the `rwviewpr.ini` file, if available. This file is created by selecting the Save On Exit check box on the Print Setup dialog box for UNIX. On Windows and Macintosh, the paper size and orientation are derived from the system's settings.



Paper Size: Custom

Custom size: 8.5 x 11in (Letter)

Orientation

● Portrait

○ Landscape

Margins

Left:

Top:

Right:

Bottom:

Figure 42. On the Page Setup dialog box, set additional margins, change the page size, and change the page orientation.

Paper Size

Specify the dimensions of the output area on report pages. You can select one of the predefined page sizes from the drop-down list, or specify your own page size in this format:

width x *height [unit-spec]*

Refer to page 71 for valid units of measurement.

Orientation

Select Portrait to use the shorter page measurement as the width of the report. Select Landscape to use the longer page measurement as the width.

Margins

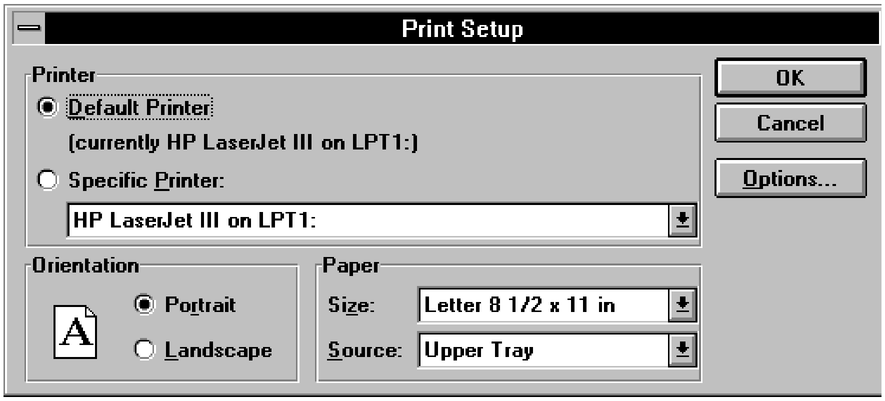
If you want to add space to the report's margins, enter the additional margins here. Specify the amount of space in this format:

margin-space [unit-spec]

Units of measurement (*unit-spec*) can be specified in inches (`in`), millimeters (`mm`), or characters (`c`—the default). If you specify dimensions in character (`c`) units, ReportWriter uses the average character size in the report's default font to calculate a character unit's width.

Windows Print Setup

In Windows, choosing Print Setup from the editor displays the Windows Print Setup dialog box:



Printer

● Default Printer

(currently HP LaserJet III on LPT1:)

○ Specific Printer:

HP LaserJet III on LPT1:

Orientation

● Portrait

○ Landscape

Paper

Size: Letter 8 1/2 x 11 in

Source: Upper Tray

Printer

The default printer is automatically selected, but you can change it to any printer available in your Windows Print Manager.

Page Orientation

Select Portrait to use the shorter page measurement as the width of the report. Select Landscape to use the longer page measurement as the width.

Paper Size

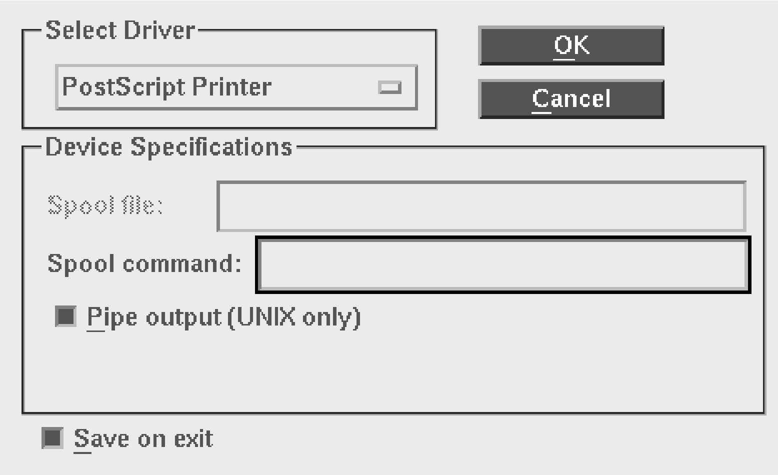
Specify the dimensions of the output area on report pages. You can select one of the predefined page sizes from the drop-down list, or specify your own page size in this format:

width x *height [unit-spec]*

Refer to page 71 for valid units of measurement.

UNIX Print Setup

On UNIX systems, choosing Print Setup displays a dialog box that lets you control the print driver, the spool command, and the device file for the report:



Select Driver: PostScript Printer

☒ Pipe output (UNIX only)

☒ Save on exit

Select Driver

Choose the output driver to use for the file:

- PostScript Printer generates PostScript output.
- Generic Printer uses the text driver for ASCII report output.

Spool File

Enter the name of the file to hold the report output before it is sent to the printer. For this option to be active, the Pipe Output option must be inactive.

Spool Command

Enter the print command used in your operating system. For Motif and character mode, if Pipe output is selected, the report is sent directly to the printer. Otherwise, the report is stored in the spool file and the command should include the spool file name.

Pipe output
On UNIX systems, this option will automatically send the report output to the printer using the command in Spool Command.

Device File
Enter the name of the device file to use in printing an ASCII report with the Generic Printer driver. Each device file is a binary file, compiled from ASCII, that enables printing options. For more information on creating device files, refer to page 173.

Save on exit
If selected, Page Setup and Print Setup dialog settings on UNIX platforms are saved to `rwviewpr.ini` in the current directory.

Running Reports Outside the Editor

You can run a report outside the editor in three ways:

- From the command line with the `rwrun` utility.
- Call the JPL command `runreport`.
- Call the library function `sm_rw_runreport`.

All of these invocation options use the same report invocation string:

filename [`!`*reportname] [* `(`*arg[*`,` *...]* `)`*] [option]...*

filename [`!`*reportname]*
filename is the name of a binary report file. If *reportname* is supplied, the specified report in this file is invoked. If *reportname* is omitted, the first report defined in the file is used.

arg
Supply one or more arguments. Refer to page 139 for more information.

option
Specify one or more options to specify the output file or report format specifications. For descriptions of available options, refer to page 136.

To invoke a report from the command line:

Use `rwrun` to run reports from the command line or in a batch file. `rwrun` has this syntax:

`rwrun` *filename* [`!`*reportname*] [`(`*arg*[`,` ...] `)`] [*option*]...

For example, the following UNIX invocation runs the `custinfo.jrw` report and writes the report to the file `custinfo.txt`. The invocation string contains an argument, `11`, which is the customer ID number to use in the report.

```
rwrun custinfo.jrw \(11\) output=custinfo.txt
```

For more information about this utility, refer to page 155.

Note: *Any punctuation that has special usage in the operating system, such as parentheses* `()`*, must be prefixed with an escape character (*`\`*).*

To invoke a report from a JPL procedure:

`runreport` *filename* [`!`*reportname*] [`(`*arg*[`,` ...] `)`] [*option*]...

The following JPL procedure runs the the `custinfo.jrw` report for customer ID `11` and writes the report to the file `custinfo.txt`.

```
proc make_report
runreport custinfo.jrw (11) output=custinfo.txt
return
```

To invoke a report from a C function:

`sm_rw_runreport (char *`*invocation-string*`)`

For example:

```
int retcode;
retcode = sm_rw_runreport
          (”custinfo.jrw (11) output=custinfo.txt”);
```

For more information about this function, refer to page 146.

This invocation string can also be used for subreports. For more information on subreports, refer to page 84.

Defining Invocation Options

The following options are available for `rwrun`, `runreport`, and `sm_rw_runreport`.

Output Options

`output =` *output-file*

Direct the finished report to the named file. If this option is omitted, and if no spool command or output procedure is specified in the device file, the report is sent to standard output.

`overwrite`

The `overwrite` option allows report output to overwrite the current contents of the file named in the `output` option.

`driver =` *driver-name*

Generate the report according to the specified output driver, where *driver-name* can have one of these values:

- `postscript` generates the report in Adobe PostScript.
- `text` generates the report in ASCII text.
- `windows` generates the report in the format needed by the specified Windows printer.
- `macintosh` generates the report in the format needed by the Macintosh system.
- `rwmetafile` generates the report in ReportWriter's metafile format.

`printer =` *printer-name*

(Windows driver only) Direct the finished report to the named printer.

`device =` *device-file*

(Text driver only) Use the capabilities in the named device configuration file to control report output. Refer to page 173 for a detailed description of the device file.

`spool =` *spool-command*

(UNIX only) Spool the output using the specified command. For example:

```
rwrun custhist.jrw spool=”lpr –Pljet”
```

`showwarnings | nowarnings`

If `showwarnings` is specified, warning messages are displayed on the screen or sent to the standard destination for error messages in your configuration. If `nowarnings` is specified, warning messages are ignored. `showwarnings` and `nowarnings` are mutually exclusive.

If neither is specified, the default depends on whether the report is invoked as a primary report or as a subreport:

- Primary report — warning messages are displayed.
- Subreport — warning messages are ignored.

Format Options

`paper_size =` *dimensions*
Generate the report using the paper size specified in the following format:

width x *height [unit-spec]*

margin-type = *margin-space [unit-spec]*
You can specify four margins for the page, where *margin-type* can be one of these mnemonics:

```
leftmargin
rightmargin
topmargin
bottommargin
```

Units of measurement (*unit-spec*) can be specified in inches (`in`), millimeters (`mm`), or characters (`c`—the default). If you specify dimensions in character (`c`) units, ReportWriter uses the average character size in the report's default font to calculate a character unit's width. The default value for all margins is 0.

For example, this string specifies a left margin of 1 inch:

```
leftmargin = 1 in
```

`portrait | landscape`
The `portrait` keyword denotes output printed with paper in an upright position: the longest edges are vertical. The `landscape` keyword directs the device in use to print with paper in the sideways position: the longest edges are horizontal. The two keywords are mutually exclusive; `portrait` is the default.

When the `text` driver is in use (the default on all platforms except Windows), ReportWriter seeks the capabilities `landscapeon` and `landscapeoff` in the device file in use. It outputs the control string given by `landscapeon` at the beginning of each page printed in landscape mode, and follows each such page with the string given by `landscapeoff`. Refer to page 176 for details.

The `postscript` and `windows` drivers require no additional support for landscape output.

Examples

The following examples use the `rwrun` command to generate the report found in `custhist.jrw` directly from the command line:

Generate the report and display the output on the terminal screen

Enter the following at the command line:

```
rwrun custhist.jrw
```

Generate output using a device file

The following command suppresses any warning messages and uses the device file `printfile` to generate the output of `custhist.jrw` and to specify the printer for the output:

```
rwrun custhist.jrw nowarnings device=printfile
```

Send PostScript output to a file

Generate the report in PostScript and save it to the file `c.ps`, even if the file already exists:

```
rwrun custhist.jrw driver=postscript output=c.ps overwrite
```

Supplying Arguments

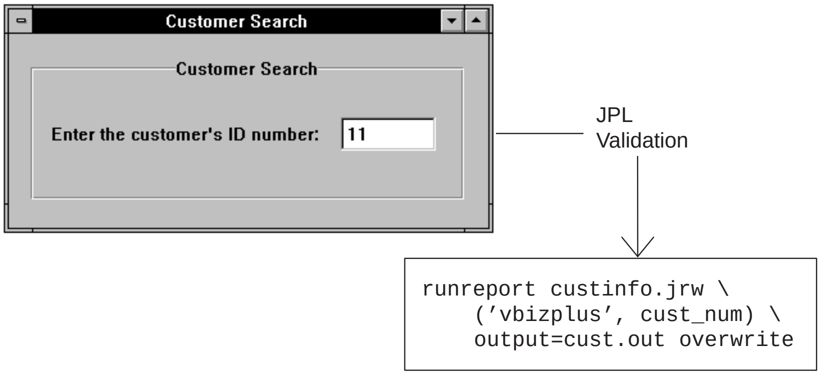
Reports can be invoked with arguments. Each argument must be a valid JPL expression—either a string within quotation marks, a number, or the name of a JAM variable to evaluate when the report is run. Typically, these arguments contain data that define the scope of the report. In order to process these arguments, the following conditions must be met:

- A parameter corresponding to the argument must be declared in the report node’s Parameters property (refer to page 69).
- Arguments must be supplied in the same order as their corresponding parameters are defined.
- Each declared parameter variable must exist in the report as a widget, a JPL global variable, or an LDB variable.

Figure 43 shows a sample report invocation that uses arguments

Creating Interactive Reports

You can design a report to give its user considerable control over the actual output. For example, a report can accept user input to determine report composition, such as which layout areas to use (refer to page 110). The following example shows an application that invokes a report and asks the user to enter information that determines the data fetched for the report. In Figure 43, a user enters the customer code for which they want data:



Enter the customer's ID number: 11

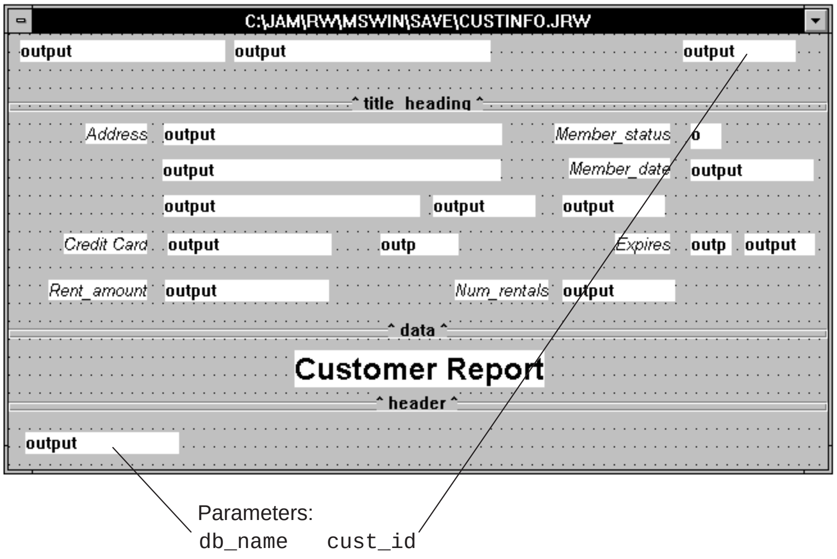
```
runreport custinfo.jrw \
    ('vbizplus', cust_num) \
    output=cust.out overwrite
```

Figure 43. Query screen asks for the customer ID number, which the application supplies to the report as an invocation argument.

After the code is entered in the `cust_num` widget, the JPL Validation property for the widget is used to generate a report with the customer data. In this example, the database name and the customer code are passed as arguments, the database name as a string and the customer code as a JPL variable:

```
runreport custinfo.jrw \
     ('vbizplus', cust_num)\
     output=cust.out overwrite
```

The report has two parameters defined in its report node's Parameters property, `db_name` and `cust_id`, which are dynamic output widgets:



Parameters:
`db_name` `cust_id`

Figure 44. The values are then stored in the widgets named in the Parameters property.

`db_name`, the first parameter, receives the value of the first invocation argument, which is a database name. The report uses this database name in a call node to connect to the database.

The second parameter `cust_id` receives the value of the second invocation argument, `cust_num`. A call node executes this JPL procedure to declare the cursor:

```
proc make_cursor
DBMS DECLARE sel_cursor CURSOR FOR \
   SELECT cust_id, first_name, last_name, \
   address1, address2, city, state_prov, postal_code, \
   phone, cc_code, cc_number, cc_exp_month, cc_exp_year, \
   member_date, member_status, num_rentals, rent_amount
   FROM customers
   WHERE cust_id = ::cust_id
```

The detail node has its Data Source property set to Predefined Cursor and executes the following entry in the Cursor and Using property to fetch data for the report:

```
sel_cursor cust_id
```

Debugging Reports

The JAM debugger is just as useful in debugging reports as in analyzing JAM applications. The debugger's Data Watch window, for instance, can display the current value of any widget in your report layout window, as well as JPL variables and LDB entries.

When a report is running, the debugger's Source Code window lists the report structure with the current node highlighted. You can step through execution of the report, set breakpoints, and animate the display exactly as you would in a JPL program. JPL appears in the Source Code window if you are tracing it, whether it is called directly by your application—say, from a control string—or from within a report.

Report events are displayed in the debugger's Status window. These events include node execution, area output, subreport invocation, and page processing. Because they might cause other report events, and non-report events such as execution of a JPL procedure, each report event remains displayed in the debugger's Event Stack window until it completes.

Refer to page 495 in the *Application Development Guide* for information about JAM's debugger.

8

Library Functions

This chapter contains descriptions of library functions supplied with JAM/Report-Writer. Each function description tells what the function does, and where and how to use it. Information about each function is organized into the following sections:

- Syntax lines that are patterned after C function declarations. Syntax lines are preceded by `include` statements that are specific to the function.
- Parameter descriptions.
- Return values.
- Description of the function—typical usage, prerequisites, results, and potential side-effects.
- An example that shows how to use the function.

sm_rw_play_metafile

Displays or prints a report which is in metafile format

```
#include <rwdefs.h>

int sm_rw_play_metafile( char *report_string );
```

`report_string` A string composed of the metafile name and one or more arguments:

filename { driver_option... | viewer_option } output_option...

filename
The metafile's file name.

driver-option
Specify one or more driver options with the following strings:

Driver specifier	Description
`device=`*device-file*	Name of a compiled device configuration file (text driver only).
`driver=`*driver-name*	One of the following drivers: `postscript`, `text`, `macintosh`, `windows`, or `rwmetafile`.
`printer=`*printer-name*	Name of the printer (Windows driver only).

viewer-option
One of the viewer options listed in the following table. Viewer specifiers are only available in interactive mode.

Viewer specifier	Description
`view`	Display the output in the viewer.
`print`	Print the output using the current settings for the Print Setup and Page Setup menu options.
`printtofile`	Save the output to a file using the current settings for the Print Setup and Page Setup menu options.

output-option

Specify one or more output options with the following strings:

Output specifier	Description
`output=`*output_file*	Name of the output file.
`overwrite`	Overwrite the output file if it already exists. If this keyword is not specified, ReportWriter asks if you want to overwrite the file.
`frompage=`*startpage*	Positive integer specifying the starting page.
`topage=`*endpage*	Positive integer specifying the ending page.

Returns

0	Success.
–1	A syntax error occurred or the specified file is not in metafile format.

Description

`sm_rw_play_metafile` takes an existing metafile and processes it according to the output specifications.

Example

```
#include <rwdefs.h>
int retcode;
/* Display the report output in the viewer */
retcode = sm_rw_play_metafile (”report1.rwm view”);

/*
 * Print the output to a user–specified file using the
 * driver and printer currently selected in the Print
 * Setup dialog box
 */
retcode = sm_rw_play_metafile (”report1.rwm printtofile”);

/* Save the metafile as a PostScript file.*/
retcode = sm_rw_play_metafile (”report1.rwm driver=postscript
          output=report1.ps overwrite”);

/* Save the first two pages of the metafile to a
 * PostScript file
 */
retcode = sm_rw_play_metafile (”report1.rwm driver=postscript
          output=report1.ps overwrite topage=2”);
```

sm_rw_runreport

Invokes the report generator from a user-written function

```
#include <rwdefs.h>

int sm_rw_runreport ( char *report_string );
```

`report_string` A string that contains the name of the report to be invoked, arguments passed to the report, and output and page layout options. The format of the string is identical to the invocation string for the JPL command `runreport`:

"filename [`!`*reportname] [('arg' [, ...])] [option]..."*

For a description of invocation arguments and options, refer to page 136.

Returns

0	Success.
–1	Failure.

Description

`sm_rw_runreport` invokes the report generator from a user-written function linked into a JAM/ReportWriter application. It is the C equivalent of the JPL command `runreport`.

`sm_rw_runreport` is intended for use within a JAM/ReportWriter application. It must be called after the JAM initialization normally performed by `jmain.c` or `jxmain.c`.

Example

```
#include <rwdefs.h>

if (sm_rw_runreport("rptfile!myreport
   ('myarg1', 'myarg2') output=myoutput") == –1)
{
   sm_n_putfield ("myrwstatus", "failure");
}
```

9

Utilities

This chapter describes the utilities supplied with ReportWriter. Each description contains a synopsis of the command, including a listing of available keywords and arguments, and a description of the utility's operation.

Typographical conventions used here are listed in the Preface.

The following utilities are supplied with ReportWriter:

- `dev2bin` — Compile a device configuration source file.
- `r2asc` — Obtain an ASCII printout of a report file.
- `rinherit` — Update inheritance for screens and reports.
- `rw6to7` — Update a ReportWriter6 file to ReportWriter7 format.
- `rwrun` — Run ReportWriter.

Your distribution includes these executables and `jamdev`, the JAM development environment with JDB and ReportWriter linked in.

Source code is provided for `rwrun` in `rwmain.c` so you can customize this utility. The makefile supplied with your JAM distribution or database driver makes both `rwrun` (the stand–alone ReportWriter) and `jamdev` (JAM with ReportWriter and the screen editor). Use this makefile also if you want to make `jam`, the application-mode executable, with ReportWriter linked in. Each makefile includes instructions.

dev2bin

Compiles a device configuration file

`dev2bin` *[* `–e` *ext] filename*

filename	Specifies the name of the device configuration file. If the name does not include an extension, `dev2bin` looks first for *filename*`.dev`. If that file cannot be opened, it attempts to open *filename* (with no extension).
`–e` *ext*	Specifies the extension for the device binary file. If the `–e` option is omitted, the resulting file is named *filename*`.bin`.

Description

`dev2bin` produces a device binary file from the device configuration file identified by *filename*. The output of `dev2bin` is a binary file named *filename*`.bin` or *filename*`.`*ext* if the `–e` option is specified.

A report accesses a device configuration file it by specifying it as an invocation option.

Device configuration files specify formatting options for reports generated in ASCII text format. For more information on these formatting options, refer to Appendix C.

r2asc

Produces an ASCII file of the report binary

`r2asc –a`*[* `cf` *]* *ascii-file* *[* `–i` *header-file]* *report...*

`r2asc –b`*[* `f` *]* *ascii-file*

ascii-file
With the `–a` option, *ascii-file* is the name of the file to receive the ASCII version of *report*. With the `–b` option, *ascii-file* is the name of the file to convert into a report file.

report
Specifies the file name of a report to convert to ASCII.

`–a`
Creates an ASCII listing of one or more reports.

`–b`
Creates or extracts all binary reports from an ASCII listing. Note that this option does not accept an output file name.

`–c`
Do not generate comment lines (`–a` option only).

`–f`
Overwrites an existing file.

`–i` *header-file*
Includes *header-file* at the beginning of the ASCII output.

Description

`r2asc` produces either an ASCII listing of an existing report file or a binary report file from the ASCII file, depending on which options are specified. With `r2asc`, either the `–a` or `–b` option must be used. With `–a`, you must specify the name of at least one report (or use wildcard characters). With `–b`, report names are ignored. The `–b` option automatically extracts all report files from *ascii–file*.

ASCII output

The text file generated by `r2asc` describes the contents of the report—the widgets that compose it and their respective properties. It is broken into sections by object type, starting with the report itself, then any table views, followed by the fields of the screen in numerical order, and finally the labels, boxes, and areas in the layout window. Nodes are listed within the `R:` (report) section). Each object within the object types begins with its own header:

`R:` *reportname*
`T:` *tableviewname*
`F:` *fieldname*
`L:` *labelname*
`B:` *boxname*
`A:` *areaname*

The report structure is listed following the `RW–SCRIPT` keyword. Comments appear in lines beginning with the # character.

There are two types of keywords describing object properties, flags and values:

- A flag keyword is by itself and requires no other information—for example the `NUMERIC` keyword represents the numeric field type property and needs no value. A flag keyword can appear on the same line as other keywords.
- A value keyword must be accompanied by more information—it is followed by an equals sign (=) and a value represented by another keyword or a number or string. For example, `WIDGET_TYPE=OUTPUT–ONLY` indicates that this object is a dynamic output widget. Value keywords that begin with `PI` describe graphical properties of an object; ones that begin with `DBI` describe properties used by the database driver or by transaction manager.

Errors

The following list describes possible errors, their causes, and the corrective action to take:

`File already exists; use ’–f’ to overwrite.`

Cause: The specified output file already exists in the current directory.
Action: Add the `–f` option in order to overwrite the file.

`Invalid file format`

Cause: The file specified as the report is not a binary file.
Action: Specify the ASCII file first, followed by the name of the report binary file.

`Line #, token = <string>: unknown keyword.`

Cause: The keyword listed in the warning message cannot be processed by `r2asc`.
Action: Edit the ASCII file to the correct value and regenerate the binary report file.

`Must use option –a or option –b.`

Cause: ReportWriter could not determine whether you wanted ASCII or binary output.
Action: Specify either the `–a` or the `–b` option.

`Unrecognized section`

Cause: The sections must be identified by one of the specified labels.
Action: Edit the ASCII file to the correct value and regenerate the binary report file.

rinherit

Updates or reports on the inheritance of properties for screens and reports

```
rinherit [ –r repository ] [ –vlevel ] [ –u ] filename...

rinherit [ -rrepository ] [ -vlevel ] [ –u ] –llibrary member...
```

filename — Specifies the name of a JAM screen or report; more than one filename can be included. If the filename does not exist or is not of the correct type, it is skipped.

`–l` *library* — Specifies the name of the screen library when updating or reporting on individual library members.

member — Specifies the member of a screen library; more than one member can be included. If the library member does not exist, it is skipped.

`–r` *repository* — Specifies the name of the repository. If `–r` is not specified on the command line, `rinherit` looks first for the value of `SMDICNAME`, then for the repository `data.dic` in the current directory. If it cannot find a repository, `rinherit` reports an error.

`–u` — Updates the files in addition to listing the differences.

`–v` *level* — Specifies the level of reporting desired when running the utility, where *level* can be one of the following values:

0 No reporting.

1 List screens and reports as they are processed (the default setting).

2 List screens, reports, and widgets as they are processed.

3 List screens, reports, widgets, and properties as they are processed.

Description

`rinherit` updates the inheritance of properties for reports, application screens or a JAM library. When specified with the `–u` option, `rinherit` updates properties in the reports and screens to match the values in the repository. If specified without the `–u` option, `rinherit` only reports on the differences in property values between the screens and the repository.

Inheritance is updated each time you open a report or screen in the editor and then save it. `rinherit –u` performs this operation in batch mode, opening the specified files and saving them.

When `rinherit` opens the specified file, it looks for widgets having the Inherit From property set to a repository entry, for example, `titles!title_id`. For those widgets, it compares the inherited property values with the values in the repository. The properties that have inheritance disabled are ignored.

It also checks the Inherit From property for each screen and report to see if inheritance is designated. If it is, it compares the values in the inherited properties with the corresponding values in the repository.

Errors

The following list describes possible errors, their causes, and the corrective action to take:

`No repository is open.`

Cause: The `SMDICNAME` variable is not set, or the `data.dic` repository cannot be found in the current directory.
Action: Verify the directory location, set the repository on the command line, or set the `SMDICNAME` variable.

`Not a JAM repository.`

Cause: File specified after the `–r` option was incorrect.
Action: Check the spelling and location of the specified repository.

`Unable to inherit property` *property_name* `for` *object_id*

Cause: The object listed in the Inherit From property cannot be found in the current repository.
Action: Make sure the correct repository was specified.

`Unable to open JAM library.`

Cause: Unable to find the specified library.
Action: If the library is not in the current directory, include the pathname.

`Unable to open JAM repository.`

Cause: Unable to find the specified repository.
Action: Check the spelling and location of the specified repository. If the repository is not in the current directory, include the pathname.

`Verbosity (–v) must be 0, 1, 2, or 3`

Cause: An invalid value followed the `–v` option.
Action: Supply one of the listed values in the command line.

rw6to7

Updates a report binary file to ReportWriter 7 format

`rw6to7` *[* `–fm` *] rw6-file rw7-file*

`–f`	The output file can overwrite an existing file.
`–m`	Merges included files into the main report.

Description

`rw6to7` updates a ReportWriter6 report file to ReportWriter7 format.

For detailed information about upgrading, refer to Appendix A.

Errors

The following list describes possible errors, their causes, and the corrective action to take:

```
Found use of AUTO_KEY in source report. Ignoring.
Found screen entry function in source report. Ignoring.
Found screen exit function in source report. Ignoring.
```

Cause: Under ReportWriter 6, reports are JAM screens, but under ReportWriter 7 they are not. The AUTO key definition and screen entry and exit functions, which might have been used in ReportWriter 6 report format screens, are not applicable to the resulting report module.

Action: If these functions contain processing required for the report, use call nodes to invoke the desired functions at appropriate points during report generation.

```
Translating FLOAT keyword to the group node property
’Floating Footer = Yes’. The change in scope might affect
output.
```

Cause: The `float` keyword is applied inconsistently to two or more page footer areas in the source report. `rw6to7` has set the Floating Footer property to Yes for the corresponding page format node, thus applying this format to all layout areas associated with the page footer.

Action: ReportWriter 7 does not support both floating and non-floating footers on the same page. Make adjustments to the content of the associated layout areas. For additional information setting on page format properties, refer to page 71.

```
Translating NODUPL keyword to the group node property ’Local
Header Only = Yes’. The change in scope might affect output.
```

Cause: The `nodupl` keyword is applied inconsistently to two or more break header areas in the source report. `rw6to7` has set the Local Header Only property to Yes for the corresponding group node, thus applying this format to all layout areas associated with the header.

Action: If this is not the desired behavior, create separate group nodes and apply the desired properties to each individually. For additional information, refer to page 82.

```
Translating NOORPHANBREAK keyword to the group node property
’Footer Has Totals = Yes’. The change in scope might affect
output.
```

Cause: The `noorphanbreak` keyword is applied inconsistently to two or more break footer areas in the source report. `rw6to7` has set the One Detail Footer property to No for the corresponding group node, thus applying this format to all layout areas associated with the footer.

Action: If this is not the desired behavior, create separate group nodes and apply the desired properties to each individually. For additional information, refer to page 82.

```
Translating SHOWATTOP keyword to the group node property
’Running Header = Yes’. The change in scope might affect
output.
```

Cause: The `showattop` keyword is applied inconsistently to two or more break header areas in the source report. `rw6to7` has set the Running Header property to Yes for the corresponding group node, thus applying this format to all layout areas associated with the header.

Action: If this is not the desired behavior, create separate group nodes and apply the desired properties to each individually. For additional information, refer to page 82.

rwrun

Runs ReportWriter

`rwrun` *filename* [`!`*reportname*] [`(`*arg*[`,` ...] `)`] [*option*]...

filename
Specifies the name of a report file.

reportname
If *reportname* is supplied, the specified report in this file is invoked. If *reportname* is omitted, the first report in the file is generated.

arg
You can supply one or more arguments to the report. Each argument can be either a string within quotation marks, a number, or the name of a JAM variable to be evaluated when the report is run. Each argument must also have a parameter defined in the report node's Parameter property, and the order of the arguments in this command must match the order specified in that property.

Note that any punctuation having a special usage in the operating system such as parentheses `()`, must be prefixed with an escape character (`\`)

option
You can specify one or more options for report generation. Table 4 lists available options. For more detailed descriptions, refer to page 136.

Table 4. Report invocation options.

Invocation option	Description
`output =` *output-file*	Name of the output file.

`driver =` *driver-name*	Type of report output, where *driver-name* can be one of the following:
	`postscript`
	`text`
	`windows`
	`macintosh`
	`rwmetafile`.

`device =` *device-file*	Name of the device configuration file (text driver only).
`printer =` *printer-name*	Name of the printer (Windows driver only).
`papersize =` *papersize*	Size of the paper to use for the report, for example `papersize = 8.5inx11in`

Invocation option	Description
`leftmargin =` *margin* `rightmargin =` *margin* `topmargin =` *margin* `bottommargin =` *margin*	The amount of space reserved for the report's margins. Specify the amount of space in this format: *margin-space [unit-spec]* For valid units of measurement (*unit-spec*), refer to page 133.
`spool =` *spool-command*	Command to use to send the report to an output device (UNIX only).
`overwrite`	Overwrite the output file if it exists.
`portrait | landscape`	Determines the page orientation of the report.
`showwarnings | nowarnings`	Determines if warning messaged are displayed.

Description

`rwrun` invokes ReportWriter to execute the specified report. The following requirements apply when using `rwrun`:

- Set the output file on the command line. Otherwise, the report is displayed in standard output on the screen. Generally, the output file is specified as one of the invocation options.
- Control the database connection within the report file. Generally, a call node attached to an instance node calls the JPL procedure or C function which connects to the database, and a final call node calls the procedure or function which disconnects from the database.

The following example runs the `custinfo.jrw` report for customer ID `11` and writes the report to the file `custinfo.txt`.

```
rwrun custinfo.jrw \(11\) output=custinfo.txt
```

A

Upgrading To ReportWriter 7

With ReportWriter 7, the binary file that defines the report can now be edited directly. The editing environment provides two views into the report:

- The layout window defines the format and content of report output, replacing the report format screen of earlier versions. Report areas are called *layout areas* and area name tags are phased out completely and replaced by layout area widgets.

- The report structure window provides a graphical representation of the report structure, replacing the report script. Processing and formatting options previously specified by script keywords are now properties of the applicable nodes in the Report Structure window.

The `rw6to7` utility converts existing ReportWriter 6 reports to ReportWriter 7 format. Refer to page 153 for a description of this utility. This appendix describes the editing environment, conversion issues, and new functionality.

Developing Reports

ReportWriter's development environment takes full advantage of the JAM user interface. To familiarize yourself with the JAM screen editor, refer to the JAM *Editors Guide*.

The sections below highlight major innovations that ReportWriter 7 provides in report development. Refer to the main body of this manual for more complete information.

Layout Window

The layout window defines the format and content of report output. Three widget types are unique to this window: layout area widgets, dynamic output widgets, and static output widgets.

The layout window can also include other widgets: graph widgets let you present data as a pie or bar graph; box and line widgets can be used to enhance a report's appearance. Table view and link widgets can also be included to enable JAM's transaction manager to fetch report data.

All other JAM widget types are disallowed. `rw6to7` converts invalid types such as single line text and static labels to dynamic or static output widgets, depending on their type. For more information about type conversions, refer to page 55.

Layout Area Widgets

Report areas are now known as layout areas. Each layout area widget serves as a boundary for its corresponding area and also specifies that area's name.

For more information on layout area widgets, refer to page 51.

Output Widgets

Two widget types are responsible for most report output:

- Dynamic output widgets get their data at runtime—for example, from a database or from other widgets.

- Static output widgets have their data set in the editor; their data remains constant.

For more information on report output widgets, refer to page 53 .

Report Structure Window

The report script of earlier versions is replaced by a graphical representation in the report structure window. This window shows the report structure schematically, using various types of nodes to represent report elements and processing.

The table below shows how these nodes correspond to ReportWriter 6 script components.

ReportWriter 7 node type	Corresponding ReportWriter 6 statement, clause, or keyword	More...
Structure nodes:		
Report	`<<begin report>>`	p. 69
Page Format	`page`	p. 70
Instance	`insert`	p. 89
Detail	`detail`	p. 73
Group	`break`	p. 77
Action nodes:		
Print	`area`	p. 77
Call	`call`	p. 88
Subreport	`report`	p. 84
End Page	`forcepage`	p. 118

For more information, refer to page 59.

Format and Processing Specifications

In ReportWriter 6, various formatting and processing options were specified in the report script as clauses or keywords. In ReportWriter 7, these options are specified as properties of the applicable report nodes.

In general, you create a node that corresponds to the script statement or clause you would have used under ReportWriter 6. Then set the properties of the node to specify further levels of detail, such as the data source for a detail node, formatting options for a print or subreport node, or the invocation string for a call node.

For example:

- To use the transaction manager to fetch data, create a detail node and set its Data Source property under Identity to TM View. This is equivalent to using the `tm` clause in a `detail` statement under previous ReportWriter versions.
- To specify the query for a data fetch, create a detail node and set its Data Source property under Identity to SQL Query. Enter the desired SQL

statement in the SQL Statement subproperty. In ReportWriter 6, this is equivalent to using the `query` clause in a `detail` statement.

- To eliminate excess blank lines in a report area, set the Shrink property of the corresponding print node to Yes. This is equivalent to using the `shrink` keyword in an `area` statement.
- To keep data for a break group together on one page, set the Keep on Page property of the corresponding group node to Yes. This is equivalent to using the `nosplitgroup` keyword in a `break` statement.

When reports developed under ReportWriter 6 are converted by the `rw6to7` utility, node properties are set to correspond to the keywords and values specified in the script. In a few instances, an exact conversion cannot be performed automatically. `rw6to7` displays a warning message when this occurs. Refer to page 161 for more information.

Converting Existing Reports

Use the `rw6to7` utility to convert reports developed under version 6 of ReportWriter. The output of this utility is a ReportWriter 7 report file, which is both executable and editable. Refer to page 153 for a description of `rw6to7`.

Included Files

ReportWriter 6 processed inclusions in its compiler, `rprt2bin`. ReportWriter 7 processes inclusions at runtime. Reports that contain included files can be converted in either of the following ways:

Convert the main report and its inclusions into separate files.
Locate the main report and all its included files and convert each one with `rw6to7`. Then modify the main report file in one of two ways:

- Edit the Report Files property (refer to page 90) to include the names of each included report file.
- In the report structure, edit each subreport node's Report Invocation property so that it names the desired report and the file that stores it with this syntax:

  report-filename `!`*report-name*

  Refer to page 90 for more information about invoking external files as subreports.

Merge the main report and its inclusions into a single report file.

- `rw6to7`’s `–m` option automatically searches `SMPATH` to find inclusions. The utility converts included subreports into internal subreports; these appear in the layout and structure views of the report file.

Widget Name Extensions

`rw6to7` automatically converts all alias widgets that use the ReportWriter 6 widget naming convention of *base-name*`.`*extension*. `rw6to7` converts each alias widget to a copy widget—that is, dynamic output widget whose Value property is set to Copy—and sets its Value Source property to *base-name*. ReportWriter no longer supports this naming convention as a means copying output from one widget into another. For more information, refer to page 99.

Calling Sequences

ReportWriter complies with JAM 7 syntax conventions for calling functions (refer to page 47 in the *Language Reference*). However, the syntax of report invocations in earlier versions continues to be supported. For future development, you should use current syntax conventions.

Errors

Conversion errors fall into two general categories:

- Header or footer `area` clauses in the original report contain mutually inconsistent formatting keywords.

- An AUTO control string, a screen entry function, or a screen exit function is associated with the original report format screen.

In each case `rw6to7` displays a warning message to alert you to a possible problem.

Header and Footer Area Properties

If the header or footer contains multiple `area` clauses and these are not all modified by the same formatting keywords—`showattop`, `nodupl`, `noorphanbreak` and `float`—the resulting output might be different than anticipated, and ReportWriter issues an appropriate error message. This occurs because ReportWriter 6 applies these formatting keywords to individual output areas, but ReportWriter 7 applies the corresponding properties to the group node or page layout node as a whole, and, therefore, to all output associated with that node.

In the case of break headers or footers, you might want to split the group node into two, each with its own properties. In the case of page footers, you will probably need to make adjustments to the content of the associated layout areas.

Screen Functions and the AUTO Control String

If an AUTO control string, a screen entry function, or a screen exit function is associated with the original report format screen, it is ignored when the report is converted. `rw6to7` displays an appropriate warning message.

New Features in ReportWriter 7

Although the user interface has a completely new look, all the functionality of ReportWriter 6 has been retained in ReportWriter 7. In addition, the following new features have been added:

- Wizard-generated reports. Refer to page 21.
- Total (page 96) and history widgets (page 100).
- Context–sensitive summary data. Refer to page 96 for information on automatically calculating and resetting column and other totals.
- External subreports. Refer to page 84. Note that external, internal, and included subreport invocations can be intermixed within a report.
- Output viewer that is fully integrated with the editing environment.
- Full integration with JAM, including support for:
  - Business graphics
  - Pixmaps
  - Fine control over widget placement
  - Fonts

B

Vbizplus Database

The vbizplus database extends and modifies the JAM videobiz database. This appendix describes the tables in the vbizplus database by listing the following information for each table:

- Column names.
- Data type of each column.
- Length of character columns.
- Status of column detailing whether it is a primary or foreign key and whether it can accept null values.
- Description of the data to be entered into the column.
- Sample entry.

Vbizplus Schema

The following tables outline the database tables in the `vbizplus` database. A diagram of the schema appears in Figure 45 on page 172.

Table 5. Actors table.

Column Name	Data Type	Length	Status	Sample	Description
actor_id	integer		primary key not null	87	Unique number code for each actor.
last_name	char	25	not null	Ullmann	Actor’s last name or only name.
first_name	char	20		Liv	Actor’s first name.

Table 6. Codes table.

Column Name	Data Type	Length	Status	Sample	Description
code_type	char	32	primary key not null	genre_code	Type of code. Corresponds to column name.
code	char	4	primary key not null	ADV	Code value.
dscr	char	40		Adventure	Description of code value.

Table 7. Customers table.

Column Name	Data Type	Length	Status	Sample	Description
cust_id	integer		primary key not null	10	Unique number code for each customer.
last_name	char	25	not null	Stephens	Customer’s last name.
first_name	char	20	not null	Darrin	Customer’s first name.
address1	char	40		937 Brewster	Customer’s address.
address2	char	40			Additional address information.
city	char	25		Geneva	City customer lives in.
state_prov	char	10		NY	State/Province.
postal_code	char	10		10234	Postal code.
phone	char	15		515–555–5668	Customer’s telephone number.
cc_code	char	4		MAST	Code for type of credit card. List in `codes` table.
cc_number	char	16		5000...	Number on credit card.
cc_exp_month	integer			10	Month of credit card expiration. 1=January, 12=December.
cc_exp_year	integer			1997	Year of credit card expiration (4 digits).
member_date	datetime			1996/01/19 00:00:00	Date when customer became a member.
member_status	char	1	not null	A	Current status of membership. Values include: (A)ctive, (I)nactive, (F)requent renter.
num_rentals	integer		not null	4	Total number of rentals customer has made.
rent_amount	float		not null	11.50	Total amount of money paid by customer.
notes	char	254		Notify for ADV videos.	Comments about customer.

Table 8. Distributors table.

Column Name	Data Type	Length	Status	Sample	Description
distrib_id	integer		primary key not null	1	Unique number code for each distributor.
distrib_name	char	20		Geneva	Distributor's name.
address1	char	40		4201 Washington Street	Distributor's street address.
address2	char	40		NULL	Additional address information.
city	char	25		Geneva	City in which the distributor is located.
state_prov	char	10		NY	State/Province.
postal_code	char	10		10234	Postal code.
phone	char	15		515–555–7232	Distributor's telephone number.

Table 9. Orders table.

Column Name	Data Type	Length	Status	Sample	Description
order_num	integer		primary key not null	1001	Unique number code for each order.
distrib_id	integer		foreign key not null	1	Distributor who placed the order.
order_date	datetime			1996/01/29 00:00:00	Date the distributor placed the order.
ship_date	datetime			NULL	Date the order is shipped to the distributor.
po_num	char	15		D1456	Purchase order number for the order.

Table 10. Order_items table.

Column Name	Data Type	Length	Status	Sample	Description
order_num	integer		primary key foreign key not null	1001	Unique number code for each order.
title_id	integer		primary key foreign key not null	78	Unique number code for each video title.
price	float			23.50	Price of the video.
qty	integer		not null	8	Quantity ordered.
order_flag	char	1		O	Flag indicating the status of the order: O)rdered, B)ack-ordered, C)ataloged, S)hipped

Table 11. Pricecats table.

Column Name	Data Type	Length	Status	Sample	Description
pricecat	char	1	primary key not null	N	Unique letter code for each category.
pricecat_dscr	char	40		New Release	Category description.
rental_days	integer			2	Number of rentals days available in this category.
price	float			2.50	Amount to be paid for rentals in this category.
late_fee	float			2.00	Amount of late fee for rentals in this category.

Table 12. Rentals table.

Column Name	Data Type	Length	Status	Sample	Description
cust_id	integer		primary key foreign key not null	3	Code identifying the customer for this rental.
title_id	integer		primary key foreign key* not null	69	Code identifying the video title for this rental.
copy_num	integer		primary key foreign key not null	2	Copy of this video being rented.
rental_date	datetime		primary key not null	1996/02/07 19:56:00	Date/time the video was rented.
due_back	datetime		not null	1996/02/09 00:00:00	Date the video is due back to avoid late fee.
return_date	datetime			NULL	Actual date/time the video was returned; NULL until then.
price	float		not null	2.50	Rental fee for video at time rental was made.
late_fee	float		not null	2.00	Late fee per day for video at time rental was made.
amount_paid	float		not null	2.50	Total amount paid on this rental as of current date.
rental_status	char	1	not null	C	Status of rental. Values include (C)urrently out, Back and (P)aid, (B)alance is due.
rental_comment	char	76		NULL	Comments about rental, if any.
modified_date	datetime		not null	1996/02/07 19:56:00	Date this record was last modified.
modified_by	integer		foreign key not null	2	Last user who modified record.

*title_id is a foreign key from the tapes table, in combination with copy_num.

Table 13. Roles table.

Column Name	Data Type	Length	Status	Sample	Description
title_id	integer		primary key foreign key not null	33	Unique number code for each video title.
actor_id	integer		primary key foreign key not null	87	Unique number code for each actor.
role	char	40		Marianne	Role the actor plays in the video.

Table 14. Tapes table.

Column Name	Data Type	Length	Status	Sample	Description
title_id	integer		primary key foreign key not null	33	Unique number code for each video title.
copy_num	integer		primary key not null	1	Number identifying the copy of this video.
status	char	1	not null	O	Code specifying the current status of this copy. Values include (A)vailable, (R)eserved, (O)ut, (I)nactive.
times_rented	integer		not null	53	Number of times this copy has been rented.

Table 15. Titles table.

Column Name	Data Type	Length	Status	Sample	Description
title_id	integer		primary key not null	33	Unique number code for each video title.
name	char	60	not null	Scenes from a Marriage	Video title.
genre_code	char	4		CLAS	Code specifying the video category. Values include: ADLT, ADV, CHLD, CLAS, COM, HORR, MUS, MYST, SCFI, TV, VID. See `codes` table.
dir_last_name	char	25		Bergman	Director’s last name.
dir_first_name	char	20		Ingmar	Director’s first name.
film_minutes	integer			168	Length of the video.
rating_code	char	4		PG	Rating code given the film by the Motion Picture Association of America. Values include: G, PG, PG13, R, NC17. See `codes` table.
release_year	integer			1974	Year the film was released to movie theatres.
pricecat	char	1	foreign key not null	G	Code taken from the `pricecats` table specifying the price category.
order_price	float			28.00	Price to use to order the video.
quantity_avail	integer			2	Number of video copies available.

Table 16. Title_dscr table.

Column Name	Data Type	Length	Status	Sample	Description
title_id	integer		primary key foreign key not null	33	Unique number code for each video title.
line_no	integer		primary key not null	1	Line number of the video description.
dscr_text	char	76		Relationship of a couple...	Description of the video.

Table 17. Users table.

Column Name	Data Type	Length	Status	Sample	Description
user_id	integer		primary key not null	3	Unique number code for each employee/system user.
logon_name	char	8		jack	User’s logon name.
password	char	8		go	User’s password.
last_name	char	25		Ryan	User’s last name.
first_name	char	20		Jack	User’s first name.
customer_flag	char	1		Y	Y allows access to customer subsystem.
admin_flag	char	1		N	Y allows access to administrative subsystem.
marketing_flag	char	1		Y	Y allows access to marketing subsystem.
frontdesk_flag	char	1		Y	Y allows access to front desk subsystem.

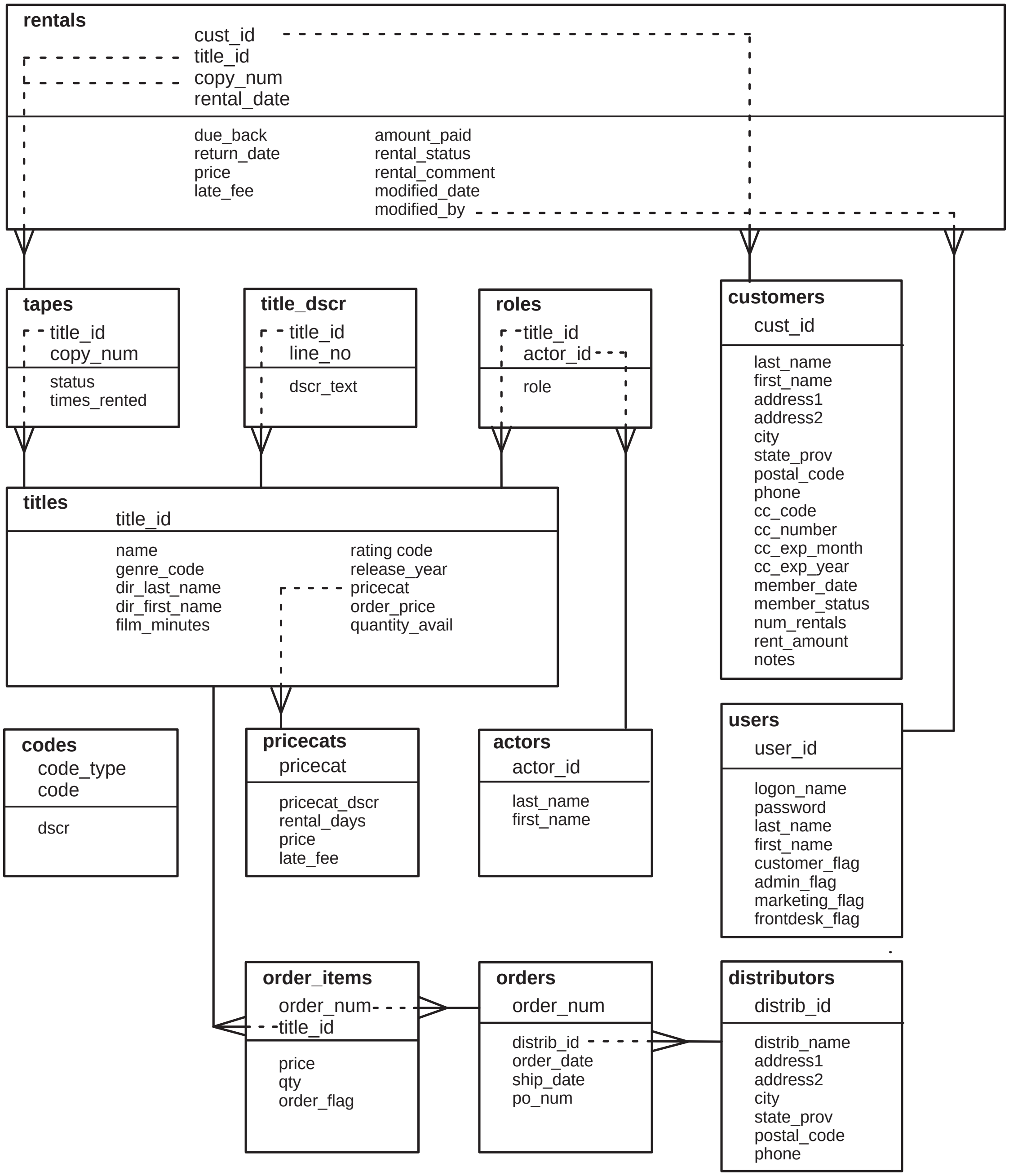


Figure 45. Diagram of the vbizplus database.

C

Device Configuration Files

ReportWriter’s support for ASCII text output can be enhanced and customized through device configuration files. These files can define control sequences for the following capabilities of an output device:

- Initialization and reset strings.
- Font specifications.
- On/off strings for bold, italic, and underline attributes.
- Directives for enabling landscape mode.
- Characters for printing lines and boxes.

The device configuration file also can provide information to ReportWriter about the device itself:

- Left margin.
- The file name or process to which output is spooled.
- Name of the developer-written output function and the size of its output buffer.

Some of these settings can be specified as properties within the report or as arguments to the report’s invocation string. Properties and invocation arguments always have precedence over their equivalent settings in the device configuration file.

Device configuration files are created and edited as ASCII source files. After editing the file, compile it with the `dev2bin` utility (refer to page 148). You specify the desired device configuration file in the report invocation string (refer to page 129).

File Format

The ASCII source for a device configuration file contains one of more statements in this format:

key-word clause = value

For example:

```
init    = ESC P
reset   = ESC Q
lines   = 55
columns = 80
spool   = lp
```

The following section describes each device file keyword and its possible values.

Keywords

`init/reset`
Specify the initialization and reset strings for the output device in this format:

```
init  = init-str
reset = reset–str
```

If an `init` string is specified, ReportWriter prefaces the report with the string. If a `reset` string is specified, ReportWriter appends it to the report.

These strings follow the conventions of JAM video file capability strings: the capabilities are given byte-by-byte, separated by spaces. Non-printable bytes and the space character are represented by their respective ASCII names—for example, NUL, NL, ESC, and SP. Any byte can instead be represented by its octal value `\`*ddd*, where *d* is an octal digit. Refer to Chapter 7 in the *Configuration Guide* for more information on how to format these capabilities.

`FontOn/FontOff`

Resolve font aliases that are defined in the configuration file's `[Text Fonts]` section as follows:

```
FontOn fontname = escape-sequence
FontOff fontname = escape-sequence
```

Refer to Chapter 7 in the *Configuration Guide* for information about JAM escape sequences.

For example, given these font alias definitions:

```
# JAM Font Name      Qualifiers  =  Text font
# -------------      ----------     ---------
JAM Times Roman      18           = TimesRomanBig
JAM Times Roman                   = TimesRoman
JAM Courier                       = Courier
```

The device file requires these corresponding statements:

```
FontOn TimesRoman     = ESC m t f 1
FontOff TimesRoman    = ESC m t f 0
FontOn TimesRomanBig  = ESC m t h 1
FontOff TimesRomanBig = ESC m t h 0
FontOn Courier        = ESC m c f 1
FontOff Courier       = ESC m c f 0
```

text-attr = string

The output device turns italics, boldface, and underlining on and off through these command strings:

```
ItalicOn     = ital-on-str
ItalicOff    = ital-off-str
BoldOn       = bold-on-str
BoldOff      = bold-off-str
UnderlineOn  = under-on-str
UnderlineOff = under-off-str
```

`spool =` *spool-cmd*

spool-cmd is the name of a program or file. On UNIX systems, output is piped to the specified program. On other operating systems, the output is written to the specified file.

`procedure =` *outproc*

outproc is the name of a user-written procedure that writes or filters output. For more information about writing output procedures, refer to page 179.

`obuffsize =` *bufsize*

bufsize is the size of the output buffer, in bytes, used by the function *outproc*. If not specified, the default size is 256 bytes.

`leftmargin =` *nblanks*

nblanks specifies the number of blank spaces to left-indent all output. These spaces must be included in the line length *ncols*. This parameter can also be specified in the Indent property of a print or subreport node. The default setting is 0.

`feedlines =` *nflines*

nflines is the number of line feed characters that should be used to separate pages. If specified, *nflines* plus *nlines* must equal the physical length of the page.

If *nflines* has a value less than 0 or this statement is absent, ReportWriter outputs a form feed to begin the next page. If the value of *nflines* is greater than or equal to 0, the form feed is suppressed. The default value of *nflines* is –1.

The `feedlines` parameter can also be specified in the report invocation string.

`landscapeon/landscapeoff`

These keywords enable landscape mode on an output device:

`landscapeon  =` *landscapeon-str*
`landscapeoff =` *landscapeoff-str*

- *landscapeon-str* is output at the beginning of a page printed in landscape mode.
- *landscapeoff-str* is output at the end of a page printed in landscape mode.

Note: *The presence of these capabilities does not by itself enable landscape mode in a report. To print a report in landscape mode, either the report invocation or the report itself (through its Page Format node) must specify landscape orientation.*

graphic-type = char-spec

Use the following keywords to specify the characters used to print lines and boxes:

Graphic type keyword	Usage
`horizseg`	Horizontal line segment.
`vertseg`	Vertical line segment.
`topright`	Top right box corner.
`topleft`	Top left box corner.
`bottomright`	Bottom right box corner.
`bottomleft`	Bottom left box corner.

`fixedlength`

Specifies to pad the end of each line with enough spaces to produce output of uniform width., based on the number of columns specified for report width. If `fixedlength` is not specified, ReportWriter outputs variable-length lines.

D

Output Procedures

This appendix explains how to write your own procedure to write or filter ReportWriter output to a text driver. This function must be installed as a prototyped function; for more information on installation, refer to page 121 in the *Application Development Guide*.

To use an installed output function, specify it in the device file’s `procedure` statement. Use the `obuffsize` statement to specify a buffer size greater than the default of 256 bytes. Refer to page 173 for more information on device configuration files.

Arguments

The output function is declared with two parameters in this order: an integer-type code, and an output buffer.

Integer type code

This parameter can be supplied one of the following arguments:

```
RW_P_OPEN
RW_P_CLOSE
RW_P_WRITE
```

ReportWriter calls the function once with `RW_P_OPEN`, once per line of output with `RW_P_WRITE`, and once with `RW_P_CLOSE`.

Output buffer

The second argument is an output buffer:

- When the type code is `RW_P_WRITE`, the buffer contains a line of output, terminated by the NEWLINE and NULL characters. Your procedure can modify the contents of the buffer.
- When the type code is `RW_P_OPEN`, the buffer contains the initialization string, as specified in the device file; the string is terminated by the NULL character.
- When the type code is `RW_P_CLOSE`, the buffer contains the reset string, as specified in the device file; the string is terminated by the NULL character.

Note: *Because the initialization and reset strings are terminated with the NULL character when passed to the output function, neither can contain this character as part of the string.*

Return Values

Custom output procedures should return one of the codes listed in the following table.

Return code	Effect
`SM_RW_OK`	ReportWriter should output the buffer; the developer-written procedure has only filtered or analyzed the data.
`SM_RW_DIDOUTPUT`	ReportWriter should not output the buffer; the developer-written procedure has handled this step.
`SM_RW_ERROR`	Request for ReportWriter to abort the rest of the report.

Invoking the Output Procedure

To use a custom output procedure with ReportWriter, specify the procedure's name in the `procedure` parameter of the device configuration file. The default size for the output buffer is 256 bytes. You can specify a different size through the `obuffsize` parameter of the device configuration file.

Index