QAD Enterprise Applications
Enterprise Edition

User Guide

QAD Reporting Framework

70-3092A
QAD Enterprise Applications 2011
Enterprise Edition
March 2011

QAD Inc. provides this material as is and makes no warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. QAD Inc. shall not be liable for errors contained herein or for incidental or consequential damages (including lost profits) in connection with the furnishing, performance, or use of this material whether based on warranty, contract, or other legal theory.

ReportingFramework_UG_v2011EE.pdf/sti/sti

QAD Inc.
100 Innovation Place
Santa Barbara, California 93108
Phone (805) 566-6000
http://www.qad.com

Contents

Chapter 1

Welcome to the QAD Reporting Framework

Introduction to the QAD Reporting Framework

Reports help you analyze and interpret important information. The QAD Reporting Framework makes it easy to create simple reports, and it also has the comprehensive tools you need to produce complex or specialized reports.

Multiple Data Sources

The QAD Reporting Framework is designed to produce the report you want from a range of data sources. You can extract data from Progress databases, browses, or through the QAD Financials API for reporting purposes.

Powerful and Flexible Report Authoring

The QAD Reporting Framework offers you both simplicity and flexibility in creating your reports.

The built-in Report Wizard guides you step by step through building basic reports and completing common reporting tasks. A rich set of report design tools lets you create more complex reports tailored to your specific requirements.

Columns, groups, calculated fields, subreports, and formatting help make sense of data and uncover important relationships that might otherwise be hidden.

Multiple Output Formats

The flexibility of the QAD Reporting Framework does not end with creating reports. Your reports can be published to a variety of outputs including printers, PDFs, and Excel files.

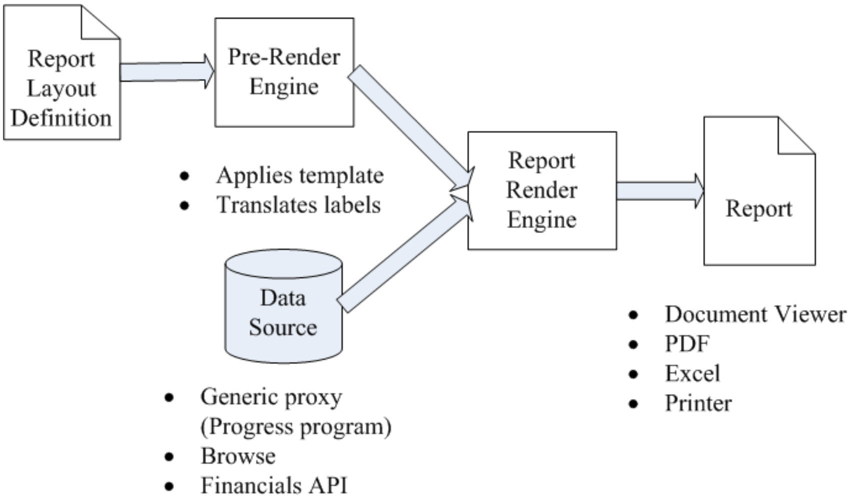
Report Scheduling

You can schedule the system to run your reports automatically at a certain time or at a specified interval and send scheduled report outputs to your desired destination, such as a printer or the document service on the report server. You can also have the system notify you that your scheduled reports have run.

QAD Reporting Framework Architecture

The QAD Reporting Framework contains five key components: report render engine, report layout definition, data source, pre-render engine, and report. The following diagram illustrates the QAD Reporting Framework architecture at a high level.

Fig. 1.1
QAD Reporting Framework Architecture



- Data source

  Data to be displayed on the report are queried from the underlying business system through the data source definition. The QAD Reporting Framework supports three types of data sources: generic proxy (Progress program), browse, and Financials API. For detailed information on the three types of data sources, see “Choosing a Report Data Source” on page 11.

- Report layout definition

  Report layout definition defines what gets displayed on the report, and where. You use Report Designer in the QAD .NET UI to define the report layout in WYSIWYG (What You See Is What You Get) fashion. You can also import and export report layout definitions as XML files, which makes it very easy for you to deploy reports or migrate them between systems, such as moving reports from the test environment to the production environment.

- Pre-render engine

  The pre-render engine pre-processes the report layout definition by applying a report template to it as well as performing label translations and produces a modified report layout definition. The resultant report layout definition along with the data source are then fed into the report render engine, which generates the actual report.

- Report render engine

  As the core of the solution, the report render engine takes in data set and report layout definition as inputs, then renders and produces the report output. Since the QAD Reporting Framework is a .NET solution, the report render engine can only run on the Windows operating system.

  The rendering process takes place on the computer that actually runs the report. If you run a report in the QAD .NET UI on your PC, then your PC’s CPU power is consumed to render the report, which helps to distribute the processing load across client machines.

- Report

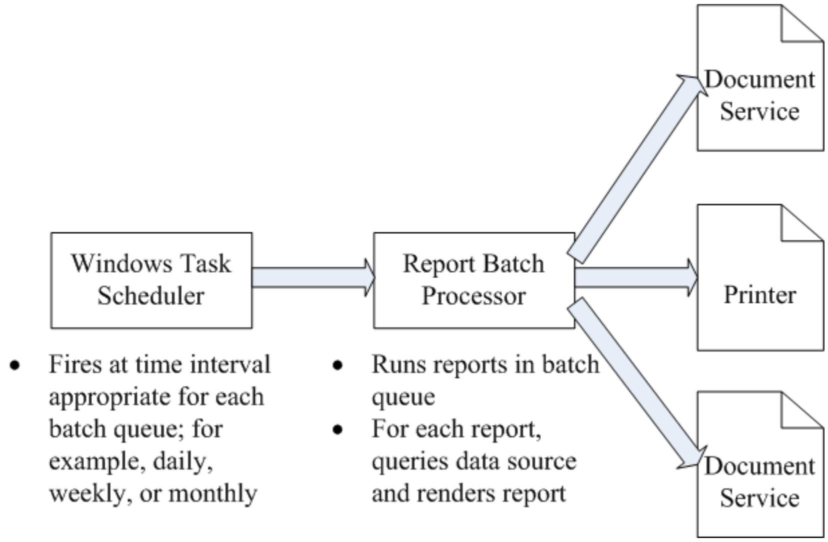
  The report output can be rendered in three different formats, depending on your preference. The default format is a document displayed on the screen, which you can view by paging and zooming. You can then send it to printer if you want a hard copy. Alternately, the report can be exported into the PDF or Excel format, which you can print or save as a file.

QAD Report Server Architecture

Reports can also be run on the report server. In this operational mode, the report render engine is run as part of a batch process that runs on the server. Reports in a batch are generated periodically during each batch run, such as in a daily, weekly, or monthly batch. Typically, you use the Windows Task Scheduler on the server to periodically fire off the running of a batch through a command line.

Any report run from the server can be sent to the printer of your choice, or an output file, typically a PDF file, that is stored on the QAD .NET UI web server, which can be accessed from a URL. Optionally, you can have the system send you an e-mail notification every time a report is run.

Fig. 1.2
QAD Reporting Framework Architecture


Windows Task Scheduler
Report Batch Processor
Document Service
Printer
Document Service
• Fires at time interval appropriate for each batch queue; for example, daily, weekly, or monthly
• Runs reports in batch queue
• For each report, queries data source and renders report

You can have any number of Windows report servers that can process report batches. If you have a batch with a lot of reports in it that runs frequently, you can have several servers pointing to the batch, which balances the processing load as well as serves as a fail-safe redundancy.

Understanding Report Design

It is advisable to take a structured approach to preparing a report. This approach includes the following elements:

- Deciding on the content of the report
- Developing a prototype on paper

This section is designed to provide a conceptual understanding of the reporting process.

Deciding on the Content of the Report

Before you do anything else, you should outline the information you want the report to provide. The following sections provide a guide to making that outline.

Stating the Purpose

What is the overall purpose of the report?

Reports are management tools. Their purpose is to help you quickly grasp the essential elements and relationships found in raw data, to help you make effective decisions. For a report to be effective, it has to present the correct data in a logical way. If it presents the wrong data, or if it presents the right data in a haphazard manner, the report may slow the decision-making process or may even lead to incorrect decisions.

A good starting place in the development of a report is to write out the purpose of the report in a sentence or two. The purpose statement helps you focus on your primary needs, and it gives the report both a starting point and a goal. Here are some examples of purpose statements:

- “The purpose of this report is to show monthly and year-to-date sales by sales representatives, compare this year’s numbers to last year’s, and flag representatives whose sales figures do not meet company standards.”
- “The purpose of this report is to show sales activity for each item in inventory, and to suggest reorder quantities based on that activity.”

Defining the purpose of the report before you start is a critical step in the overall process.

Who is going to read the report?

A single report is often used by many individuals. A detailed, company-wide sales report, for example, may be used by sales representatives, the regional sales manager, the national sales manager, and the Chief Operating Officer (COO).

These individuals will be interested in different aspects of the report:

- A sales representative will use the report to evaluate individual sales performance and compare this performance to that of other representatives in the region.
- The regional sales manager will use the report to evaluate regional representatives and compare the region’s performance to that of other regions.
- The national sales manager will use the report to evaluate the performance of regional managers and compare overall sales to the current sales forecasts.
- The COO will use the report to evaluate the performance of the Vice President of Marketing and the sales department as a whole, and to project such things as manufacturing needs and warehouse locations.

Since each user of the report has different interests, it is important to plan the report so it includes the information each user is looking for.

Determining the Layout of the Report

What is the report title going to be?

Write out a working title for the report. You may decide to change it later, but at least you will have a title to use when creating the prototype report.

What identifying information is needed in the header and footer?

You may wish to include the print date, information on who prepared the report, a block of text to describe the purpose of the report, the range of data covered, or something similar. If you are going to include such information, write it down so you can use it in preparing your prototype. The information can come from a variety of sources, depending on the kind of information you plan to use.

- Information on who prepared the report might be drawn from individual data fields in the database tables used. If it is to be drawn from a database table, what table? Or, what combination of tables?
- A block of text can be created as a text object and placed anywhere on the report.
- The QAD Reporting Framework can generate information such as the print date or page numbers.

Finding the data

What data do you want to use in the report?

Do you know the type of database you are reporting from? Will you be reporting off a browse or a database table? Are you familiar enough with the data to find the necessary information? When looking for a Customer ship-to address, can the field be found in a database table? If not, seek help from someone who is familiar with the system database.

What specific data should appear in the body of the report?

The body should contain all the data needed to fulfill the statement of purpose you wrote for the report. It should also contain all of the data needed by the various users that you have identified.

This step requires you to look at the available database tables. The QAD Reporting Framework allows you to combine data from different databases when you create reports, so you have a great deal of flexibility in your work.

- Much of the data in a typical report is taken directly from data fields. Which data fields will be used, and where are they located?
- Other data will be calculated based on data fields. Which data fields will be used in the calculations?
- Still other data will be placed directly into the report using text objects (headings, notes, labels, and so on).

Does the data exist or does it need to be calculated?

Some report information can be drawn directly from data fields (sales information, for example); other information will have to be calculated based on data field values (for example, sales commission, based on the relationship of sales to quota). In your planning, it can be helpful to segregate or flag data that needs to be calculated from data that can be used directly.

What types of fields contain data?

You should take the time to get to know the data type for data fields that will be used in your calculations. Since formula functions and operators work with specific kinds of data, it is important to recognize the data type you are working with, before you start any calculations. For example, some functions require numeric data, while others work with only string fields.

Developing a Prototype on Paper

While a paper prototype is useful regardless of your level of expertise with the QAD Reporting Framework, it is particularly valuable when you are first learning the system. With the paper prototype in hand, you can put your full effort into learning and using the functions, rather than into trying to design and learn at the same time.

To design a paper prototype:

1. Get the same size paper you will be using for the finished report.
2. Position the title and other descriptive header information, using boxes or lines to represent report elements.
3. Position the footer information.
4. Review the page layout for balance.
5. Look at the information you intend to include in the body of the report:
   - Count the number of fields being used and estimate the appropriate spacing between fields.
   - Use rectangles to pencil in the fields within the estimated spacing.
   - Change the spacing if you need to.
   - Decide on a logical sequence for presenting the data in the body of the report.
   - Label the fields to indicate that sequence.
6. Use small boxes to indicate group values and totals.
7. Darken any elements you want highlighted to make them stand out from the rest of the prototype.
8. Review the finished product for layout and balance, and make changes as needed.

Chapter 2

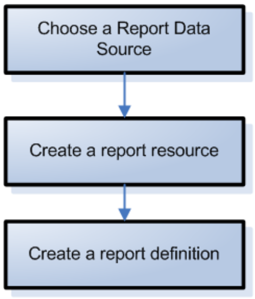
Creating a Basic Report

Basic Report Creation Workflow

The QAD Reporting Framework makes it very easy and straightforward to build a basic report. It takes just a matter of minutes to generate your first basic report. Report Wizard provides you with step-by-step instructions to build a basic report. Built-in report templates take care of most of the reporting layout and formatting for you.

Creating a basic report consists of three main steps.

Fig. 2.1
Basic Report Creation Workflow



Report Resource, Report Definition, and Report

Report resource, reporting definition, and report are a set of interrelated concepts in the QAD Reporting Framework.

A **report resource** represents a unique, cross-domain report object that contains report metadata, report definitions, report data source definitions, filter definitions, report parameters, and report settings.

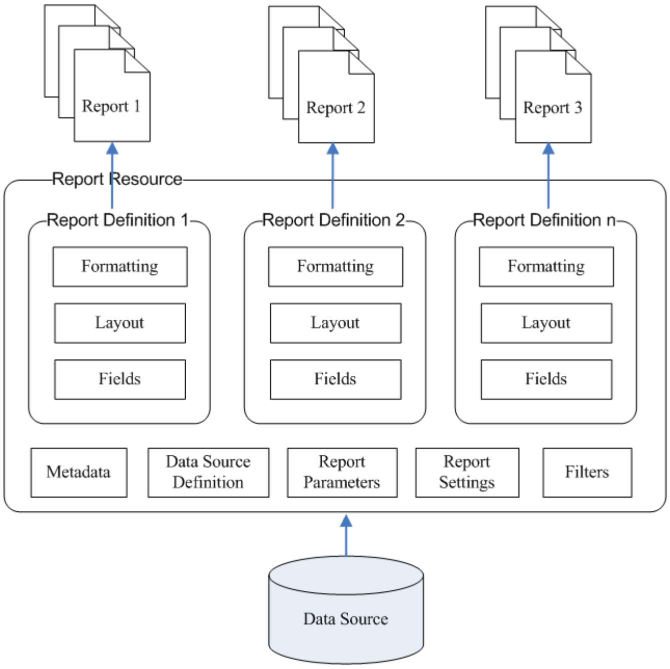
A **report definition** contains all the information that defines the data binding, layout, and customized formatting of a report. It is saved as an XML file that can be edited in the Report Designer, either visually or in the text editor mode.

For a report resource, you can create multiple report definitions that represent different layout and formats to address different requirements. For example, you can design a cash flow statement report definition to meet external financial requirements, and another with a different layout and formatting for internal reporting.

A **report** is a collection of your desired data, as defined in the report resource, organized in your desired format, as defined in the report definition. This is what the QAD Reporting Framework is all about.

To sum up, report resources are primarily concerned with what data to display and report definitions are mainly about how to display them in the generated reports. And since you may want to retrieve different data from the same data source and present it in different formats, you can create multiple report definitions within the same report resource to generate different reports.

Fig. 2.2
Report Resource, Report Definition, and Report



Example You need to generate sales order by customer reports and unconfirmed sales order reports on a weekly basis. You create a report resource that retrieves data from the sales order-related tables and then create two report definitions for the two kinds of reports. You then either run these reports manually every week or schedule a weekly batch run for them to generate the reports you need.

Choosing a Report Data Source

Before you create a report, you need to determine where your report data comes from. The QAD Reporting Framework supports three types of data sources: generic proxy (Progress program), browse, and QAD Financials API. Depending on which type of data source you use, you may need to perform some additional implementation steps.

Generic Proxy (Progress Program) Data Source

Generic proxy is a built-in data source provider that calls a Progress program through the QAD .NET UI to get the required data. The Progress program implements all the logic of data queries, calculations, or even database updates when the report is run. This is the most powerful and flexible type of data source in that you can do anything that the Progress language could do to manipulate data in the report. However, it does require you to have Progress programming experience and knowledge of the underlying database schemas in order to write such Progress programs.

Generic data source implementation entails the following general steps:

1. Develop a generic proxy `.p` program. To develop proxy programs for the QAD Reporting Framework, you are required to be familiar with both Progress programming and QAD ERP database schema.

2. Deploy the generic proxy program into the QAD .NET UI AppServer tier in the following directory under the webapp root location:

   `<Web App Root>/WEB-INF/pro/com/qad/shell/report/reports`

   Note The proxy layer is generic and can call any data source `.p` program you deploy.

3. To improve performance, compile the program into a `.r` file by running `mkdt compile` from the `<Web App Root>/WEB-INF/pro/com/` directory.

For detailed information on implementing generic proxies, see “Implementing a Progress Data Source Program” on page 89.

Browse Data Source

The QAD Reporting Framework supports both classic QAD ERP (MFG/PRO) browses and Financials browses as data sources.

In QAD Enterprise Applications, browses display selected data in the form of a table. Column headings are field labels; rows are field values. The field values in a browse come from any table in the QAD Enterprise Applications (MFG/PRO) schema. A browse includes selected values from one table or several joined tables.

You can use an existing browse as the data source for your report by associating the browse ID with the report resource. Then, you can have all the fields in the browse at your disposal to design your report definition.

Classic QAD ERP (MFG/PRO) browses are maintained using Browse Maintenance (36.20.13) and View Maintenance (36.20.18).

Financials browses are maintained using the Financials CBF tool.

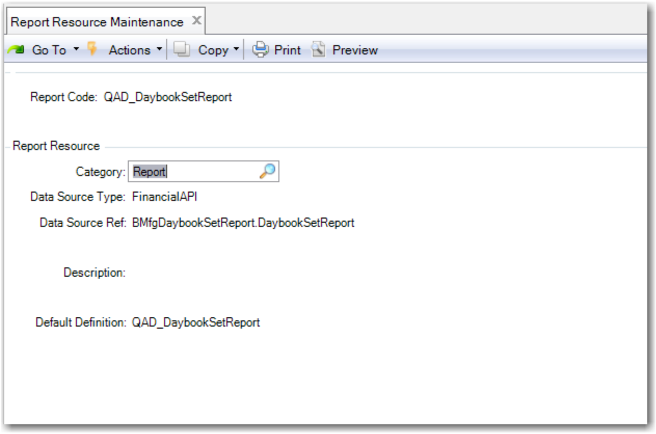
QAD Financials API Data Source

Any query defined in the QAD Financials API can be used as a report data source.

Creating a Report Resource

Use Report Resource Maintenance to create a report resource.

Fig. 2.3
Report Resource Maintenance



Report Code: QAD_DaybookSetReport

Report Resource

Category: Report

Data Source Type: FinancialAPI

Data Source Ref: BMfgDaybookSetReport.DaybookSetReport

Description:

Default Definition: QAD_DaybookSetReport

Enter the following fields. Click Next or press Enter to move to the next frame or field; click Back to return to the previous field.

Report Code. Specify a code that identifies a report resource.

Important QAD-provided built-in reports, report resources and templates all begin with "QAD_". Do not create or modify reports, report resources, or templates with this prefix. Otherwise, your customized changes will get overwritten during system upgrades from QAD.

Category. Select one of the following report resource types for different report providers: Report for QAD .NET-based reporting and Dashboard for Cognos dashboard reports.

Note Cognos dashboard reports are currently not implemented.

Data Source Type. Specify a data source type that indicates how the report retrieves its data:

- Browse: The report uses browses as its data source. Both classic browses (created in Browse Maintenance) and Financials browses (created in the Financials CBF tool) are supported.
- Proxy: The report accesses the database through the generic proxy program.
- Financials API: The report retrieves data through the QAD Financials API.

Data Source Ref. Provide the reference information for retrieving data through the data source provider.

- For classic QAD ERP browses maintained in Browse Maintenance, enter `<BrowseServerType>:<QueryID>`; for example, `QAD.Browse.MfgProBrowseServer:so009`. `<BrowseServerType>` is optional. If it is not specified, the data source reference defaults to the QAD ERP browse server. In this case, `so009` is equivalent to `QAD.Browse.MfgProBrowseServer:so009`.

  For Financials browses created in the Financials CBF tool, enter the following:

  ```
  BaseLibrary.Lookup.BLFBrowseServer:<BusinessComponent>.
  <QueryMethod>
  ```

  Here is an example using this naming syntax:

  ```
  BaseLibrary.Lookup.BLFBrowseServer:BJournalEntry.SelectPosting
  ```

- For the generic proxy data source, specify a data source proxy program file name; for example, `myReport.p`.
- For the Financials API data source, specify a Financials reporting component name followed by a method name; for example, BGLReport.GLList, where BGLReport is a component name and GLList is a method name.

Description. Provide a description of the report resource.

Default Definition. Specify the default report definition for the report resource. When you open a report resource in Report Viewer or Report Designer, this report definition is loaded by default.

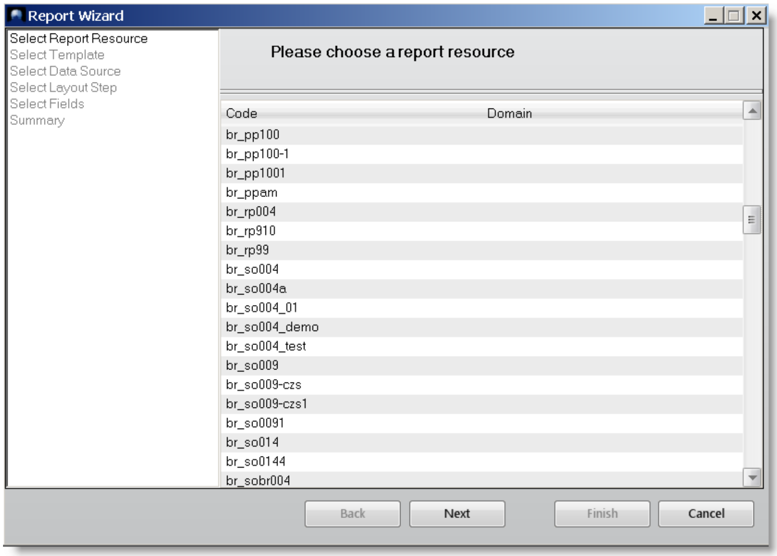
Creating a Report Definition

1 Type Report Resource Designer in the menu search field and press Enter.

2 Click the New icon on the Report Designer Toolbar. The Report Wizard window appears.

3 Select the report resource you previously created and click Next.

Fig. 2.4
Report Wizard: Select a Report Resource



Please choose a report resource

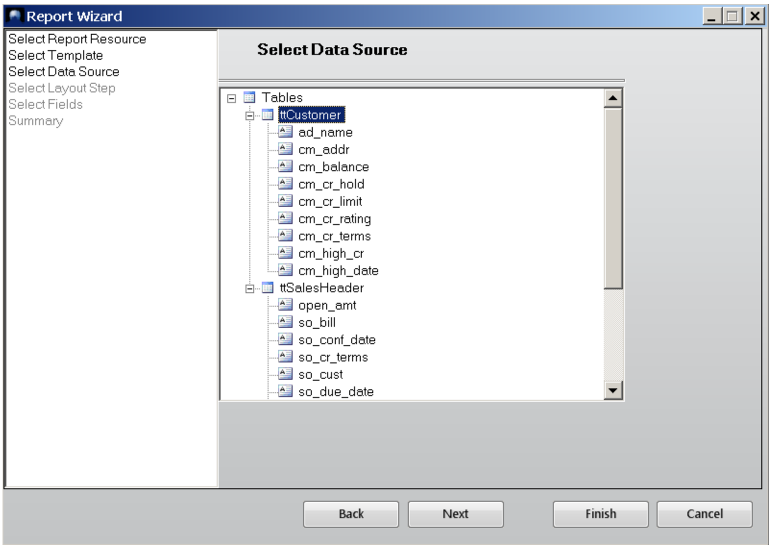
Code	Domain
br_pp100	
br_pp100-1	
br_pp1001	
br_ppam	
br_rp004	
br_rp910	
br_rp99	
br_so004	
br_so004a	
br_so004_01	
br_so004_demo	
br_so004_test	
br_so009	
br_so009-czs	
br_so009-czs1	
br_so0091	
br_so014	
br_so0144	
br_sobr004	

4 Select a report template or select None to use the default built-in report template. Click Next.

5 Select a table as the report data source and click Next.

- All the available tables you can select as data sources are listed in a tree.
- To view all the fields in a table, click the plus sign next to the table to expand the tree.

Fig. 2.5
Report Wizard: Select Data Source

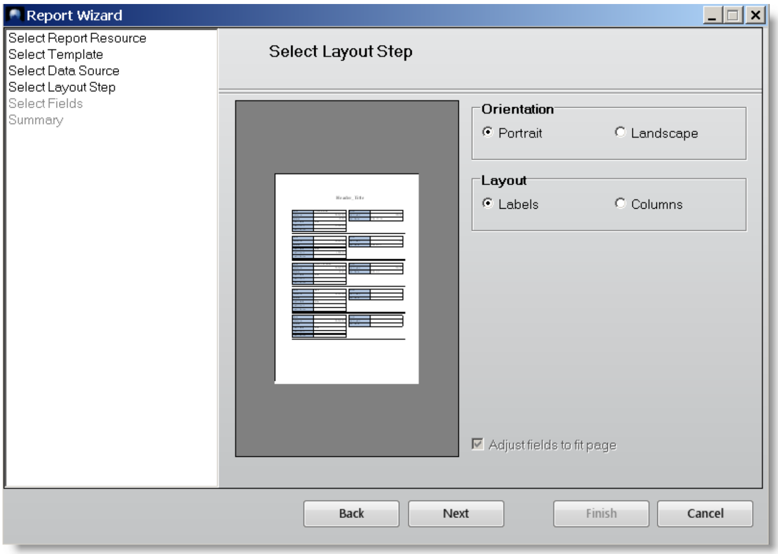


Select Data Source

- Tables
  - ttCustomer
    - ad_name
    - cm_addr
    - cm_balance
    - cm_cr_hold
    - cm_cr_limit
    - cm_cr_rating
    - cm_cr_terms
    - cm_high_cr
    - cm_high_date
  - ttSalesHeader
    - open_amt
    - so_bill
    - so_conf_date
    - so_cr_terms
    - so_cust
    - so_due_date

6 This screen offers you several options to define how the data will be organized on the page. Select the layout that best approximates what you want the final report to look like.

Fig. 2.6
Report Wizard: Select Layout



Select Layout Step

Orientation
● Portrait ○ Landscape

Layout
● Labels ○ Columns

☑ Adjust fields to fit page

Orientation. Choose whether to design and render the report in Portrait or Landscape mode.

Layout. Specify how you want the field names and fields organized in the report.

- Labels: For each field in the report, the corresponding field name is placed to the left as a label.
- Columns: Field data is organized in a column or multiple columns with field names as column headers.

Adjust fields to fit page. Select this option to adjust fields to fit the page width; otherwise, clear this option.

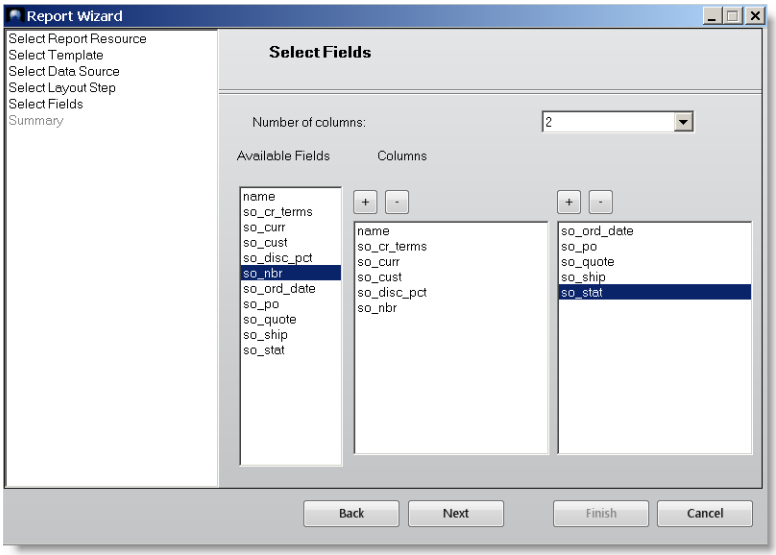
Click Next to continue.

7 Select Fields into the report.

This screen displays differently depending on whether you chose the label-style or column-style layout in the previous step.

- This screen displays when you chose the label-style layout in the previous step.

Fig. 2.7
Report Wizard: Select Fields



Number of columns: 2

Available Fields: name, so_cr_terms, so_curr, so_cust, so_disc_pct, so_nbr, so_ord_date, so_po, so_quote, so_ship, so_stat

Columns:

name, so_cr_terms, so_curr, so_cust, so_disc_pct, so_nbr

so_ord_date, so_po, so_quote, so_ship, so_stat

In this screen, choose the number of columns you want to place data in and specify the fields to display in each column.

To specify the number of columns in the report

Select a column number from the list. A report can have up to four columns. When you select a column number, the number of the Details boxes changes accordingly.

If you have selected fields into the Selected Fields boxes, they will be cleared when you select a new column number.

To select a field into a column

You can either drag the field into the Selected Fields box or select the field in the Available Fields box and then click the plus icon (+) above the Selected Fields box.

To select multiple fields all at once into a column

Hold down Shift and click to select a number of fields in a row or hold down Ctrl and click several discontinued fields and then perform the drag-and-drop action or use the plus icon.

To remove a field from a column

Select the field in the Selected Fields box and click the minus icon (–) above the Selected Fields box.

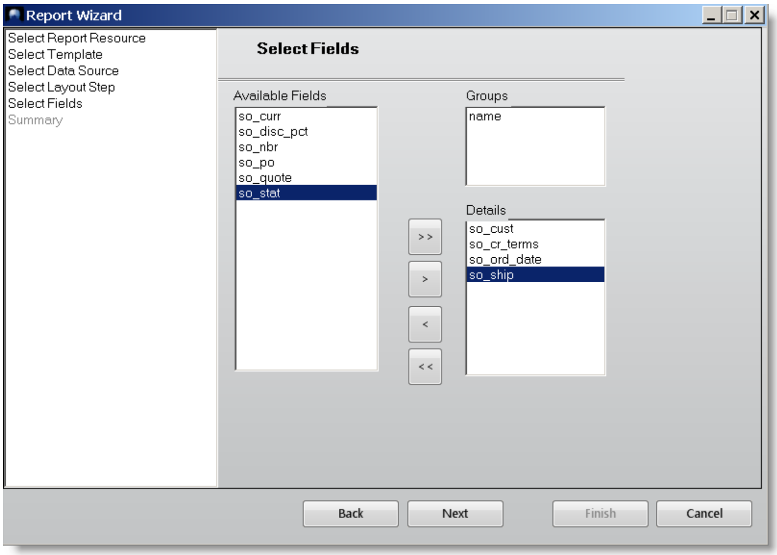
To remove multiple fields all at once from a column

Hold down Shift and click to select a number of fields in a row or hold down Ctrl and click several non-contiguous fields and then click the minus icon above the Selected Fields box.

Click Next to continue.

- This screen displays when you chose the column-style layout in the previous step.

Fig. 2.8
Report Wizard: Select Fields



In this screen, select the fields you want to display in the report and optionally specify fields to group data by in the report.

To select fields from the source table into the report

Select them in the Available Fields box on the left and drag them into the Details box.

To select fields by which to group data in the report

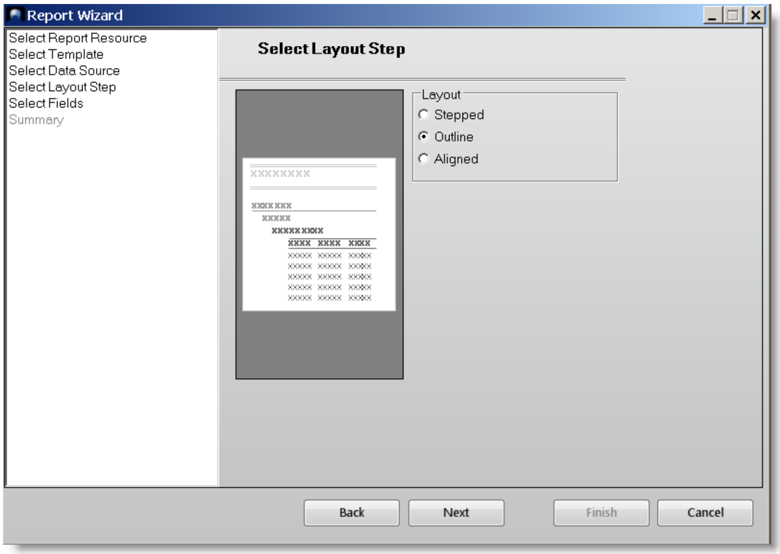
Select them in the Available Fields list box on the left and drag them into the Groups box on the right.

You can remove fields from the Groups and Details boxes through drag-and-drop in the opposite direction.

To move all the fields from the Available Fields box to the Details box and vice versa, use the >> and << buttons.

Click Next. Another layout option screen displays. Select a layout for the report and click Next.

Fig. 2.9
Report Wizard: Select Column-Style Layout



Layout
○ Stepped
● Outline
○ Aligned

8 The Summary screen recaps the information you have specified for the report definition. If you want to modify the settings, click Back to return to previous steps to edit them; otherwise, click Finish to complete the basic report setup and exit Report Wizard.

9 When you return to the Report Designer main screen, the report displays in the visual design mode in the Design pane based on the newly created report definition. Save the report as a new report definition. You can further customize it in Report Designer.

Chapter 3

Exploring the Report Designer Workspace

About Report Designer

Report Designer is a powerful, visual report design tool that lets you manage report definition files, define report layout, bind data to report fields, format reports, and enhance your reports using advanced components such as labels, charts, pictures, and drawings.

Report Designer features an intuitive, Microsoft Access-style user interface with a rich set of form components and formatting tools at your disposal that give you total control over the content, layout, and format of your reports. If you are familiar with the Microsoft Access report design view, you will find Report Designer very easy to use.

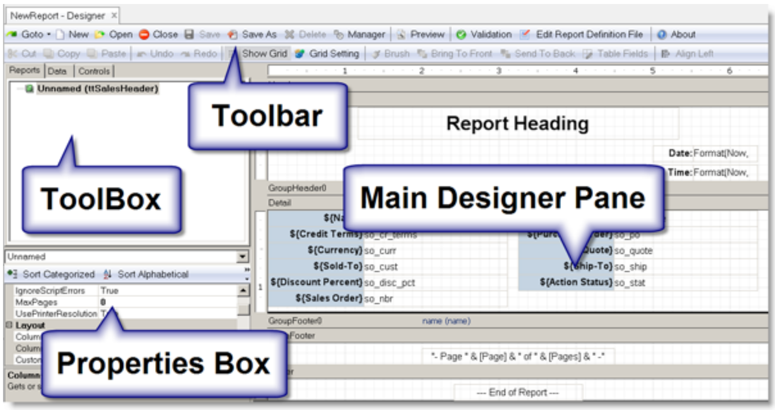
Report Designer provides a Report Wizard that guides you through the process of creating a basic report definition from scratch. You can then proceed from this starting point to further design specialized or more complex reports using Report Designer's comprehensive data-binding, layout design, and formatting tools.

Report Designer Work Areas

The main Report Designer window has the following components:

- Toolbar: Provides shortcuts to the most common design functions.
- Toolbox: Provides tools for creating report fields.
- Main Designer Pane: This is the main working area of the Designer. It shows the report's sections and fields and allows you to change the report definition.
- Properties Window: Allows you to edit properties for the objects that are selected in the Designer.

Fig. 3.1
Report Designer Workspace



Toolbar

The toolbar provides access to the following groups of functions:

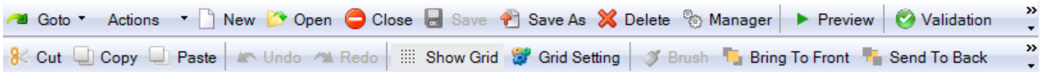


Table 3.1
Main Functions

Button	Name	Description
↷	Goto	Go to Report Resource Maintenance.
	Actions	Menu of options for exporting and importing metadata and data for testing and debugging purposes.
🗋	New	Launch Report Wizard to create a new report definition.
📂	Open	Launch Report Definition Manager to open an existing report definition.
⛔	Close	Close the current report definition.
💾	Save	Save the current report definition.
⇧	Save As	Save the current report definition as another one.
✕	Delete	Delete the current report definition.
⚙	Manager	Launch Report Definition Manager to delete existing report definitions, set the default report definition, and modify some of their attributes.
▶	Preview	Display the current report definition in preview mode. In the preview mode, you can only navigate through the generated report.
✓	Validate	Check the validity of the current report definition file. If errors are found, error messages will be displayed.
✎	Edit Report Definition File	Open the current report definition file in code mode for editing.
?	About	Display Report Designer version information.

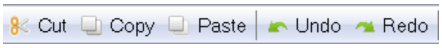


Table 3.2
Edit Functions

Button	Name	Description
✂	Cut	Cut the selected objects on the report.
❐	Copy	Copy the selected objects on the report to the clipboard.
❐	Paste	Paste cut or copied objects from the clipboard to the currently selected area on the report.
↶	Undo	Undo any actions you have performed on the report.
↷	Redo	Redo the actions you have undone.

Table 3.3
Format Functions

Button	Name	Description
	Show Grid	Toggle background grid on and off.
	Grid Settings	Configure grid settings such as grid units and grid spacing.
	Brush	When multiple objects are selected, apply the format of the last selected object to all other selected objects.
	Bring to Front	Bring the selected object to the foreground.
	Send to Back	Send the selected object to the background.
The following functions only apply to multiple objects on the report and will be grayed if only one object is selected.		
	Table Fields	Merge selected objects into table fields.
	Align Left	Align multiple selected objects to the left boundary of the last selected object.
	Align Center	Horizontally align multiple selected objects to the center of the last selected object.
	Align Right	Align multiple selected objects to the right boundary of the last selected object.
	Center Horizontally on Section	Horizontally position selected objects to the center of the section.
	Align Top	Align multiple selected objects to the top boundary of the last selected object.
	Align Middle	Vertically align multiple selected objects to the middle of the last selected object.
	Center Vertically on Section	Vertically position selected objects to the center of the section.
	Align Bottom	Align multiple selected objects to the bottom boundary of the last selected object.
	Equal Height	Resize multiple selected objects to the same height.
	Equal Width	Resize multiple selected objects to the same width.
	Equal Size	Resize multiple selected objects to the same height and width.
	Equal Horizontal Spacing	Reposition multiple selected objects so that they are equally spaced out horizontally.
	Decrease Horizontal Spacing	Horizontally reduce spacing between multiple selected objects.
	Increase Horizontal Spacing	Horizontally increment spacing between multiple selected objects.
	Equal Vertical Spacing	Reposition multiple selected objects so that they are equally spaced out vertically.

Button	Name	Description
	Decrease Vertical Spacing	Vertically reduce spacing between multiple selected objects.
	Increase Vertical Spacing	Vertically increment spacing between multiple selected objects.

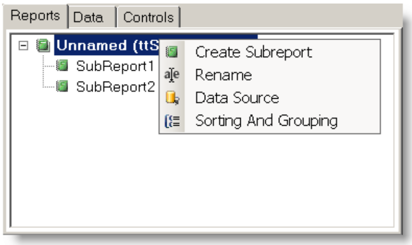
Toolbox

The Report Designer toolbox has three tabs: Reports, Data, and Controls.

Reports Tab

The Reports tab displays the report definition you are designing as well as all subordinate subreports embedded in the current report.

Fig. 3.2
Reports Tab

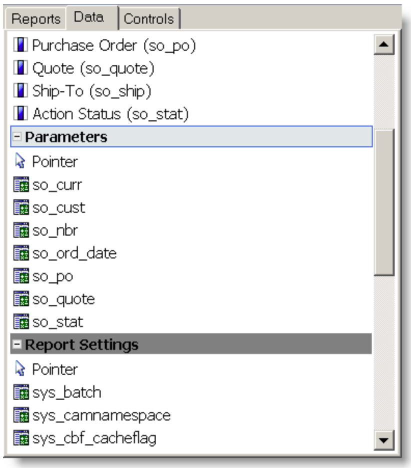


Right-clicking on the current report definition in this window brings up a shortcut menu that gives you access to a number of report design functions.

Data Tab

The Data tab contains three groups of data: fields, parameters, and report settings. Except for the Pointer button, which is used to deselect any currently selected object, each button under these groups creates a field on the report and initializes its properties.

Fig. 3.3
Data Tab



- Purchase Order (so_po)
- Quote (so_quote)
- Ship-To (so_ship)
- Action Status (so_stat)

- Parameters
  - Pointer
  - so_curr
  - so_cust
  - so_nbr
  - so_ord_date
  - so_po
  - so_quote
  - so_stat

- Report Settings
  - Pointer
  - sys_batch
  - sys_camnamespace
  - sys_cbf_cacheflag

The Fields group contains all the available fields that are bound to the source record set.

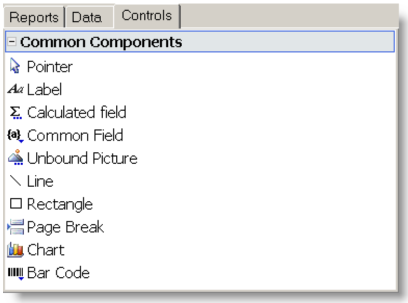
The Parameters group contains all the available search condition parameters under the Filter tab in Report Viewer.

The Report Settings group contains all the available setting variables under the Settings tab in Report Viewer.

Controls Tab

The Controls tab displays all the common controls and components that you can add to your reports.

Fig. 3.4
Controls Tab



- Common Components
  - Pointer
  - Label
  - Calculated field
  - Common Field
  - Unbound Picture
  - Line
  - Rectangle
  - Page Break
  - Chart
  - Bar Code

Table 3.4
Controls

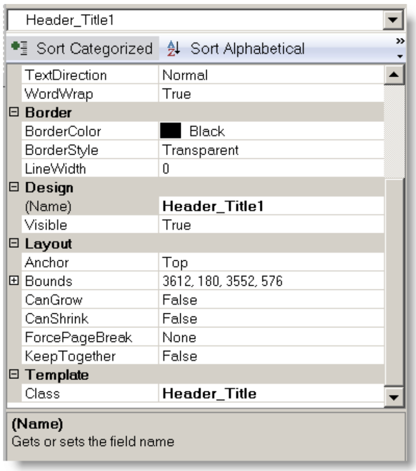
Control	Name	Description
↖	Pointer	Deselects any field on the report and releases the mouse focus.
Aa	Label	Creates a field for a label. Use the Properties window to specify the appearance, border, design, and layout of the label.

Control	Name	Description
Σ	Calculated Field	Creates a calculated field. When you click this button, the code editor dialog box appears so you can enter the VBScript expression whose value you want to display.
{a}	Common Calculated Field	Creates a field with a commonly used expression. When you click this button, a menu appears and you can select expressions that render the date or time when the report was created or printed, the page number, page count, “page n of m,” or the report name.
🖼	Unbound Picture	Creates a field that displays a static picture, such as a logo. When you click this button, a dialog box appears to prompt you for a picture file to insert in the report. A copy is made of the picture you select and placed in the same directory as the report file. You must distribute this file with the application unless you embed the report file in the application. When you embed a report file in your application, any unbound picture files are embedded, too.
╲	Line	Creates a line. Lines are often used as separators.
□	Rectangle	Creates a rectangle. Rectangles are often used to highlight groups of fields or to create tables and grids.
▤	Page Break	Creates a field that inserts a page break.
📊	Chart	Creates a field that displays a chart. Unlike most bound fields, Chart fields display multiple values. To select the data you want to display, set the Chart field’s DataX and DataY properties. To format the values along the X and Y axis, set the FormatX and FormatY properties. You can use the ChartType property to specify the type of the chart: bar, column, pie, scatter, line, or area.
‖‖‖	Barcode	Creates a field that displays a barcode that is rendered from your designated data. To specify a barcode standard, select the standard you want to use from the Barcode property list in the Properties box. You can choose from the following barcode standards: Code39, Code 93, Code128, Code2of5, Codeabar, PostNet, Ean13, Ean8, UpcA.

Properties Window

The Properties window allows you to edit properties for the objects that are selected in the Designer.

Fig. 3.5
Properties Window



Header_Title1

TextDirection	Normal
WordWrap	True
⊟ **Border**	
BorderColor	■ Black
BorderStyle	Transparent
LineWidth	0
⊟ **Design**	
(Name)	**Header_Title1**
Visible	True
⊟ **Layout**	
Anchor	Top
⊞ Bounds	3612, 180, 3552, 576
CanGrow	False
CanShrink	False
ForcePageBreak	None
KeepTogether	False
⊟ **Template**	
Class	**Header_Title**

(Name)
Gets or sets the field name

The object drop-down list always displays the currently selected object on the report. If no report object is selected, the box displays the current report definition type.

To select an object, either click on it on the report in the design pane, or select the name of the object from the object drop-down list.

The Properties window's toolbar lets you change the way properties are displayed and revert a property's value to the default.

Table 3.5
Properties Window Toolbar

Button	Name	Description
	Sort by Category	Groups properties under categories, which vary according to the type of the selected object. You can expand or collapse a category to show or hide the properties under it. This is the default display mode.
A Z↓	Sort Alphabetically	Sorts properties in alphabetical order.
	Use Default Value	Reverts a selected property value to the default. If a template is specified for the report, this button will be greyed if the property value has not been changed from the value inherited from the template. Otherwise, the button will be enabled and if clicked will revert the property value to the value from the template, and re-couple the property to the template such that any future changes to the property in the template will be inherited by this selected object.

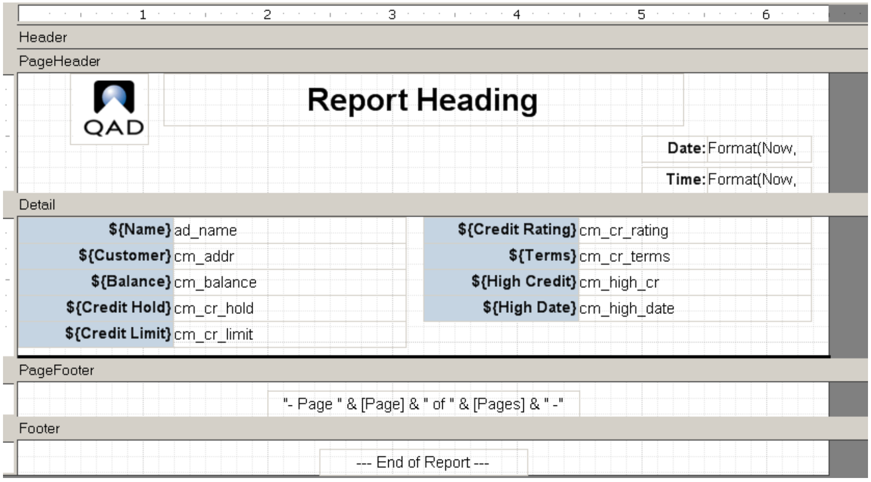
The main area of the Properties window displays all the properties of the selected object. They change as the selected object changes.

The property value displays to the right of the property name. To edit a property value, click on the value field and it turns into a text field, drop-down box, or combo box, depending on the type of the value; specify the value you want.

Design Pane

The Design pane is the main working area of the Designer. It shows the report's sections and fields and allows you to change the report definition visually using point-and-click selections. For details on how to design a report, see Chapter 4, "Designing Reports in Report Designer," on page 29.

Fig. 3.6
Design Pane



Header

PageHeader

Report Heading

Date: Format(Now,

Time: Format(Now,

Detail

${Name}	ad_name	${Credit Rating}	cm_cr_rating
${Customer}	cm_addr	${Terms}	cm_cr_terms
${Balance}	cm_balance	${High Credit}	cm_high_cr
${Credit Hold}	cm_cr_hold	${High Date}	cm_high_date
${Credit Limit}	cm_cr_limit		

PageFooter

"- Page " & [Page] & " of " & [Pages] & " -"

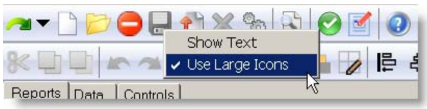
Footer

--- End of Report ---

Customizing the Toolbar

To change the way a toolbar looks, right-click on the toolbar and choose the Show Text and Use Large Icon options.

Fig. 3.7
Customize the Toolbar



Show Text: Select this option to display tooltips for the toolbar menus or clear this option to hide tooltips.

Use Large Icon: Select this option to display large menu icons or clear this option to show small icons.

Shortcut Menus

Report Designer provides context-sensitive shortcut menus that give you access to most commonly used functions.

To bring up shortcut menus:

- Right-click a report definition under the Reports tab in the toolbox.
- Right-click on a blank space in the Design pane.
- Right-click an object in the Design pane.
- Right-click a subreport in the Design pane.

Fig. 3.8
Subreport Shortcut Menu

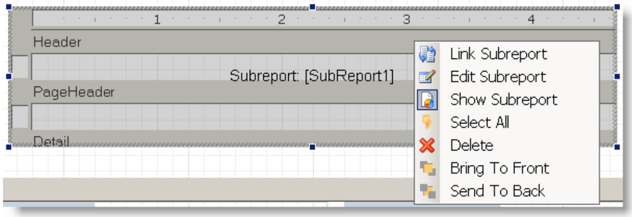


Table 3.6
Subreport Shortcut Commands

Button	Name	Description
🔗	Link Subreport	Opens a dialog box that lets you define which records will be included in the subreport by specifying a master field in the main report and a child field in the subreport. This will set the Text property on the subreport field to an expression that will be used as a filter on the subreport data source.
✎	Edit Subreport	Open the subreport fully in the Design pane for editing.

Chapter 4

Designing Reports in Report Designer

Importing and Exporting Report Resources 67

Describes how to import report resources using Report Resource Import or export them using Report Resource Export.

The QAD Reporting Framework gives you not only the simplicity to build basic reports from scratch, but also the flexibility to tailor them to meet your specific requirements.

The basic report generated for you by the Report Wizard is a good starting point, but you will usually need to adjust and enhance it to get exactly what you want. You can do this with Report Designer.

Launching Report Designer

Use one of the following ways to access Report Designer:

- Type Report Resource Designer in the menu search field and press Enter.
- If you have created a menu item for your report, locate it in the Applications Pane and right-click on it; then choose Design from the shortcut menu. The Report Designer window appears.

Note Before you can access Report Designer to create a report definition, you must first create a valid report resource.

Managing Report Definition Files

In the Report Designer, you can create, load, and delete report definition files, as well as edit them either in the WYSIWYG (What You See Is What You Get) or code edit mode.

Creating a New Report Definition

Click the New button on the Report Definition toolbar. Report Wizard takes you through the process of creating a basic report definition. Click the Save icon on the toolbar to save the definition as an XML file. For details on creating a basic report definition using Report Wizard, see “Creating a Report Definition” on page 14.

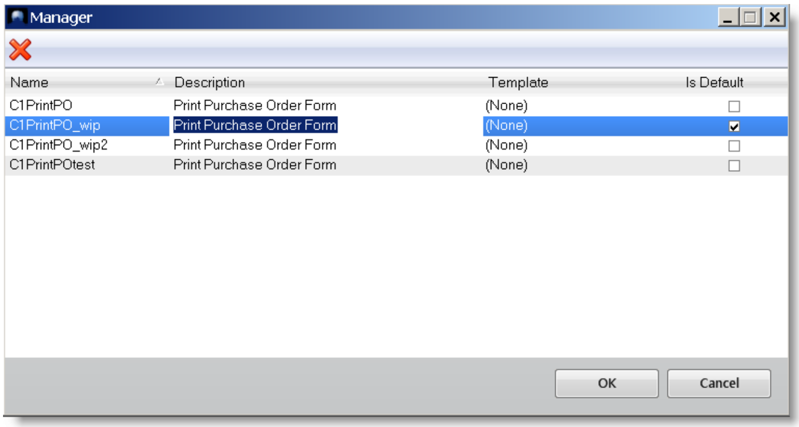
Loading an Existing Report Definition File

Click the Open button on the toolbar; then in the Select Report Definition window, double-click the report definition you want to load. The Select Report Definition window also let you enter search conditions to search for the report definition you want to load.

Using Report Definition Manager

You can use Report Definition Manager to delete existing report definitions as well as modify some of their attributes.

Fig. 4.1
Report Sections Example



Name	Description	Template	Is Default
C1PrintPO	Print Purchase Order Form	(None)	☐
C1PrintPO_wip	Print Purchase Order Form	(None)	☑
C1PrintPO_wip2	Print Purchase Order Form	(None)	☐
C1PrintPOtest	Print Purchase Order Form	(None)	☐

To launch Report Definition Manager

Click the Manager button on the toolbar.

To delete an existing report definition

In the Manager window, click the report definition and then click the Delete button at the top. Confirm the deletion when prompted.

To select a different template for a report definition

Click the current template next to the report definition and select a different template from the list. The layout and formatting of the report definition will be changed after you assign a different template to it.

To set the default report definition for the report resource

Select the Is Default check box for the report definition. You must set one and only one default report definition.

When you open the report resource from the Applications menu tree, the default report definition is loaded.

Working with Report Sections

About Report Sections

A basic report is divided into five sections: header, page header, details, page footer, and footer. The sections contain fields that hold the labels, variables, and expressions that you want in the printed report. If you add groups to a report, the report will also contain a group header and a group footer section. For example, a report with 3 grouping levels will have 11 sections.

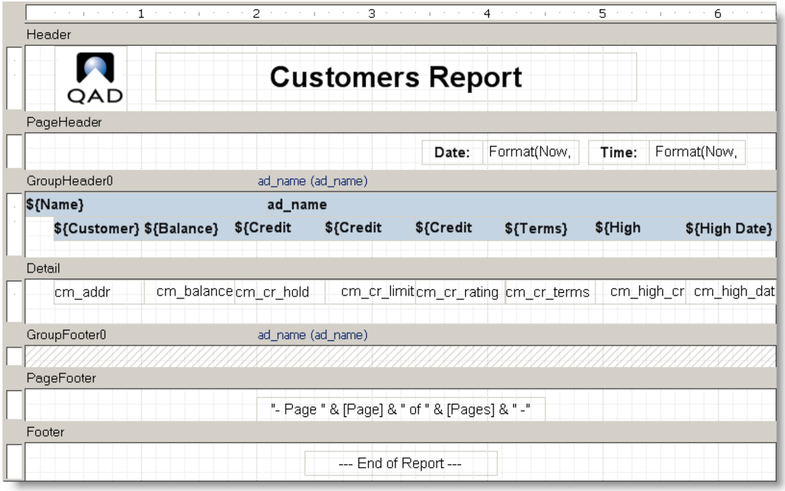
The sections of the report determine what each page, group, and the beginning and end of the report look like. The following table describes where each section appears in the report and what it is typically used for.

Table 4.1
Report Sections

Section	Appears	Typically Contains
Report Header	Once per report	The report title and summary information for the whole report.
Page Header	Once per page	Labels that describe detail fields, and/or page numbers.
Group Header	Once per group	Fields that identify the current group, and possibly aggregate values for the group; for example, total, percentage of the grand total.
Detail	Once per record	Fields containing data from the source record set.
Group Footer	Once per group	Aggregates values for the group.
Page Footer	Once per page	Page number, page count, date printed, report name.
Report Footer	Once per report	Summary information for the whole report.

In this example, the Header section contains a label with the report title. The Page Header section contains labels that display the current date and time. The Group Header section contains labels that identify the fields in the Detail section, and the Page Footer section contains fields that show the page number and the total page count for the report. Data is grouped inside a group section marked by a group header and a group footer.

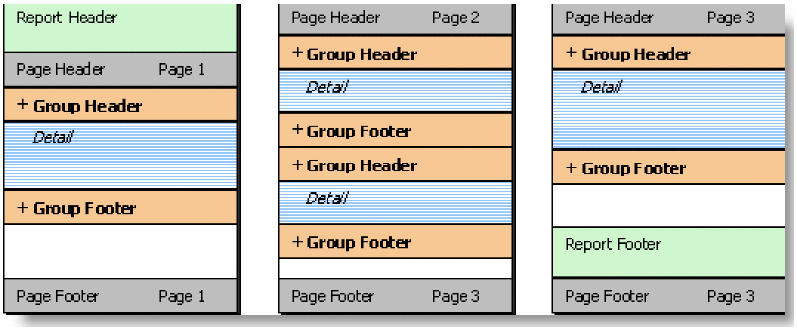
Fig. 4.2
Report Sections Example



Note that sections can be made invisible, but they cannot be added or removed, except by adding or removing groups.

The following diagram shows how each section is rendered on a typical report.

Fig. 4.3
Rendering a Report



- Report Header

  The first section rendered is the Report Header. This section usually contains information that identifies the report.

- Page Header

  After the Report Header comes the Page Header. If the report has no groups, this section usually contains labels that describe the fields in the Detail Section.

- Group Headers and Group Footers

  The next sections are the Group Headers, Detail, and Group Footers. These are the sections that contain the actual report data. Group Headers and Footers often contain aggregate functions such as group totals, percentages, maximum and minimum values, and so on. Group Headers and Footers are inserted whenever the value of the expression specified by the GroupBy property changes from one record to the next.

- Detail

  The Detail section contains data for each record. It is possible to hide this section by setting its Visible property to False, and display only Group Headers and Footers. This is a good way to create summary reports.

- Page Footer

  At the bottom of each page is the Page Footer Section. This section usually contains information such as the page number, total number of pages in the report, and/or the date the report was printed.

- Report Footer

  Finally, the Report Footer section is printed before the last page footer. This section is often used to display summary information about the entire report.

- Customized sections

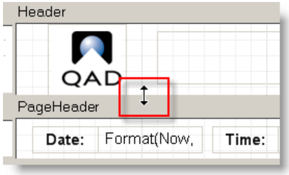
  You can determine whether a section is visible by setting its Visible property to True or False. Group Headers can be repeated at the top of every page (whether or not it is the beginning of a group) by setting their Repeat property to True. Page Headers and Footers can be removed from pages that contain the Report Header and Footer sections by setting the PageHeader and PageFooter properties on the Layout object.

Resizing a Report Section

To resize a section, select its border and with your mouse pointer drag to the position where you want it. The rulers on the left and on top of the design window show the size of each section (excluding the page margins). Note that you cannot make the section smaller than the height and width required to contain the fields in it. To reduce the size of a section beyond that, move or resize the fields in it first, then resize the Section.

To see how this works, move the mouse to the area between the bottom of the Page Header section and the gray bar on top of the Detail Section. The mouse cursor changes to show that you are over the resizing area. Click the mouse and drag the line down until the section is about twice its original height.

Fig. 4.4
Resizing a Report Section



Release the mouse button and the section is resized.

Hiding a Report Section

You can hide a report section so that it will not appear in the printed report. However, a hidden section is still visible in the design view.

To hide a report section, click a section to select it; then set its Visible property to False in the Properties box.

Grouping and Sorting

You can organize the data in your report by grouping and sorting data, using running sums, and creating aggregate expressions.

Grouping and Sorting Data

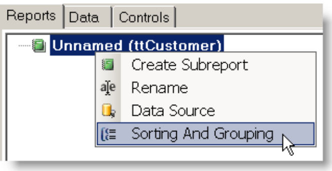
After designing the basic report layout, you may decide that grouping the records by certain fields or other criteria would make the report easier to read. Grouping allows you to separate groups of records visually and display introductory and summary data for each group. Groups are also used for sorting the data, even if you do not plan to show the Group Header and Footer sections.

You can also specify how each group should be sorted using the group’s GroupBy and Sort properties.

To add or edit the groups and specify the sorting rule in the report

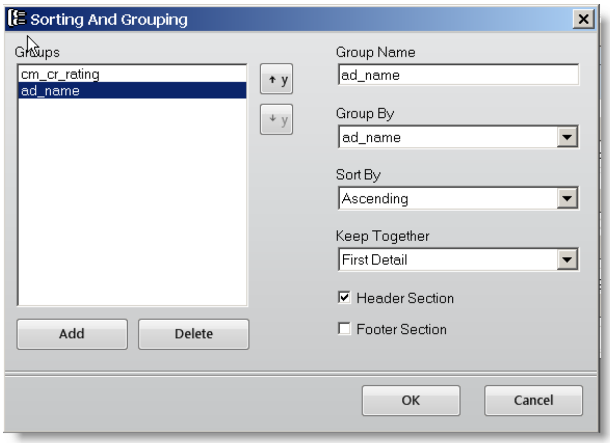
1 In Report Designer, right-click a report under the Reports tab in the toolbox; then select Sorting and Grouping from the shortcut menu.

Fig. 4.5
Select Sorting and Grouping



2 The Sorting and Grouping dialog box appears. Use this dialog box to create, edit, reorder, and delete groups.

Fig. 4.6
Sorting and Grouping Dialog Box



Group Name: ad_name

Group By: ad_name

Sort By: Ascending

Keep Together: First Detail

☑ Header Section

☐ Footer Section

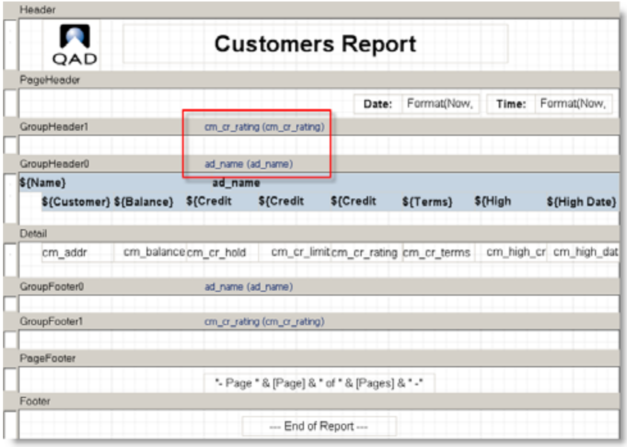
3 To create a group, click the Add button and set the properties for the new group. The Group By field defines how the records will be grouped in the report. For simple grouping, you can select fields directly from the drop-down list.

4 Next, select the type of sorting you want (Ascending in this example). You can also specify whether the new group will have visible Header and Footer sections, and whether the group should be rendered together on a page.

5 If you add more fields, you can change their order using the arrow buttons on the right of the Groups list. This automatically adjusts the position of the Group Header and Footer sections in the report. To delete a field, use the Delete button.

6 Once you are done arranging the fields, click OK to dismiss the dialog box and see the changes in the Designer. On the top of the new sections there are labels that contain the section name and the value of the group’s GroupBy property.

Fig. 4.7
New Group Section


Header
QAD
Customers Report
PageHeader
Date: Format(Now,
Time: Format(Now,
GroupHeader1
cm_cr_rating (cm_cr_rating)
GroupHeader0
ad_name (ad_name)
${Name}
ad_name
${Customer} ${Balance} ${Credit ${Credit ${Credit ${Terms} ${High ${High Date}
Detail
cm_addr cm_balance cm_cr_hold cm_cr_limit cm_cr_rating cm_cr_terms cm_high_cr cm_high_dat
GroupFooter0
ad_name (ad_name)
GroupFooter1
cm_cr_rating (cm_cr_rating)
PageFooter
"- Page " & [Page] & " of " & [Pages] & " -"
Footer
--- End of Report ---

Adding Running Sums

You can easily maintain running sums over groups or over the entire report.

To keep running sums over groups:

1 In Report Designer, open the report definition you want to design.

2 Switch to the Item tab in the toolbox and click Calculated Field under the Common Components group.

3 The VBScript Editor displays. Enter the following script:

```
Sum(FieldToSumUp)
```

Where *FieldToSumUp* is the name of the field you want to sum up.

4 Move the mouse pointer over to the GroupHeader section of the report and the pointer changes into a cross-hair. Click and drag to define the rectangle that the new field will occupy, and then release the button to create the new field.

Enhancing the Report with Fields

To enhance your report, you can add fields (for example, lines, rectangles, labels, pictures, charts, and so on) to any section. You can also modify the existing fields by changing their properties with the Properties window, or move and resize the fields with the mouse.

Creating Reporting Fields

With the report definition loaded into Report Designer and a data source defined, you can add and edit report fields.

You can bind three different types of data to the fields you add to a report:

- Fields: Values of fields that are bound to the source record set.
- Parameters: Values of search condition parameters under the Filter tab in Report Viewer.
- Report settings: Values of settings variables under the Settings tab in Report Viewer.

The Report Designer toolbox allows you to easily add fields to your report.

To add a field to your report

1 Click the Data tab in the Report Designer toolbox.

2 Click and expand a group to display all available data buttons. In the Data tab, the Fields and Parameters groups include fields for the report data source. The Report Settings group includes the following system settings:

sys_base_currency	The base currency (base_curr) from the underlying QAD Applications system.
sys_batch	Reserved for future use.
sys_camnamespace	Reserved for future use.
sys_cbf_cacheflag	Reserved for future use.
sys_cbf_entity	Reserved for future use.
sys_cbf_instance_id	Reserved for future use.
sys_cbf_language	Reserved for future use.
sys_cbf_sessionid	Reserved for future use.
sys_ci_date_separator	The character that separates the components of a date value. (For example, the backslash character: `/`). This is determined by the culture of the user ID running the report.
sys_ci_decimal_digits	The number of decimals places used for numeric values. This is determined by the culture of the user ID running the report.
sys_ci_decimal_separator	The character that represents the decimal point in a numeric value. This is determined by the culture of the user ID running the report.
sys_ci_group_separator	The character that separates groups of digits to the left of the decimal in numeric values. (For example, the comma in `1,000`). This is determined by the culture of the user ID running the report.
sys_ci_group_sizes	The number of digits in each group of digits to the left of the decimal. This is determined by the culture of the user ID running the report.
sys_ci_short_date_pattern	The format of a short date value. This is determined by the culture of the user ID running the report.
sys_client_email	Reserved for future use.
sys_default_report_definition	The name of the report definition that is the default for the report resource.
sys_domain	The domain that the user is in while running the report.
sys_email	The e-mail address specified when the report is scheduled.
sys_file_path	Reserved for future use.
sys_ips	IP address of the client machine that is running the report.
sys_language	The ISO culture name in the format <*languagecode*>-<*country/regioncode*> (For example: `en-US`).
sys_output_type	Reserved for future use.
sys_printer	The printer that the report was sent to (only applies to reports scheduled to a printer).

sys_render_as	A numeric code that represents the output type: 1=PDF 2=Excel 3=Document 4=Text 5=PDFProtected 6=RTF 7=TIFF
sys_reportformat	Reserved for future use.
sys_report_template	Reserved for future use.
sys_run_at_server	Reserved for future use.
sys_schedule_type	Reserved for future use.
sys_search_criteria_display	The position on the report at which the search criteria will be displayed. Values are `Header`, `Footer`, or `None`. These three values will be in English regardless of the language of the report. By default this parameter is set to `Footer`. You can force a specific search criteria location (see "Adding Search Criteria Report Parameter" on page 59).
sys_trusted_signon_session	The ID of the user that is running the report.

3 Click the field you want to add to the report; then move the mouse pointer over the report and the pointer changes into a cross-hair.

4 Click and drag to define the rectangle that the new field will occupy, and then release the button to create the new field.

If you change your mind, press Ctrl+Z or click the Undo button to cancel the operation.

You can also add fields by copying and pasting existing fields, or by holding down the CTRL key and dragging a field or group of fields to a new position to create a copy.

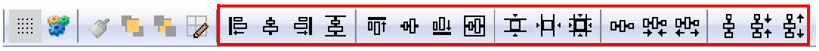
Manipulating and Formatting Fields

You can use the mouse to select fields in Report Designer as usual:

- Click a field to select it.
- Shift-click a field to toggle its selected state.
- Ctrl-drag creates a copy of the selected fields.
- Click the empty area and drag your mouse pointer to select multiple fields.
- With your mouse pointer, drag field corners to resize fields.
- Double-click right or bottom field corners to auto size the field.

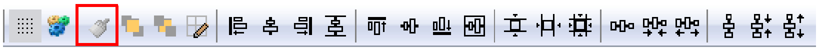
To select fields that intersect vertical or horizontal regions of the report, click and drag the mouse on the rulers along the edges of the Designer. If fields are small or close together, it may be easier to select them by name. You can select fields and sections by picking them from the drop-down list above the Properties window.

Format Fields



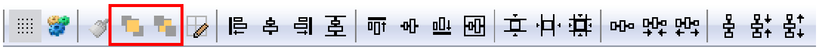
When multiple fields are selected, you can use the buttons on the Format toolbar to align, resize, and space them. When you click any of these buttons, the last field in the selection is used as a reference and the settings are applied to the remaining fields in the selection.

Apply Styles



The Brush button applies the style of the reference field to the entire selection. The style of a field includes all font, color, line, alignment, and margin properties. You can use the Properties window to set the value of individual properties to the entire selection.

Determine Order For Overlapping Fields



If some fields overlap, you can control their z-order using the Bring to Front/Send to Back buttons. This determines which fields are rendered before (behind) the others.

Move Fields Using the Keyboard

You can also select and move fields using the keyboard:

- Use the TAB key to select the next field.
- Use SHIFT-TAB to select the previous field.
- Use the arrow keys to move the selection one pixel at a time (or shift arrow to move by five pixels).
- Use the DELETE key to delete the selected fields.
- When a single field is selected, you can type into it to set the Text property.

Changing Field, Section, and Report Properties

Once an object is selected, you can use the Properties window to edit its properties.

When one or more fields are selected, the Properties window shows property values that all fields have in common, and leaves the other properties blank. If no fields are selected and you click on a section (or on the bar above a section), the Section properties are displayed. If you click the gray area in the background, the Report properties are displayed.

To see how this works, click the label in the Header section and change its Font and ForeColor properties. You can also change a field’s position and dimensions by typing new values for the Left, Top, Width, and Height properties.

The Properties window expresses all measurements in twips (the native unit used by Report Designer), but you can type in values in other units (in, cm, mm, pix, pt) and they will be automatically converted into twips. For example, if you set the field’s Height property to 0.5 inches, the Properties window will convert it into 720 twips.

A twip (derived from TWentieth of an Imperial Point) is a typographical measurement, defined as 1/20 of a typographical point. One twip is 1/1440 inch or 17.639 µm when derived from the PostScript point at 72 to the inch, and 1/1445.4 inch or 17.573 µm based on the printer's point at 72.27 to the inch.

Field Object Settings

Table 4.2 Section Object Settings

Category	Setting	Description
Appearance	Align	Specifies how text is aligned within the field. Click on the field and select icons for Left Top, Center Top, Right Top, Justify Top, Left Middle, Center Middle, Right Middle, Justify Middle, Left Bottom, Center Bottom, Right Bottom, and Justify Bottom.
	BackColor	Select the background color from the Custom, Web, or System tabs.
	Font	Specify the font, font style, size, script, and effects such as strikeout and underline.
	Font Color	Select the foreground color from the Custom, Web, or System tabs.
	Format	Specify the format of the field value, including number, date, and time formats.
	Margins	Specify the Bottom, Left, Right, and Top margin settings.
	Text	Specify the text string of the field. For further information on formatting and translated labels, see "Using Translated Labels" on page 45.
	TextDirection	Specify the direction of the text within the field.
	WordWarp	Specifies whether the text will wrap within the field area.
Border	BorderColor	Specifies the border color of the field.
	BorderStyle	Specifies the field's border style as Transparent, Solid, Dash, Dot, DashDot, or DashDotDot.
	LineWidth	Specifies the width of a field's border (or line).
Data	Text	Specifies the field's text or corresponding database identifier for the field data.
Design	(Name)	Specifies the system field name.
	Visible	Specifies whether the field is visible on the report (True or False).
Layout	Anchor	Specifies the vertical position of the field area relative to its containing section as Top, Bottom, or Top and Bottom.
	Bounds	Specifies the Height, Left, Top, and Width bounds of the field area.
	CanGrow	Specifies whether to increase the field height to fit the field automatically (True or False).
	CanShrink	Specifies whether to decrease the field height to fit the field automatically (True or False).
	ForcePageBreak	Specifies when to insert page breaks. Select None, Before, After, BeforeAndAfter, PageBefore, and PageAfter.
	KeepTogether	Specifies whether the field area should be kept together on the same page (True or False).
	ZOrder	Specifies the layering priority of the field area when two or more fields overlap. The higher the number, the higher the priority.
Template	Class	Specifies the class of the field inherited from the template for the report.

Section Object Settings

Table 4.3
Section Object Settings

Category	Setting	Description
Appearance	BackColor	Specifies the section's background color. Click on the field and select a color from the pull-down menu.
Design	(Name)	Specifies the name that identifies the section.
	Visible	Specifies whether the section will be rendered when the report is run (True or False).
Layout	CanGrow	Specifies whether to increase the section height to fit the content automatically.
	CanShrink	Specifies whether to decrease the section height to fit the content automatically.
	ForcePageBreak	Specifies when to insert page breaks. Select None, Before, After, BeforeAndAfter, PageBefore, and PageAfter.
	Height	Specifies the section height in twips.
	KeepTogether	Specifies whether the section should be kept together on a page (True or False).
	Repeat	Specifies whether the section should be repeated (True or False).
Script	OnFormat	Enter a Visual Basic script that will be executed during the formatting stage of report rendering.
	OnPrint	Enter a Visual Basic script that will be executed in the final stage of rendering, after all report field values have been filled.
Template	Class	Specifies the class of the section inherited from the template for the report. For example, classes can include Detail, Header, Footer, PageHeader, and PageFooter.

Report Object Settings

Table 4.4
Report Object Settings

Category	Setting	Description
Behaviour	DoEvents	Specifies whether to handle messages while rendering reports (True or False).
	ExposeScriptObjects	Specifies whether script objects should be exposed to subreports (True or False).
	GrowShrinkMode	Specifies the method used to process the CanGrow and CanShrink settings.
	IgnoreCase	Specifies case-insensitivity when grouping by a field (True or False).
	IgnoreScriptErrors	Specifies whether to ignore script errors (True or False).
	MaxPages	Specifies the maximum number of pages to render.
	UsePrinterResolution	Specifies whether to use .NET printing support.
Layout	ColumnLayout	Specifies the layout for the columns as Down, Across, or Labels.

Category	Setting	Description
	Columns	Specifies the number of detail columns.
	CustomHeight	Specifies the custom height for the report in twips. Use this property only if you have set PaperSize to Custom.
	CustomWidth	Specifies the custom width for the report in twips. Use this property only if you have set PaperSize to Custom.
	LabelSpacingX	Discounts horizontal label spacing in the design surface.
	LabelSpacingY	Discounts vertical label spacing in the design surface.
	MarginBottom	Specifies the bottom margin for each page in twips.
	MarginLeft	Specifies the left margin for each page in twips.
	MarginRight	Specifies the right margin for each page in twips.
	MarginTop	Specifies the top margin for each page in twips.
	Orientation	Specifies the page orientation as Auto, Portrait, or Landscape.
	PageFooter	Specifes the page footer as AllPages, NotWithReportHdr, NotWithReportFtr, or NotWithReportHdrFtr.
	PageHeader	Specifies the page header as AllPages, NotWithReportHdr, NotWithReportFtr, or NotWithReportHdrFtr.
	PaperSize	Specifies the paper size to one of a wide variety of settings, including: Letter LetterSmall Tabloid Ledger Statement Executive A3 A4 A4Small A5 B4 B5 Folio Quatro Standard10x14 Standard11x17 The Custom option allows you to define your height and width where needed. If you specify Custom, you must also specify the CustomHeight and CustomWidth properties. For more information, see “Paper Size Configuration Guidelines” on page 44.
	Picture	Specifies a background picture for the report body. Click on the field to browse to and select an image file.
	PictureAlign	Specifies how the background picture is aligned. Select from the icons that indicate the positioning relative to the page area.
	PictureScale	Specify the picture scaling method as Clip, Stretch, Scale, Tile, or Hide.
	PictureShow	Specifies where the background picture is displayed. Select NoPages, AllPages, FirstPages, or AllButFirstPage.
	Width	Specifies the width of the report’s detail section in twips. For more information, see “Paper Size Configuration Guidelines” on page 44
Script	OnClose	Enter a Visual Basic script that will be executed when the report finishes rendering.

Category	Setting	Description
	OnError	Enter a Visual Basic script that will be executed when an error occurs.
	OnNoData	Enter a Visual Basic script that will be executed when the report has no data.
	OnOpen	Enter a Visual Basic script that will be executed when the report starts rendering.
	OnPage	Enter a Visual Basic script that will be executed each time a new page is created.

Paper Size Configuration Guidelines

Several settings must be set properly when configuring the desired width for a report. It is a good idea to specify these properly before spending time designing the detailed layout of fields because incorrect width settings can sometimes require moving and resizing fields to correct width problems.

To start, determine the paper size that the report is targeted for, and set the PaperSize report property (see "Report Object Settings" on page 42) accordingly. Next, set the margin properties, which are in units of twips (1/1440 of an inch). The margins are blank in the output, so the actual width available for use in placing report fields and other objects is determined by the paper width minus the left and right margins.

The ruler at the top of the designer has a light-colored segment whose width is equal to the available width (paper width minus right and left margins). This represents the actual region that can contain fields for this paper and margin size.

The next thing to set is the Width property, which can be set either numerically in the property grid or by clicking and dragging the mouse on the right edge of the light-colored canvas rectangle in the designer's main pane. It is recommended to set the Width after the Paper Size and margins are set, and to make the width line up exactly with the ruler width that reflects the paper size and margins. If this is not done, and the Width is set larger than the ruler width, then truncation will likely occur in the printed output when the report is run.

After the Paper Size, Margins, and Width are set up properly, the report is ready for detailed work to put the fields and other objects onto the desired locations on the page. The light-colored canvas region represents the usable area (since the margins have already been accounted for), so fields can be placed right up to the very edge of this region and they will not be truncated or bleed into the margins.

Note The Width property does not have any direct effect on the actual size of the report output. The output size is determined solely by the paper size setting. The Width is mostly used to visually set the canvas background so that the developer can see the available region to place fields in without truncation.

Note The system will not allow the Width property to be set smaller than needed to fit the fields that are currently on the report page. If a smaller width is desired, the fields must be moved or resized to allow the Width to be reduced.

Changing the Data Source

1. In Report Designer, right-click a report under the Reports tab in the toolbox; then select Data Source from the shortcut menu.
2. The Select Data Source dialog box displays. Click to select the table you want to set as the new data source; then click OK.
3. The data source is changed. When you switch to the Data tab in the toolbox, you can see a new set of data-bound fields.

Note When you select a new data source, any field associated with the old data source becomes invalid and will not display data in the report.

Controlling the Maximum Number of Records Per Report Page

You can control the maximum number of records to display on each page of a report by using the page break control.

In Report Designer, place the page break control beneath the data fields on the report; then in the Properties pane, specify the value of its RecordsPerPage property. During report rendering, when the number of records on a page reaches the maximum number of records allowed, the remaining records go to subsequent pages.

Note The records-per-page property does not apply to reports in the Excel format. All records are displayed continuously in an Excel report.

Note A report contains subreports and when the records in a subreport reaches the maximum number per page allowed, the report page breaks even when the parent report has not reached the records per page limit.

Using Translated Labels

You can design reports that can be readily localized into multiple languages. When localized, the system dynamically reads the label master table to determine the appropriate labels to display on your reports. This table contains translated labels that can be specified with a string that specifies the desired label (the label term key, for example “SALES_ORDER”). A report design can use these label term keys so that when the report is run, the system will retrieve the translated label in the appropriate language for the user.

If you attempt to translate a label that does not exist in the label master, you should create a record for it using Label Master Maintenance (36.4.17.24) so that the label can be translated and maintained.

For Label report fields, the value of the Text property will typically be literally displayed in the field when the report is rendered. However, the system can be instructed to automatically select translated labels when the report is rendered. This mode of operation is triggered when the Text property contains a value with the special `${`*`LABEL_TERM`*`}` format, which instructs the system to look up the translated label corresponding to the LABEL_TERM used.

System labels can have short, medium, and long versions, of which the system intelligently selects the largest that will fit into the report field, taking into account factors such as field width and font size. For example, if the field width is not large enough to display the large label for the given text in the given font, the medium label will be displayed automatically if it fits.

Note that the Reporting Framework is flexible about the format of the label terms. For example, you can enter the term for Sales Order as `${SALES_ORDER}` or `${Sales Order}`. The system will replace spaces with underscores and perform case-sensitive matching at run-time.

There are also some variations on this format that allow you to explicitly specify which label should be used:

`${TERM}S` — specifies the short label

`${TERM}M` — specifies the medium label

`${TERM}L` — specifies the long label

`${TERM}!` — specifies a stacked label

Note that in the case of stacked labels, the height of the field must be large enough to fit all the levels in the stacked label or the display of the label will be truncated.

For Calculated fields, the value of the Text property is evaluated as a VBScript expression and the result will be displayed in the field when the report is rendered. One special case is when the expression consists of the name of a report data field, in which case the value of that data field will be displayed when the report is rendered.

For Calculated fields, you can also specify to have integers converted into the words for the number. For instance, you can have "12" displayed as "twelve." The format is as follows:

`${TERM}N` — specifies integer to words conversion

For logical values and value lists, you can specify to have the value rather than the label display:

`${TERM}V` — specifies the value rather than the label

Using .NET Script Objects

.NET Script Objects make it possible to expand the functionality available in the report designer to go beyond VBScript logic. It is now possible to write custom C# (or any .NET language) classes and expose their public methods such that they can be invoked from within VBScript code blocks that execute during report rendering. This is a powerful capability that overcomes the many limitations of the VBScript language and the further limitations imposed by the report framework VBScript interpreter. Now any operation that can be written in a .NET language can be invoked dynamically by reports. This opens the door to performing file I/O, network calls, and custom data structures, for example.

To expose one or more .NET classes to be accessible as report script objects, perform the following steps:

1 Create an XML file that contains script object definitions. An example is given here (the XML file used for the QAD-provided script objects available for this release, which is already deployed):

```
<?xml version="1.0" encoding="utf-16"?>
<ScriptObjectsData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema">
  <ScriptObjects>
    <ScriptObjectData>
      <Name>QAD_Map</Name>
      <Assembly>QAD.Plugin.Reports.ReportFramework</Assembly>
      <Class>QAD.Plugin.Reports.ReportFramework.ScriptObjects.Map</Class>
    </ScriptObjectData>
    <ScriptObjectData>
      <Name>QAD_Map2D</Name>
      <Assembly>QAD.Plugin.Reports.ReportFramework</Assembly>
      <Class>QAD.Plugin.Reports.ReportFramework.ScriptObjects.Map2D</Class>
    </ScriptObjectData>
    <ScriptObjectData>
      <Name>QAD_NumberUtil</Name>
      <Assembly>QAD.Plugin.Reports.ReportFramework</Assembly>
      <Class>QAD.Plugin.Reports.ReportFramework.ScriptObjects.NumberUtil</Class>
    </ScriptObjectData>
  </ScriptObjects>
</ScriptObjectsData>
```

Each .NET class to be exposed as a script object must have a `<ScriptObjectData>` entry. For each, specify:

- Name — the name that the script object will appear with in the report designer VBScript editor.
- Assembly — the name of the .NET DLL assembly in which the class' implementation is contained (without the .dll extension).
- Class — the fully namespace-qualified name of the class to expose from within the DLL.

As a naming convention, QAD-shipped script object classes will always have names that begin with a "`QAD_`" prefix. It is recommended that you not use this prefix when deploying your own custom script object classes, so that they do not conflict with any QAD-shipped script objects (either now or after future upgrades).

2 Deploy the XML file into a `ReportScriptObjects` subdirectory under any QAD .NET UI plug-in directory on the client machine (for example, C:\Program Files\QAD\QAD Enterprise Applications 2010.1 EE\plugins\qad.plugin.reports\ReportScriptObjects). The name of the XML files is arbitrary, as long as they have an `.xml` file extension; however, the directory name must be `ReportScriptObjects`.

3 Deploy the DLLs containing the .NET classes (and any dependent classes) into the same `ReportScriptObjects` directory as the XML. If the class is already in a DLL that is in a plug-in directory, that will also work. However, DLLs in any directories other than plug-in directories or an immediate plug-in subdirectory named `ReportScriptObjects` will not be accessible for script object exposure.

4 When the .NET UI is restarted on that client machine, the script objects will be dynamically loaded into the system. This can be verified by opening the Report Resource Designer program, creating a calculated field, and clicking the ellipses ("...") button in the Text property of the field in the property grid. This will launch the VBScript editor. Click the "Fields>>" button in the lower left of this editor form, and navigate to the Script Objects sub-menu. Your class should no appear in the list of available script objects. If you do not see your script object here, then re-check the above steps, and check the client .NET UI log file for additional error information that might help troubleshoot the issue.

Note Performing the above steps will only deploy the script objects to the client machine on which the steps are performed. To push this deployment out to all client machines, you must apply the same steps to the plug-in package archive on the home server.

Note The report script object will expose just the public methods of the .NET class. Non-public methods will not be exposed for report designer usage.

Important Due to various implementation details, severe performance problems result from the use of certain .NET data types as input parameters or return values of script object methods. Excellent performance is attained by using parameters of type string, bool, or object. Other types (including any numeric types) will result in poor run-time performance when that method is invoked by the report designer. This situation is easy to deal with: when writing the .NET classes that get exposed, simply use the object type for all parameters that have types other than string or bool. Internally, your method can have code to check the type and perform any necessary casting from object to the intended type.

This release of the Reporting Framework contains three Script Object classes provided by QAD:

- QAD_Map — a one-dimensional map data structure that provides array-like capabilities.
- QAD_Map2D — a two-dimensional map data structure that provides array-like capabilities.
- QAD_NumberUtil — performs number-to-word translation and related utilities.

The following sections discuss these classes in more detail.

QAD_Map Script Object

Script Object that implements map functionality exposed in VBScript for use in report designs. The VBScript interpreter in ComponentOne reporting does not support arrays, so this is a way to provide that functionality. This implementation is a map from an integer key to a value. The values are of type object, so they can hold any type passed from VBScript. The keys are restricted to integers since ComponentOne's VBScript interpreter cannot iterate in loops except for for-loops based on integer indices. Restricting keys to integers restricts them to a type that can be iterated over, and that avoids casting issues arising from hard-coded integers in VBScript appearing as "double" numbers in .NET (causing key lookup failures if not explicitly cast as int in .NET).

QAD_Map.Clear()

Clears map contents.

Table 4.5
QAD_Map.Clear() Parameters

Name	Input/Output	Data Type	Description
	Return Value	Boolean	Clears the map contents.

QAD_Map.Contains(key)

Returns true if the map contains an element with the specified key, false otherwise.

Table 4.6
QAD_Map.Contains(key) Parameters

Name	Input/Output	Data Type	Description
key	Input	Integer	Integer key of element whose existence to check for, typed as an object for better performance.
	Return Value	Boolean	Returns true if the map contains an element with the specified key, false otherwise.

QAD_Map.ContainsValue(value)

Returns true if the map contains an element with the specified value, false otherwise.

Table 4.7
QAD_Map.ContainsValue(value) Parameters

Name	Input/Output	Data Type	Description
value	Input	Integer	Value of element whose existence to check for.
	Return Value	Boolean	Returns true if the map contains an element with the specified value, false otherwise.

QAD_Map.FindIndex(value)

Finds a key (index) corresponding to an element having the specified value. If no such element is found, then null will be returned. If more than one element is found with the value, then there is no guarantee which of their indices will be retrieved.

Table 4.8
QAD_Map.FindIndex(value) Parameters

Name	Input/Output	Data Type	Description
value	Input	Integer	Value of element whose key is to be returned.
	Return Value	Integer	Integer key of an element having the specified value, typed as an object for better performance, or null if no elements exist with the specified value.

QAD_Map.Get(key)

Gets the value of the element with the specified key.

Table 4.9
QAD_Map.Get(key) Parameters

Name	Input/Output	Data Type	Description
key	Input	Integer	Integer key of element to get, typed as an object for better performance.
	Return Value	Integer	Value of element having specified key, or null if no element exists with that key.

QAD_Map.GetCount()

Returns a count of the number of elements in the map.

Table 4.10
QAD_Map.GetCount() Parameters

Name	Input/Output	Data Type	Description
	Return Value	Integer	Count, as an integer typed as an object for better performance.

QAD_Map.GetKeys()

Returns the collection of keys in the map as a comma-separated list of strings, sorted in order of key index.

Table 4.11
QAD_Map.GetKeys() Parameters

Name	Input/Output	Data Type	Description
	Return Value	String	Comma-separated list of keys as strings.

QAD_Map.GetValues()

Returns the collection of values in the map as a comma-separated list of strings (each value object’s `ToString()` output), sorted in order of key index (not value index) so as to retain the order alignment with the `GetKeys()` function. Any literal commas in the value strings will be escaped by preceding with a “\”.

Table 4.12
QAD_Map.GetValues() Parameters

Name	Input/Output	Data Type	Description
	Return Value	String	Comma-separated list of values as strings.

QAD_Map.Remove(key)

Removes the element with the specified key. If no such element exits, then nothing is done; no exceptions are thrown.

Table 4.13
QAD_Map.Remove(key) Parameters

Name	Input/Output	Data Type	Description
key	Input	Integer	Integer key of element to remove, typed as object for better performance.

QAD_Map.Set(key, value)

Sets the element with the specified key to have the specified value. If an element already exists with the same key, then its value will be replaced by the specified value.

Table 4.14
QAD_Map.Set(key, value) Parameters

Name	Input/Output	Data Type	Description
key	Input	Integer	Integer key of element to set, typed as object for better performance.
value	Input	Integer	Value of element to set.

QAD_Map.Split(stringToSplit, delimiter, startIndex)

Splits the input string into multiple strings according to the specified delimiter, and for each creates an element in the map whose value is the string, and whose key is a sequential integer starting at the specified start index. Any backslash-escaped delimiters will be treated as literals and not delimiters.

Table 4.15
QAD_Map.Split(stringToSplit, delimiter, startIndex) Parameters

Name	Input/Output	Data Type	Description
stringToSplit	Input	String	String to split.
delimiter	Input	Character	Delimiter to use for splitting string.
startIndex	Input	Integer	Arbitrary start index; must be an integer but is typed as object for better performance.
	Return Value	Integer	Integer value indicating the number of elements that the input string was split into, typed as object for better performance.

QAD_Map.Split(stringToSplit)

Splits the input delimited list of values into a string array, recognizing escaped delimiters as being preceded by backslash characters.

Table 4.16
QAD_Map.Split(stringToSplit) Parameters

Name	Input/Output	Data Type	Description
stringToSplit	Input	String	String to split.
	Return Value	Integer	Integer value indicating the number of elements that the input string was split into, typed as object for better performance.

QAD_Map2D Script Object

Script Object that implements two-dimensional map functionality exposed in VBScript for use in report designs. The VBScript interpreter in ComponentOne reporting does not support arrays, so this is a way to provide that functionality. This implementation is a map from two integer keys to a value. The values are of type object, so they can hold any type passed from VBScript. The keys are restricted to integers since ComponentOne's VBScript interpreter cannot iterate in loops except for for-loops based on integer indices. Restricting keys to integers restricts them to a type that can be iterated over, and that avoids casting issues arising from hard-coded integers in VBScript appearing as "double" numbers in .NET (causing key lookup failures if not explicitly cast as int in .NET). This class is a direct generalization of the Map class (which is one-dimensional), and essentially provides an array of arrays, whereas if we only had the Map class, a given report would be limited to a single array.

QAD_Map2D.Clear()

Clears the entire 2-D map contents.

QAD_Map2D.Clear(key1)

Clears the contents of the 1-D map with the specified key1 value.

Table 4.17 QAD_Map2D.Clear(key1) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of map to clear, typed as an object for better performance.

QAD_Map2D.Contains(key1)

Returns true if the 2-D map contains an 1-D map with the specified key1, false otherwise.

Table 4.18 QAD_Map2D.Contains(key1) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of map whose existence to check for, typed as an object for better performance.
	Return Value	Boolean	True if a 1-D map exists with the specified key1, false otherwise.

QAD_Map2D.Contains(key1, key2)

Returns true if the map contains an element with the specified key1 and key2 values, false otherwise.

Table 4.19 QAD_Map2D.Contains(key1, key2) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of element whose existence to check for, typed as an object for better performance.
key2	Input	Integer	Integer key2 of element whose existence to check for, typed as an object for better performance.
	Return Value	Boolean	True if the map contains an element with the specified key1 and key2 values, false otherwise.

QAD_Map2D.ContainsValue(key1, value)

Returns true if the map contains an element with the specified value, false otherwise.

Table 4.20 QAD_Map2D.ContainsValue(key1, value) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of element whose existence to check for.
value	Input	Integer	Value of element whose existence to check for.
	Return Value		True if the map contains an element with the specified value, false otherwise.

QAD_Map2D.FindIndex(key1, value)

Finds a key2 (index) corresponding to an element having the specified key1 and value. If no such element is found, then null will be returned. If more than one element is found with the value, then there is no guarantee which of their indices will be retrieved.

Table 4.21 QAD_Map2D.FindIndex(key1, value) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of element whose existence to check for, typed as an object for better performance.
value	Input	Integer	Value of element whose key is to be returned.
	Return Value	Integer	Integer key2 of an element having the specified key1 and value, typed as an object for better performance, or null if no elements exist with the specified key1 and value.

QAD_Map2D.Get(key1, key2)

Table 4.22 QAD_Map2D.Get(key1, key2) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of element to get, typed as an object for better performance.
key2	Input	Integer	Integer key2 of element to get, typed as an object for better performance.
	Return Value	Integer	Value of element having specified key1 and key2, or null if no element exists with those keys.

QAD_Map2D.GetCount()

Returns a count of the number of key1 elements in the map (that is, the number of 1-D maps that exist).

Table 4.23 QAD_Map2D.GetCount() Parameters

Name	Input/Output	Data Type	Description
	Return Value	Integer	Count, as an integer typed as an object for better performance.

QAD_Map2D.GetCount(key1)

Returns a count of the number of elements in the map having the specified key1.

Table 4.24 QAD_Map2D.GetCount(key1) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of map whose elements to count, typed as an object for better performance.
	Return Value	Integer	Count, as an integer typed as an object for better performance.

QAD_Map2D.GetKeys()

Returns the collection of key1 values in the 2-D map as a comma-separated list of strings, sorted in order of key index.

Table 4.25 QAD_Map2D.GetKeys() Parameters

Name	Input/Output	Data Type	Description
	Return Value		Comma-separated list of key1 values as strings.

QAD_Map2D.GetKeys(key1)

Returns the collection of key2 values in the 1-D map specified by key1, as a comma-separated list of strings, sorted in order of key index.

Table 4.26 QAD_Map2D.GetKeys(key1) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of map whose keys to return, typed as an object for better performance.
	Return Value	String	Comma-separated list of key2 values as strings.

QAD_Map2D.GetValues(key1)

Returns the collection of values in the 1-D map specified by `key1`, as a comma-separated list of strings (each value object's `ToString()` output), sorted in order of key index (not value index) so as to retain the order alignment with the `GetKeys()` function and allow subsequently feeding the output to the `Split()` method to copy to another map `key1`.

Table 4.27 QAD_Map2D.GetValues(key1) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of map whose values to return, typed as an object for better performance.
	Return Value	String	Comma-separated list of values as strings.

QAD_Map2D.Remove(key1)

Removes the 1-D map with the specified key1. If no such map exits, then nothing is done; no exceptions are thrown.

Table 4.28 QAD_Map2D.Remove(key1) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of map to remove, typed as object for better performance.

QAD_Map2D.Remove(key1, key2)

Removes the element with the specified key1 and key2. If no such map exits, then nothing is done; no exceptions are thrown.

Table 4.29 QAD_Map2D.Remove(key1, key2) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of element to remove, typed as object for better performance.
key2	Input	Integer	Integer key2 of element to remove, typed as object for better performance.

QAD_Map2D.Set(key1, key2, value)

Sets the element with the specified key1 and key2 to have the specified value. If an element already exists with the same keys, then its value will be replaced by the specified value.

Table 4.30 QAD_Map2D.Set(key1, key2, value) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of element to set, typed as object for better performance.
key2	Input	Integer	Integer key2 of element to set, typed as object for better performance.
	Return Value	Integer	Value of element to set.

QAD_Map2D.Split(key1, stringToSplit)

Splits the input string into multiple strings using a comma delimiter, and for each creates an element in the 1-D map keyed by key1 whose value is the string, and whose key is a sequential integer starting at value 1.

Table 4.31 QAD_Map2D.Split(key1, stringToSplit) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of map to create the split elements in, typed as object for better performance.
stringToSplit	Input	String	String to split.
	Return Value	Integer	Integer value indicating the number of elements that the input string was split into, typed as object for better performance.

QAD_Map2D.Split(key1, stringToSplit, delimiter, startIndex)

Splits the input string into multiple strings according to the specified delimiter, and for each creates an element in the 1-D map keyed by key1 whose value is the string, and whose key is a sequential integer starting at the specified start index.

Table 4.32 QAD_Map2D.Split(key1, stringToSplit, delimiter, startIndex) Parameters

Name	Input/Output	Data Type	Description
key1	Input	Integer	Integer key1 of map to create the split elements in, typed as object for better performance
stringToSplit	Input	String	String to split.
delimiter	Input	Character	Delimiter to use for splitting string.
startIndex	Input	Integer	Arbitrary start index; must be an integer but is typed as object for better performance.
	Return Value	Integer	Integer value indicating the number of elements that the input string was split into, typed as object for better performance.

QAD_NumberUtil Script Objects

Script Object that implements number-to-word services exposed in VBScript for use in report designs.

QAD_NumberUtil.SplitNumberLeft(number)

Returns an integer representing the portion of the input number that is to the left of the decimal point.

Table 4.33 QAD_NumberUtil.SplitNumberLeft(number) Parameters

Name	Input/Output	Data Type	Description
number	Input	Integer	Input number to split.
	Return Value	Integer	Returns an integer representing the portion of the input number that is to the left of the decimal point.

QAD_NumberUtil.SplitNumberRight(number, numDigitsAfterDecimalPoint)

Returns an integer representing the portion of the input number that is to the right of the decimal point, including only numDigitsAfterDecimalPoint places. For example, SplitNumberRight(35.1234, 2) returns 12.

Table 4.34 QAD_NumberUtil.SplitNumberRight(number, numDigitsAfterDecimalPoint) Parameters

Name	Input/Output	Data Type	Description
number	Input	Integer	Input number to split
numDigitsAfterDecimalPoint	Input	Integer	Number of digits after the decimal point to include in the returned integer.
	Return Value	Integer	Returns an integer representing the portion of the input number that is to the right of the decimal point, including only numDigitsAfterDecimalPoint places.

QAD_NumberUtil.ToWords(number)

Converts the input number into word strings in the specified language. If there is any portion to the right of the decimal point, it will be truncated before converting. Example usage: printing the value onto a check (for example, input of 104 would result in output of “one hundred four,” in the English language).

Table 4.35 QAD_NumberUtil.ToWords(number) Parameters

Name	Input/Output	Data Type	Description
number	Input	Integer	Number to convert to words.
	Return Value	String	String representation of number in language of the report Context; throws exception if an error occurs.

Creating a Master-Detail Report Using Subreports

About Subreports

Subreports are regular reports contained in a field in another report (the main report). Subreports are usually designed to display detail information based on a current value in the main report, in a master-detail scenario. You can create multiple subreports in a main report.

For example, the main report contains product categories and the subreport in the Detail section contains product details for the current category.

Creating a Master-Detail Report

1 Create a master report.

Create a basic report using Report Wizard and, optionally, further customize the report using the design tools provided in Report Designer. For details on creating a report definition using Report Wizard, see “Creating a Report Definition” on page 14.

2 Create a subreport.

a In Report Designer, right-click a report definition under the Reports tab in the toolbox and choose Create Subreport from the shortcut menu.

b The Report Wizard window displays. Follow the same steps you used when creating the basic report. See “Creating a Report Definition” on page 14.

c When you have created a basic detail report using Report Wizard and return to the Report Designer main screen, in the Detail section of your report, click and drag the mouse pointer to make the field for the subreport. The subreport is embedded in the master report.

d Optionally, right-click the subreport field and select Edit Subreport from the shortcut menu to fully open the subreport in the Design pane; then further customize the report using the design tools provided in Report Designer.

e Optionally, you can repeat the previous steps to add multiple subreports to the master report.

3 Link the subreport to the master report

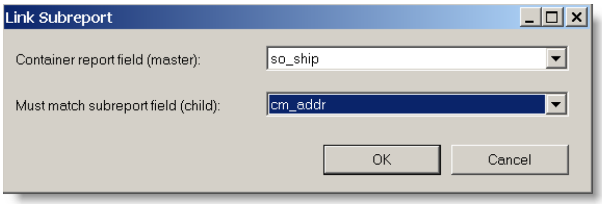
The master-detail relationship is controlled by the Text property of the subreport field. This property should contain an expression that evaluates into a search condition that can be applied to the subreport data source.

The Report Designer can build this expression automatically for you. Complete the following steps:

a Right-click the subreport field and select Link Subreport from the menu.

b A dialog box appears and lets you to select which fields should be linked. Once you make a selection and click OK, the Report Designer builds the link expression and assigns it to the Text property of the subreport field in the background.

Fig. 4.8
Link Subreports



Container report field (master): so_ship

Must match subreport field (child): cm_addr

4 The master-detail report is created.

Note If you do not link the subreport to the master report, a Descartes Accumulate of data will be displayed on the report.

Adding Unbound Images to the Report

Unbound images are static images such as logos and watermarks that are not stored in the database.

1. Click the Controls tab in the Report Designer toolbox.
2. Click and expand the Common Components tree to display all available components.
3. Click the Unbound Picture button.
4. A file browse dialog box appears. Locate and select the image file you want to add to the report.
5. Move the mouse pointer over the report and the cursor changes into a cross-hair. Click on your report where you would like to place the image, and then resize the field to show the image.

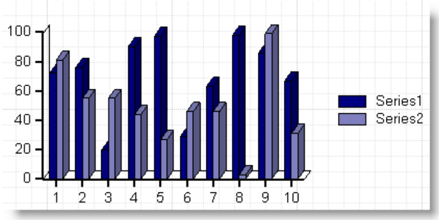
If you change your mind, press Ctrl+Z or click the Undo button to cancel the operation.

You can also add fields by copying and pasting existing images, or by holding down the CTRL key and dragging a field or group of fields to a new position to create a copy.

Adding Charts to the Report

1. Click the Controls tab in the Report Designer toolbox.
2. Click and expand the Common Components tree to display all available components.
3. Click the Chart button.
4. Move the mouse pointer over the report and the cursor changes into a cross-hair. Click on your report where you would like to place the chart, and then resize the field to show the image.

Fig. 4.9
Link Subreports



5. With the chart selected, modify the following required chart properties in the Properties window:
   - DataX: Specify the data series to plot the X axis of the chart; for example, month(sod_date).
   - DataY: Specify the data series to plot the Y axis of the chart, separating multiple data series using semicolons; for example, sod_qty_ord; sod_qty_pick.
   - ChartType: Specify the type of the chart: bar, column, pie, scatter, line, or area.
6. Optionally modify other properties to tailor the chart to your specific needs.

Adding Search Criteria Report Parameter

You can add a parameter to specify a location for the search criteria on a report using the Report Parameter Maintenance program:

1. Open Report Parameter Maintenance.
2. In Report Code, choose the resource your report is using.
3. In Param Name, choose `sys_search_criteria_display`.
4. Check the Extend checkbox.
5. Continue to the Value Settings frame and set Value to Header, Footer, or None.
6. Click Next through the remaining frames of the program to close it.
7. Run the report.

The report search criteria now is included in the location you specified.

Note Report Parameter Maintenance's other options and the additional report parameters available from the Param Name field are currently reserved for future use.

Exporting Report Metadata

You can export report metadata to an XML file for review and debug a report design.

To export report metadata

1. On the toolbar, choose Actions and click Export Metadata.
2. In the Save As dialog box, specify the name of the XML file and where you want to save the file. By default, the file is named after the current report with `_meta.xml` appended.

You can now view the report metadata in an XML format.

Exporting and Importing Report Data

You can export the data for a report to an XML file for review, testing, and debugging purposes. You can then import an XML file containing report data and run it. The exporting and importing actions will not take place until a preview of the report is run from the Report Designer.

Importing report data from an XML file is a useful tool for scenarios such as:

- Repeated testing of a report whose data source takes a long time to run.
- Manually changing data values for the purpose of quickly testing report design logic, without having to change them in the live production database.

To export report data

1. On the toolbar, choose Actions and click Enable Data Export.
2. In the Save As dialog box, specify the name of the XML file and where you want to save the file. By default, the file is named after the current report with `_data.xml` appended.

   When you preview a report, the report data will be exported to the XML file you specified.

   Note You cannot enable both exporting and importing for the same XML file at the same time.
3. Click Preview.
4. In the report viewer, click Run.

   The report is displayed on the screen and the report data is exported to the XML file you specified.

To import and run report data

1. On the toolbar, choose Actions and click Enable Data Import.
2. In the Save As dialog box, choose the XML file containing report data that you want to import.

   Note You cannot enable both exporting and importing for the same XML file at the same time.
3. Click Preview.
4. In the report viewer, click Run.

Working with Templates

A report template is special kind of report definition that cannot be rendered directly by itself, but instead can be used to control certain aspects of the rendering of other reports. When designing a report, a template can be specified (optionally) in which case the report can inherit many kinds of attributes from the template, such as field colors and fonts. If at a later time these attributes are changed in the template, those changes will be seen in every report that is using that template.

Any given report can inherit from at most one template, but a given template may be used to control any number of reports. Thus templates enable report developers to making changes in a single place (the template) which will have a mass effect on many reports. This is a powerful tool that can assist the report development process in many ways, such as reducing initial development time, enforcing common standards across reports, and quickly implementing future changes to these standards.

There are three general types of report properties that can be governed by templates:

- Top-level report properties (e.g. paper size, margins)
- Section properties (e.g. The back color of the PageHeader section)
- Field properties (e.g. The font and ForeColor of fields)

In addition to inheriting properties, reports can also inherit fields from a template. For instance, a template might contain fields in the page header that display date, time, domain, and a corporate logo. These fields will be automatically added to all reports using that template.

Templates can be edited in the Template Designer, which is very similar to the Report Designer program.

Elements in the report template represent classes or styles that can be applied to corresponding elements in a report definition based on a class-mapping relationship.

A field defined in the report template represents a field class identified by a unique class name. When the field class is applied to a field in a report definition, most of its properties are carried over to the field so that the field takes on the same formatting and layout.

The header, page header, page footer, footer, or a group section defined in the report template represents a section class identified by a unique class name. When the section class is applied to a section in a report definition, it is virtually copied over to the report definition complete with all the elements in it.

About Template Designer

You design report templates in Template Designer. If you are familiar with Report Designer, there is no learning curve in using Template Designer.

Template Designer is almost identical to Report Designer, except for the following differences.

- The Toolbox only displays the Controls tab which contains a limited set of control components that can be used in the template: Label, Calculated Field, Common Field, Unbound Pictures, Line, Rectangle, and Chart.
- The toolbar contains the following buttons that are specific to Template Designer:

Table 4.36
Template Designer Specific Toolbar Buttons

Button	Name	Description
	Edit Template	Open the current report template file in code mode for editing.
	Add Group	Add groups to create new section classes.
	Config	Configure default fields and sections mapping.
	Import	Import report template files.
	Export	Export report template files.

Creating a New Report Template

1. Launch Template Designer. Type Template Designer in the menu search field and press Enter.
2. Click the New button on the toolbar.

3 In the Create Template dialog box, enter a unique template name and click OK.

Important QAD-provided built-in reports, report resources, and templates all begin with “QAD_”. Do not create or modify reports, report resources or templates with this prefix. Otherwise, your customized changes will get overwritten during system upgrades from QAD.

4 In the Design pane, create and format field classes in the same way as you work with fields when working with a report definition. Provide unique class names for the classes. See “Enhancing the Report with Fields” on page 37.

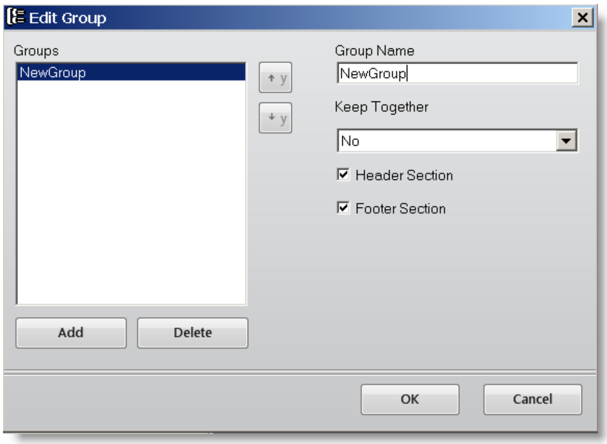
5 If you want to define a header, page header, page footer, and footer section class name, click the default section name and enter a new name in the (name) field in the Properties pane.

6 If you want to add new sections, use the following steps:

a Click the New Group button on the toolbar.

b In the Edit Group dialog box, click Add and specify the properties for the new group.

Fig. 4.10
Add Group



Edit Group

Groups: NewGroup

Group Name: NewGroup

Keep Together: No

☑ Header Section

☑ Footer Section

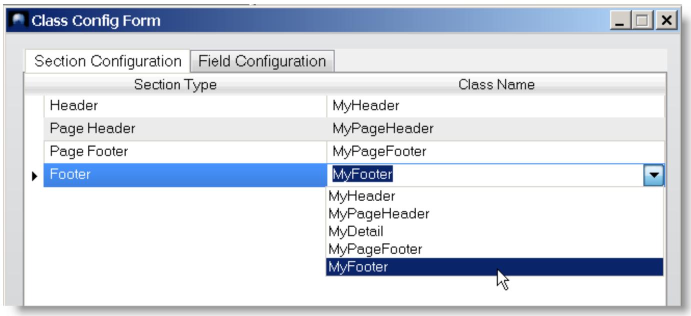
c Click OK.

7 Configure the default section and field class mapping to specify the default classes to be applied to the corresponding sections and fields in the report definition.

a Click the Configure button on the toolbar. The Class Configuration Form dialog box appears.

b Under the Section Configuration tab, specify a class name for each section type.

Fig. 4.11
Section Configuration

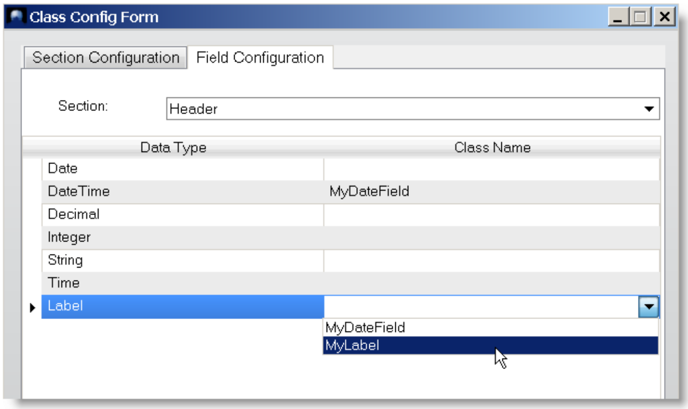


	Section Type	Class Name
	Header	MyHeader
	Page Header	MyPageHeader
	Page Footer	MyPageFooter
▸	Footer	MyFooter

MyHeader
MyPageHeader
MyDetail
MyPageFooter
MyFooter

c Under the Field Configuration tab, for each data type, select a section in the template and specify a class defined within that section.

Fig. 4.12
Field Configuration



Section: Header

	Data Type	Class Name
	Date	
	DateTime	MyDateField
	Decimal	
	Integer	
	String	
	Time	
▸	Label	

MyDateField
MyLabel

d Click OK.

8 Back in the Template Designer main screen, click the Save button on the toolbar to save the template.

Applying Report Templates

After you design a report template, you can apply it to multiple report definitions to enforce a consistent look and feel across all these reports.

Applying a report template to a report definition applies all the mapped classes defined in the template to the corresponding classes in the report definition based on a class-mapping relationship. This takes place on two levels:

- On the section level, when a section class—a class defined by the header, footer, page header, page footer, detail, or a group section in the report template—is applied to the mapped header, footer, page header, page footer section in the report definition, all the contents in the template section are copied over to the report definition section.

  Note This does not apply to the detail and group sections in the report definition.

- On the field level, when a field class—a class defined by the field in the report template—is applied to a mapped field in the report definition, a predefined set of properties are copied from the template field to the report definition field.

The class-mapping relationships are defined as a step in “Creating a New Report Template” on page 61.

To apply an existing report template to a report definition

Do one of the following:

- When “Creating a Report Definition” using Report Wizard, in the Select Template step of the Report Wizard, select the report template from the template list.
- In Report Designer, click the Manager button on the toolbar and assign the report template to the report definition in Report Definition Manager. For details, see “Managing Report Definition Files” on page 31.

Once the report template is applied to the report definition, the changes in layout and formatting immediately take effect in Report Designer.

Customizing Template-Based Report Definitions

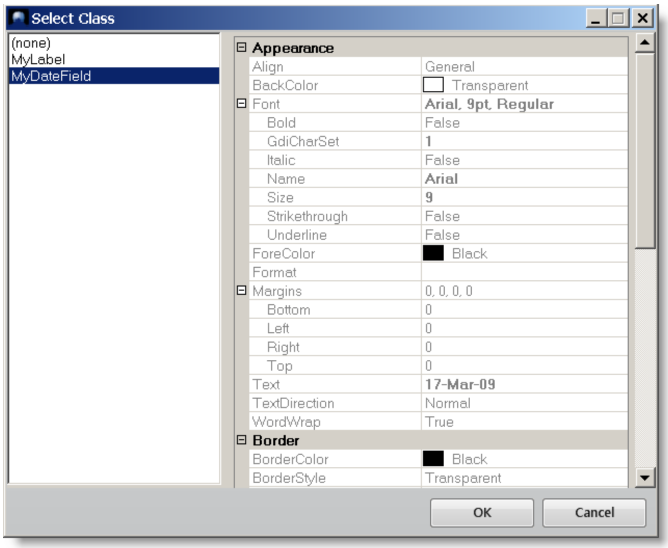
Applying a report template to a report definition applies classes to affected sections and fields in the report definition across the board. You can still customize the report definition by applying a different class to individual report element or modifying their other properties in Report Designer.

However, when you manually change an element’s properties including applied class in a template-based report definition and confirm the change, the element is disengaged from the report template in the report definition and will no longer be affected by the template. If at a later time it is desired to re-engage a property to the template you can select the property in the property grid, and then click the Use Default Value button at the top of the property grid. This will change the value of the property to match the value in the template, and re-engage the connection of that property to the template such that any future changes to that property in the template will be passed to the report.

Applying a Report Template Class to an Individual Element

1. In Report Designer, select the element by either clicking it in the Application area, or selecting it from the element list in the Properties window.
2. In the Properties window, expand Template, and click the Browse button next to the Class property.
3. The Select Class dialog box displays all the classes of the matching type (section or field) defined in the current report template. Select the class you want to apply and click OK.

Fig. 4.13
Select Class



(none)
MyLabel
MyDateField

⊟ **Appearance**	
Align	General
BackColor	□ Transparent
⊟ Font	Arial, 9pt, Regular
Bold	False
GdiCharSet	1
Italic	False
Name	Arial
Size	9
Strikethrough	False
Underline	False
ForeColor	■ Black
Format	
⊟ Margins	0, 0, 0, 0
Bottom	0
Left	0
Right	0
Top	0
Text	17-Mar-09
TextDirection	Normal
WordWrap	True
⊟ **Border**	
BorderColor	■ Black
BorderStyle	Transparent

4 The class is applied to the element with immediate changes in layout and formatting.

Exporting Report Templates

Perform the following steps to export report templates from the database to XML files:

1 In the Template Designer, click the Export button.

2 Select the check boxes in the list for the templates that you want to export. You can use the Check All and Uncheck All to select or clear all the check boxes.

3 Specify the directory to which you want to export template data files in the Export Directory field. You can open a Browse for Folder dialog to select the folder by clicking on the button to the right of the field.

4 Click Export to begin the export process.

5 After the export process is complete, look at the Status field for the items in the list to check whether the export completed successfully or if errors occurred.

Importing Report Templates

Perform the following steps to import report templates from XML files to the database.

1 In the Template Designer, click the Import button.

2 Specify the directory from which to import template data files in the Import Directory field. You can open a Browse for Folder dialog to select the files by clicking on the button to the right of the field. You must enter a valid directory in this field.

3 Click the Refresh button. The system then inspects the directory and displays a list of all template data files in that directory.

4 Select the Update check box to indicate whether you want to overwrite existing templates in the system with the imported templates. If the box is selected, the template will be imported regardless of whether it already exists in the system, possibly overwriting the contents previously in the system.

5 Select the check boxes in the list for the reports that you want to import. You can use the Check All and Uncheck All to select or clear all the check boxes.

6 Click the Import button to begin the import process.

7 After the import process is complete, look at the Status field for the items in the list to check whether the import completed successfully or if errors occurred.

Managing Templates

Starting with the QAD Enterprise Applications 2010 release, the QAD standard reports included with the product use one of the following templates:

- Active_Template_Landscape —the template typically used by QAD standard reports with landscape orientation.
- Active_Template_Portrait — the template typically used by QAD standard reports with portrait orientation.

Additionally, four QAD standard templates are included:

- QAD_Standard_Template_A4_Landscape — the QAD standard template for the A4 (210 x 297 mm, or 8.27 x 11.69 inches) paper size with landscape orientation.
- QAD_Standard_Template_A4_Portrait — the QAD standard template for the A4 paper size with portrait orientation.
- QAD_Standard_Template_Letter_Landscape — the QAD standard template for the Letter (8.5 x 11 inches) paper size with landscape orientation.
- QAD_Standard_Template_Letter_Portrait — the QAD standard template for the Letter paper size with portrait orientation.

As shipped, the contents of Active_Template_Landscape are identical to QAD_Standard_Template_Letter_Landscape, and the contents of Active_Template_Portrait are identical to QAD_Standard_Template_Letter_Portrait. If template customizations are desired, they can be done in the Active templates (which the QAD reports reference), and the original QAD_ templates should be kept unmodified in case any changes might need to be rolled back.

Careful management of these templates is highly recommended.

Template Modification Recommendations

The Active_Template_Landscape and Active_Template_Portrait are used by the QAD standard reports as their templates. Modifying these two active templates will in turn modify the format of all the reports that use these templates; such changes will take effect in the system as soon as any template changes are saved. Using Active_Template_Landscape and Active_Template_Portrait as your active templates gives you a powerful way to make a change to the format of many reports all at once.

When upgrades are installed for the Reporting Framework, the QAD-shipped templates will generally get over-written.

Before you modify the Active templates, QAD recommends that you make backup copies of them by exporting them using the Template Designer’s Export function. You can specify the directory to which you want to export the templates from the Export Template window. QAD recommends that these files be backed up or version controlled.

As an alternative to making direct changes to the Active templates, you can also replace an Active template with some other template. For example, the default Active_Template_Portrait is based on QAD_Standard_Template_Letter_Portrait, which has the 8.5 x 11 inch paper size. If you want all QAD standard reports to use the A4 paper size, you can replace Active_Template_Portrait with QAD_Standard_Template_A4_Portrait. To do so, first save a copy of the current Active_Template_Portrait by exporting it. Next, open QAD_Standard_Template_A4_Portrait and save it as Active_Template_Portrait.

QAD discourages direct modification of the Active templates in the Template Designer because as soon as those changes are saved, they will affect all the QAD standard reports without giving the designer a chance to test the template changes. Instead, QAD recommends that you make a copy of the template that you want to change, give the copy a new name, and then modify the copy and test it with a test report. Once the changes have been tested and are satisfactory, the copied report can be saved as the name of original template, at which point the changes will effectively be rolled out to the reports in the live system.

Upgrading and Templates

When you upgrade to a new version of the software, after the QAD Enterprise Applications 2010 release, you should first be sure to export and save the templates you are using, especially the two Active templates (Active_Template_Portrait and Active_Template_Landscape). This way your customized versions can be re-imported if the upgrade overwrites these templates.

Importing and Exporting Report Resources

Exporting Report Resources

Use Report Resource Export to export data of specified report resources from the database into `.xml` files. This function lets you back up report resources or create report resource files to be imported into another system. Perform the following steps to export report resources:

1 Click Search to list all the reports in the database. Because there can be many reports in the database, use the From Report and To Report fields to filter the list. For example, to list only the reports whose first letter is between the letter a and the letter d (inclusive), enter a in the From Report field and d in the To Report field and then click Search.

2 Select the check boxes in the list for the reports that you want to export. You can use the Check All and Uncheck All to select or clear all the check boxes.

3 Specify the directory to which you want to export report resource data files in the Export Directory field. You can open a Browse for Folder dialog to select the folder by clicking on the button to the right of the field.

4 Click Export to begin the export process.

5 After the export process is complete, look at the Status field for the items in the list to determine whether the export completed successfully or if errors occurred.

Importing Report Resources

Use Report Resource Import to import report resource data files into the system. Perform the following steps to load reports from files into your system database, at which point you will be able to run the reports:

1 Specify the directory from which to import report resource data files in the Import Directory field. You can open a Browse for Folder dialog to select the files by clicking on the button to the right of the field. You must enter a valid directory in this field.

2 Click the Refresh button. The system will then inspect the directory and display a list of all report resource data files in that directory.

3 Select the Update check box to indicate whether you want to overwrite existing report resources in the system with the imported report resource data. If it is not selected and a report already exists in the system with the same report code as the report in the XML file, no import will be attempted. If the box is selected, the report will be imported regardless of whether the report code already exists in the system, possibly overwriting the contents previously in the system.

4 Select the checkboxes in the list for the reports that you want to import. You can use the Check All and Uncheck All to select or clear all the checkboxes.

5 Click the Import button to begin the import process.

6 After the import process is complete, look at the Status field for the items in the list to determine whether the import completed successfully or if errors occurred.

Chapter 5

Running Reports

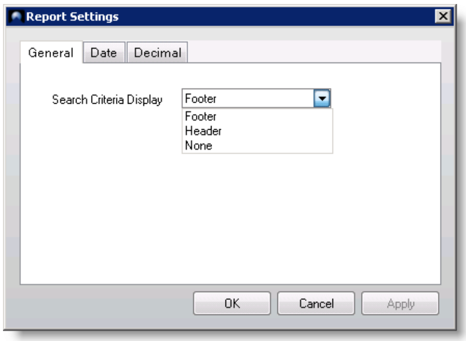
Configuring Report Settings

Use Report Settings to customize how certain elements of data will be displayed in the rendered report.

In the Filter screen, click Settings on the toolbar to bring up the Report Settings dialog box.

- Under the General tab, specify whether to display search criteria in the report, and if yes, whether to display this information in the report header or footer.

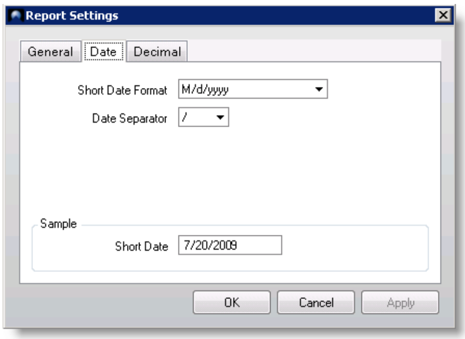
Fig. 5.1
Report Settings: General



Search Criteria Display Footer

- Under the Date tab, select a format for the dates to be displayed in the report and specify a date separator. You can see a sample of the date format you specify at the bottom of the dialog box.

Fig. 5.2
Report Settings: Date



Short Date Format M/d/yyyy

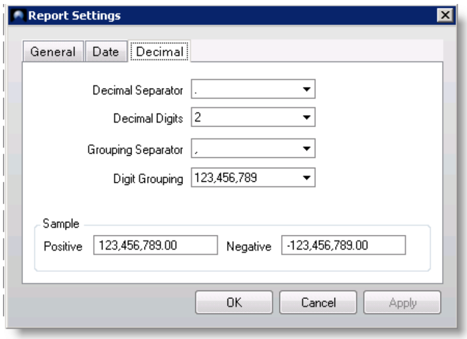
Date Separator /

Sample

Short Date 7/20/2009

- Under the Decimal tab, specify how numbers will be displayed in the report, including decimal separator, decimal digits, grouping separator, and grouping format. A sample number is displayed at the bottom of the dialog box.

Fig. 5.3
Report Settings: Number



Decimal Separator .
Decimal Digits 2
Grouping Separator ,
Digit Grouping 123,456,789

Positive 123,456,789.00 Negative -123,456,789.00

Note After you manually format a date or number field in a report in Report Designer, the reporting format settings will no longer apply to these fields.

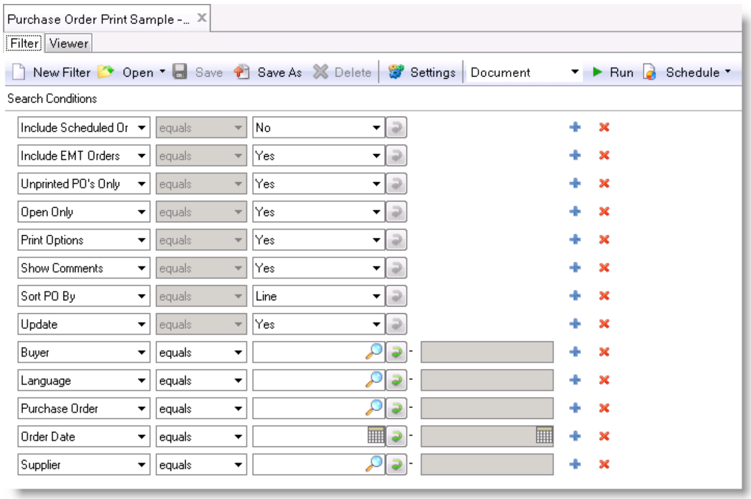
Running Reports

After a report is designed, you can set filter criteria to filter data in the report, run the report, and send it to different output destinations.

To run a report

1 Double-click the report menu item in the Applications menu tree or right-click it and choose Open from the shortcut menu. The Report Filter screen is displayed in the application area.

Fig. 5.4
Report Filter



Include Scheduled Or	equals	No
Include EMT Orders	equals	Yes
Unprinted PO's Only	equals	Yes
Open Only	equals	Yes
Print Options	equals	Yes
Show Comments	equals	Yes
Sort PO By	equals	Line
Update	equals	Yes
Buyer	equals	
Language	equals	
Purchase Order	equals	
Order Date	equals	
Supplier	equals	

2 By default, a report will display all the records available in the source data. However, you may want to retrieve just a certain range of records in the report; for example, sales records between last September and this March. You do this by setting search conditions to filter data in the report. You can also use filters to load existing search conditions. For information about using filters, see “Using Report Filters” on page 74.

Note The default report filter parameters are predefined in the data source proxy program and you can only change them by modifying the program code.

The query constructor provides extensive, configurable filter capabilities that let you create both simple and complex queries. Choose a search operator from the drop-down list.

a The search operators include the following:

- equals
- not equals
- contains
- range
- starts at (the default)
- greater than
- less than
- is null
- is not null

b If you choose the Range operator, enter a beginning value of the range in the first search box. Optionally, enter an ending value of the range in the second search box.

c To refine your search further, click the plus (+) icon to add another search row. You can add as many rows as needed, each with different search values and operators. When you specify several criteria, note that multiple criteria for the same field are treated as a logical AND condition.

d To remove a search criteria row, click on the delete (X) icon.

e Optionally, save the new search conditions as a filter for future reuse. For information about working with filters, see “Using Report Filters” on page 74.

3 On the toolbar, select a layout from the Layout pull-down list. (Available for a report resource with multiple report definitions.) The default layout is listed in bold text.

4 On the toolbar, select an output format from the pull-down list before the Run button. You can choose from three output formats when the report is run:

- Document — The report is displayed in the Report Viewer window.
- PDF — The report is rendered as a PDF file. You can save the file and open it in the Report Viewer window.
- PDF Read-only — The report is rendered as a read-only PDF file. It has a random password that prevents tampering with the document.
- TIFF — The report is rendered as a Tagged Image File Format (TIFF) file.
- RTF — The report is rendered as a Rich Text Format (RTF) file.
- Excel — The report is rendered as a Microsoft Excel (.xls) file. You can save the file and open it in the Report Viewer window.
- Plain Text — The report is rendered as a plain text (.txt) file.

The Document and PDF formats offer the highest display quality. Some images and colors may appear differently in some render types (for example, TIFF and RTF), and may not appear at all in others (for example, Plain Text).

5 Press the Enter key or click Run. A report generation progress bar appears. When report generation is complete, the report is displayed in the Report Viewer window directly or opened as a PDF or Excel file depending on which output format you selected.

Running Reports Directly From Browses

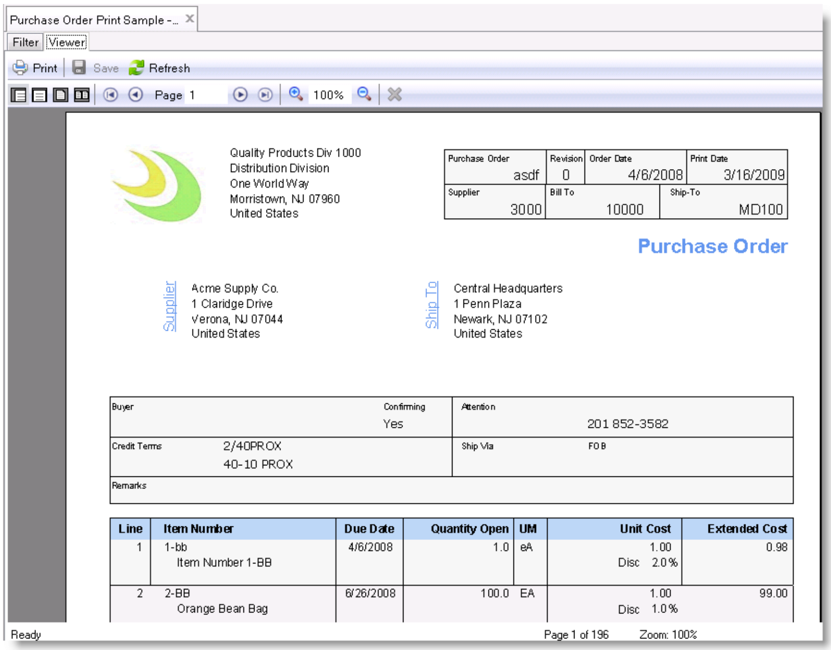
You can directly run reports from browses by selecting Report from the Action menu in the browse screen. The sorting, grouping, and search criteria in the browse are all carried over to the report, which uses the browse as its data source. You can further filter data in the report by defining new search criteria in the Filter screen. This is an alternative to generating a browse-based report by creating a report resource and manually specifying a browse as its data source. However, since you do not create a report resource, the browse-based report created this way cannot be assigned to a menu. A built-in template is used to render browse-based reports.

Note Unlike in the browse, the column header by which data is grouped is case-sensitive in the report. For example, while EA and ea are considered the same column header in the browse, they are treated as different field names with data separately grouped under them in the report.

Viewing, Exporting, and Printing a Report

In Report Viewer, use the toolbar buttons to navigate through the report and perform other functions such as saving and printing.

Fig. 5.5
Report Viewer



Quality Products Div 1000
Distribution Division
One World Way
Morristown, NJ 07960
United States

Purchase Order	Revision	Order Date	Print Date
asdf	0	4/6/2008	3/16/2009

Supplier	Bill To	Ship-To
3000	10000	MD100

Purchase Order

Supplier
Acme Supply Co.
1 Claridge Drive
Verona, NJ 07044
United States

Ship To
Central Headquarters
1 Penn Plaza
Newark, NJ 07102
United States

Buyer
Confirming: Yes
Attention: 201 852-3582
Credit Terms: 2/40PROX
40-10 PROX
Ship Via
FOB
Remarks

Line	Item Number	Due Date	Quantity Open	UM	Unit Cost	Extended Cost
1	1-bb Item Number 1-BB	4/6/2008	1.0	eA	1.00 Disc 2.0%	0.98
2	2-BB Orange Bean Bag	6/26/2008	100.0	EA	1.00 Disc 1.0%	99.00

Button	Name	Description
🖨	Print	Send the report to a printer.
💾	Save	Save the report to a specified location.
🔄	Refresh	Regenerate the report using your last setting.
▤	Actual Size	Display the report in its actual size.
☰	Page Width	Fit the report to the width of the Report Viewer window.
▯	One Page	Display the report in a one-page view in the Report Viewer window.
◫	Two Pages	Display the report in a side-by-side two-page view in the Report Viewer window.
⏮	First Page	Jump to the first page.
◀	Previous Page	Go to the previous page.
▶	Next Page	Go to the next page.
⏭	Last Page	Jump to the last page.
🔍	Zoom In	Magnify the report preview size.
90%	Size	Specify the exact report preview size.
🔍	Zoom Out	Decrease the report review size.
✖	Cancel Rendering	Cancel rendering the report.

Using Report Filters

If a report always contains a certain range of data and is exported to a certain format, you do not have to define the filter criteria and output settings every time you generate the report. You can save the search conditions and output settings as a filter and open it to load the same set of configurations when you run the report later.

A filter is a personalized set of search conditions and settings, which means that the filters you created can only be accessed and managed by you and the administrator, and no one else.

Creating a New Filter

1 In the Filter window, click New Filter on the toolbar. All the search conditions are reset to the default values.

2 Change the filter criteria.

3 On the toolbar, click Save As.

4 In the Save As dialog box, enter a unique filter name, and optionally, a brief description; then click OK.

Note The filter name can be a label term such as `${SALES_ORDERS}` so that the filter name will be translated.

5 The filter is created. If you make further changes to the search conditions, click Save on the toolbar to save the changes.

Loading an Existing Filter

1 On the toolbar, click Open and then select an existing filter from the list. (The default filter is listed in bold.)

2 If the list is too long, click More to choose the filter from a browse window.

When you select More, you can then use the Filter window to select the default filter to display in the Open menu.

3 After you select an existing filter, its search conditions and settings are loaded in the Filter window.

4 If you want to save any changes to the loaded filter, click Save on the toolbar.

Maintaining Your Own Filters

Filters are user-specific. Use Personal User Filter Maintenance (36.4.21.14) to maintain your own filters.

System. Specify whether or not the filter is system-defined.

User. View or enter the user for whom to define the filter. When the filter is system-defined, this field is disabled.

Filter. Enter a filter name.

Description. Enter the description of the filter.

Default. Indicates whether this is the default filter in Report Viewer.

Param Name. Enter a parameter name in the filter criteria. If the parameter name already exists, you can either create a new parameter with the same name or update the existing one.

Chapter 6

Administering Reports

Setting Up Access Security for Reporting

Access must be controlled to reporting programs as well as report resources (associated with menu items) so that only authorized persons can gain access to and manipulate reporting data.

You use the system's role-based access security mechanism to control access to reporting resources in the same way as you do with other menu-level programs in QAD Enterprise Applications.

Setting Up Report Resource Menu Item and Security

Users open reports through report menu items that they have been given access to based on their role membership. Use Menu System Maintenance to create a menu item to provide access to a report resource.

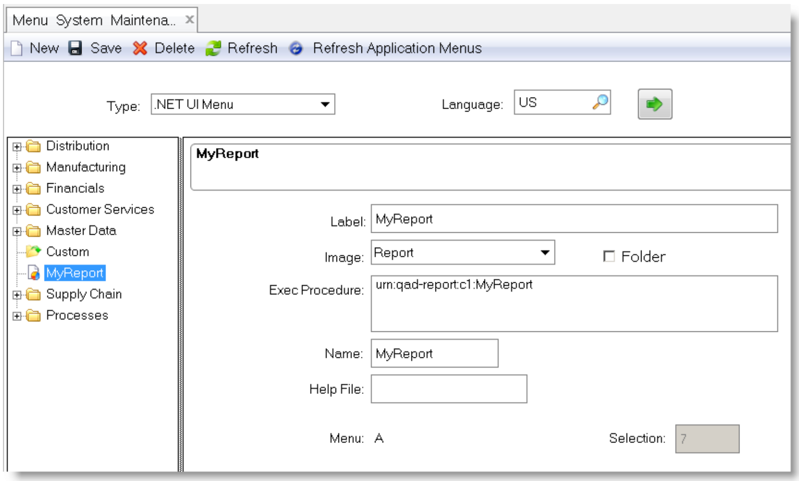
1 Go to Menu System Maintenance (36.4.4).

2 Specify .NET UI Menu in the Type field and select US in the Language field; then press Enter.

3 In the menu structure frame, right-click on a menu item under which you want to create the report menu item and then choose New from the shortcut menu.

4 Enter the detailed information for the report menu item.

Fig. 6.1
Menu System Maintenance



Type: .NET UI Menu

Language: US

Label: MyReport

Image: Report

□ Folder

Exec Procedure: urn:qad-report:c1:MyReport

Name: MyReport

Help File:

Menu: A

Selection: 7

Label. Enter a name for the report menu item. It does not have to be the same as the report code.

Image. Select Report from the list.

Exec Procedure. Enter a component-based activity specified in the form of a uniform resource name (URN):

```
urn:qad-report:c1:ReportCode
```

Where *ReportCode* is the report code of the report resource you are creating a menu item for.

Name. Optionally, enter a menu name.

5 Click Save on the toolbar to save your changes.

6 Click Refresh Application Menus on the toolbar. When refresh is complete, the new menu displays in the menu tree.

7 Set up security for the report menu item. For details on how to set up security, see *User Guide: QAD Security and Controls*.

Setting Up the rptAdmin and rptDsgn Roles

You must create two roles—rptAdmin and rptDsgn—in Role Create for the report administrator and report designer/developer respectively, and then assign them to the particular user IDs you would like to perform the associated activities. These roles add another layer of security that controls access to some activities within the programs. Since the activity-level controls are hard-coded in the reporting programs, you will not be able to perform certain activities within these programs if you create roles with other names for the report administrator and report designer/developer.

Note Make sure you use the correct capitalization for the roles.

You must grant the rptAdmin and rptDsgn roles access to the following programs based on this table:

Table 6.1
Program-Role Access Control Matrix

Reporting Program	**rptAdmin**	**rptDsgn**
Report Designer	Allow	Allow
Template Designer	Allow	Allow
Report Resource Import	Allow	Allow
Report Resource Export	Allow	Allow
Scheduled Report Maintenance	Allow	
Report Resource Maintenance	Allow	Allow
Report Parameter Maintenance	Allow	Allow
Personal User Filter Maintenance	Allow	Allow
Admin User Filter Maintenance	Allow	
Report Settings Restore	Allow	

For details about common access security features, see *User Guide: QAD Security and Controls.*

Scheduling Reports

You can automate the process of generating routine reports by scheduling them to automatically run at specified times or intervals and have the reports sent to a specified destination, such as a printer or the document service on the report server.

To schedule reports to run at a specified time or interval, on the report server, you create a Windows scheduled task for a batch and group the reports in the batch.

The Windows Task Scheduler should be configured to launch the Report Batch Processor, which is a non-GUI instance of the QAD .NET UI launched from the command line, for a specific batch as scheduled and runs all the scheduled reports grouped in the batch. If already set up, the report outputs are sent to the QAD .NET UI document service and/or server-side printer as configured.

Multiple report servers can be set up for increased throughput and failover. The different servers can be configured to process different batches, or can even jointly process reports in the same batch. The Report Batch Processor coordinates the processing of scheduled reports with different priorities in the correct sequence across multiple report servers.

Scheduled reports have the following additional features compared to reports run in Report Viewer:

- The output file (PDF/Excel) of a scheduled report can be uploaded to the document service so that the user can view it in the QAD .NET UI later.
- E-mail notifications, including SMTP mails and inbox messages embedded in the QAD .NET UI. You should specify e-mail addresses and the inbox user IDs when creating scheduled reports. When a report link is included in the e-mail notification to a scheduled report, the link is a direct link to the report file on the server. This direct link launches the report in a browser. However, for QAD .NET UI inbox messages, the link is a QAD Shell URL (http://qadsh) link that launches the report in the QAD .NET UI.
- Server-side printing. You can specify the printer that the report server uses to print the output file.
- Scheduled reports are maintained by the administrator from the maintenance program. The administrator controls the running sequence of scheduled reports by modifying their priorities.

Set Up a Scheduled Batch

1 Create a batch in Batch ID Maintenance (36.14.1).

2 On the report server, create a scheduled task for the batch through Windows Task Scheduler.

  a Create a parameter file to contain command line parameters with fixed values. Use the following `params.pf` file as an example:

```
-silent
-config-name:test
-user:mfg
-password:(blank)
-workspace:Domain1.1000
-report-batch:batch1
-enable:qad.plugin.services
-enable:qad.plugin.reports
-enable:qad.plugin.reportserver
-enable:qad.plugin.financials
-report-mode:batch
```

Note You must specify `-enable:qad.plugin.financials` if you are using QAD Enterprise Applications — Enterprise Edition. You must not specify `-enable:qad.plugin.financials` if you are using QAD Enterprise Applications — Standard Edition.

You need to set your own desired values for these parameters:

- -config-name: The name of the configuration that the report server should log on to. This is the same as the value chosen by an end user from the drop-down list in the login screen of the QAD .NET UI application when run in GUI mode.

- -user: User ID for logging on to the QAD .NET UI system
- -password: Password for logging on to the QAD .NET UI system
- -workspace: The workspace key of the desired workspace to run scheduled reports in. This is important, since any batch queue is specified by a unique combination of domain and batch ID, and the domain that the report server will use is the domain associated with the specified workspace key.
- -report-batch: The batch ID that will be used in conjunction with the domain associated with the specified workspace key to determine the batch of reports to run.

b In the launching script of the report server process, use the parameter file in place of the parameters:

```
QAD.Client.exe -param.url:file:///c:/params.pf
```

If the parameter file is referenced by a URL, you can choose to place the file on the local machine or a report server.

```
QAD.Client.exe -param.url:http://localhost/rpt/params.pf
```

Note The language that translations will be done is in the language of the user who scheduled the report, regardless of the server user's language.

Setting Up E-Mail Notifications

To set up e-mail notifications, add the following entries to `client-session.xml` on the home server:

```
<Configuration>
...
<!-- SMTP server host name -->
<Smtp.Host>SMTPHostname</Smtp.Host>
<!-- SMTP port name -->
<Smtp.Port>SMTPPortNumber</Smtp.Port>
<!-- SMTP from email address -->
<Smtp.From>E-Mail</Smtp.From>
<!-- SMTP username -->
<Smtp.Username>SMTPUsername</Smtp.Username>
<!-- SMTP password -->
<Smtp.Password>SMTPassword</Smtp.Password>
<!-- SMTP use SSL -->
<Smtp.UseSSL>false</Smtp.UseSSL>
</Configuration>
```

Use the following settings as a reference:

```
<Smtp.Host>smtp.qad.com</Smtp.Host>
<Smtp.Port>25</Smtp.Port>
<Smtp.From>Report Server <joedoe@qad.com></Smtp.From>
<Smtp.Username>admin</Smtp.Username>
<Smtp.Password>123</Smtp.Password>
<Smtp.UseSSL>false</Smtp.UseSSL>
```

Setting Up a Printer

Note Starting with the QAD .NET UI 2.9.2 (Enterprise Applications 2010.1 EE) release, printer setup for scheduled reports has changed. The Printer Setup Maintenance program is no longer used for the Reporting Framework. Instead, use the following steps to set up a printer for scheduled reports. If you are upgrading from an older version, any scheduled reports that refer to the previous printer types will still execute properly. However, the following steps must still be done to allow printers to be defined for new scheduled reports.

1 Set up a physical printer on the report server. From the Windows Start menu, select Control Panel|Printers and Faxes|Add a Printer to add a printer.

2 In the client session configuration file (`client-session.xml`), located in the *TomcatInstallDir*/webapps/qadhome/configurations/*SysEnvName*/ directory, set up printers available for scheduled reports as follows:

```
<ReportServer>
<Printer>
<UNCPath>\\machineA\printerA</UNCPath>
<Description>Description of printer (optional)</Description>
</Printer>
<Printer>
<UNCPath>\\machineB\printerB</UNCPath>
<Description></Description>
</Printer>
</ReportServer>
```

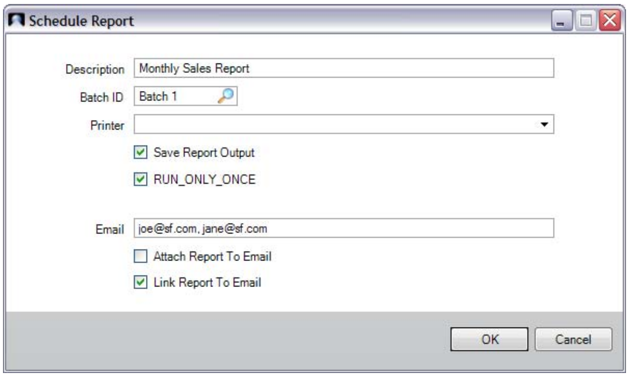
Creating a Scheduled Report

1 Open a report by double-clicking the report menu item in the Applications menu tree or right-clicking it and choosing Open from the shortcut menu. The Report Filter window is displayed in the application area.

2 On the toolbar, click Schedule and then click New.

3 In the Schedule Report dialog box, enter the required information and click OK.

Fig. 6.2
Schedule Report



Description: Monthly Sales Report

Batch ID: Batch 1

Printer:

☑ Save Report Output

☑ RUN_ONLY_ONCE

Email: joe@sf.com, jane@sf.com

☐ Attach Report To Email

☑ Link Report To Email

Description. Enter a short, meaningful description of the scheduled report.

Batch ID. Specify the batch ID for the scheduled report. The batch ID is created by the administrator in Batch ID Maintenance (36.14.1) and determines when and how often the report will be run on the report server.

Printer. If you want to have the scheduled report printed, specify a printer to send the report to. The printers available from the pull-down list are set up by the administrator (see “Setting Up a Printer” on page 82).

Save Report Output. Select this option if you want the report server to send the scheduled report output file to the document service.

Run Once. Select this option if you want to mark the scheduled report as non-permanent. A non-permanent scheduled report will run only once with the next batch run, regardless of how many times the associated batch is scheduled to run. A permanent scheduled report will run every time the batch is run.

E-Mail. Enter e-mail addresses or Inbox user IDs you want to have scheduled report notifications sent to. Separate multiple entries with commas.

Attach Report to Email. Select this option to attach a copy of the report in the e-mail notification.

Link Report To Email. Select this option to include a link to the report in the e-mail notification.

4 A confirmation message appears. The report is successfully scheduled.

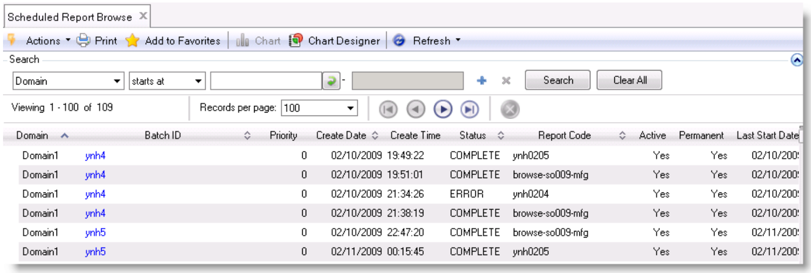
Viewing Scheduled Reports

To view details of already scheduled reports, do one of the following:

- In Report Viewer, click Schedule on the toolbar and then click View Schedule.
- Type Scheduled Report Browse in the menu search field and press Enter.

The browse displays detailed information of scheduled reports. You can modify a scheduled report by right-clicking it and then clicking Scheduled Report Maintenance.

Fig. 6.3
Schedule Report Browse



Viewing 1 - 100 of 109

Records per page: 100

Domain	Batch ID	Priority	Create Date	Create Time	Status	Report Code	Active	Permanent	Last Start Date
Domain1	ynh4	0	02/10/2009	19:49:22	COMPLETE	ynh0205	Yes	Yes	02/10/200
Domain1	ynh4	0	02/10/2009	19:51:01	COMPLETE	browse-so009-mfg	Yes	Yes	02/10/200
Domain1	ynh4	0	02/10/2009	21:34:26	ERROR	ynh0204	Yes	Yes	02/10/200
Domain1	ynh4	0	02/10/2009	21:38:19	COMPLETE	browse-so009-mfg	Yes	Yes	02/10/200
Domain1	ynh5	0	02/10/2009	22:47:20	COMPLETE	browse-so009-mfg	Yes	Yes	02/11/200
Domain1	ynh5	0	02/11/2009	00:15:45	COMPLETE	ynh0205	Yes	Yes	02/11/200

Key Field Descriptions

Create Date. The date on which the scheduled report was created.

Create Time. The time when the scheduled report was created.

Status. Specifies the status of the scheduled report.

- New: The scheduled report is newly created and has never run.
- Waiting: The scheduled report is ready to run in the next batch run.

- Running: The scheduled report is running.
- Completed: The scheduled report has run with no errors.
- Error: The scheduled report has run with errors.

Report Code. The report resource code associated with the current scheduled report.

Active. Indicates whether the scheduled report is currently active. Inactive reports will not be run.

Permanent. Indicates whether the scheduled report will run with every batch run or just once.

- Yes: The scheduled report will always run with every batch run.
- No: The report will only run once.

Last Start Date. The date on which the last run started.

Last Start Time. The time when the last run started.

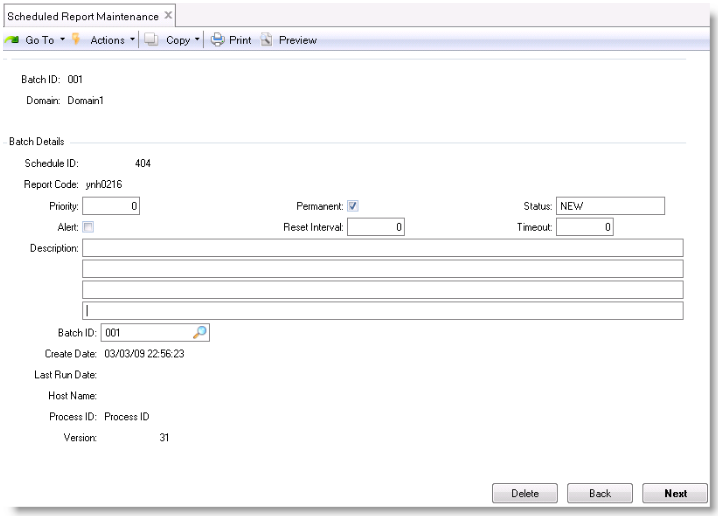
Last End Date. The date on which the last run ended.

Last End Time. The time when the last run ended.

Maintaining Scheduled Reports

To maintain a scheduled report, in Scheduled Report Browse, right-click the batch ID and then click Scheduled Report Maintenance.

Fig. 6.4
Schedule Report Maintenance



Batch ID: 001
Domain: Domain1

Batch Details

Schedule ID: 404
Report Code: ynh0216
Priority: 0
Permanent: ☑
Status: NEW
Alert: □
Reset Interval: 0
Timeout: 0
Description:
Batch ID: 001
Create Date: 03/03/09 22:56:23
Last Run Date:
Host Name:
Process ID: Process ID
Version: 31

Key Field Descriptions

Schedule ID. This is a system-assigned number that uniquely identifies a scheduled report within the current batch.

Report Code. The report resource code of the current scheduled report.

Priority. Specify an integer number that indicates the priority of the scheduled report. Scheduled reports with greater numbers have higher priorities and will run prior to lower-priority reports in the same batch.

Permanent. Specify whether the scheduled report will run with every batch run or just once.

- Yes: The scheduled report will always run with every batch run.
- No: The report will only run once.

Status. Displays the status of the current scheduled report.

- New: This is a newly created scheduled report.
- Waiting: The scheduled report is ready to run in the next batch run.
- Running: The scheduled report is currently running.
- Completed: The scheduled report has run with no errors.
- Error: The scheduled report has run with errors.

You can type in another status to manually change the current status of the scheduled report. This is useful on such occasions as when you have killed the batch run process on the report server but the status of scheduled report still shows Running.

Alert. This field is currently not implemented.

Reset Interval. This field is currently not implemented.

Timeout. This field is currently not implemented.

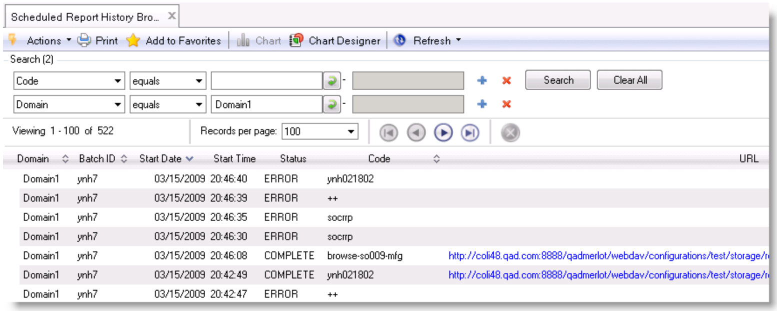
Batch ID. Displays the batch ID the current scheduled report pertains to. You can specify another batch ID if you want to move the scheduled report to that batch.

Viewing Scheduled Report History

To view the scheduled report history, do one of the following:

- In Report Viewer, click Schedule on the toolbar and then click View History.
- Type Scheduled Report History Browse in the menu search field and press Enter.

Fig. 6.5
Schedule Report History



Code equals

Domain equals Domain1

Viewing 1 - 100 of 522

Records per page: 100

Domain	Batch ID	Start Date	Start Time	Status	Code	URL
Domain1	ynh7	03/15/2009	20:46:40	ERROR	ynh021802	
Domain1	ynh7	03/15/2009	20:46:39	ERROR	++	
Domain1	ynh7	03/15/2009	20:46:35	ERROR	socrrp	
Domain1	ynh7	03/15/2009	20:46:30	ERROR	socrrp	
Domain1	ynh7	03/15/2009	20:46:08	COMPLETE	browse-so009-mfg	http://coli48.qad.com:8888/qadmerlot/webdav/configurations/test/storage/r
Domain1	ynh7	03/15/2009	20:42:49	COMPLETE	ynh021802	http://coli48.qad.com:8888/qadmerlot/webdav/configurations/test/storage/r
Domain1	ynh7	03/15/2009	20:42:47	ERROR	++	

Scheduled Report API

In addition to scheduling reports from the user interface, an API is available that can schedule reports programmatically from Progress 4GL code. Contact QAD Support to get a sample bundle (ScheduleReportSample.zip) that gives instructions and the necessary supporting files to access this API.

Report Data Source Provider Settings

The report data source provider settings are specified in the server-side client session configuration file (`client-session.xml`), which for the default environment is located in the file is located in *`TomcatInstallDir`*`/webapps/qadhome/configurations /default/client-session.xml`.

In `client-session.xml`, the following specifies the report data source provider:

```
<ReportDataSourceProviders>
    <Provider>
        <Name>GenericProxy</Name>
        <Assembly>QAD.Plugin.Reports.ReportFramework</Assembly>
      <Class>QAD.Plugin.Reports.ReportFramework.GenericProxyDatasourceProvider</Class>
    </Provider>
    <Provider>
        <Name>Browse</Name>
        <Assembly>QAD.Browse</Assembly>
        <Class>QAD.Browse.BrowseDatasourceProvider</Class>
    </Provider>
    <Provider>
        <Name>FinancialAPI</Name>
        <Assembly>BaseLibrary</Assembly>
        <Class>BaseLibrary.Reports.BLFDatasourceProvider</Class>
    </Provider>
</ReportDataSourceProviders>
```

Starting with the 2009.1 release of QAD Enterprise Applications, the settings for the `BLFDatasourceProvider` changed.

Starting with the 2009.1 release, the settings for the FinancialsAPI data source type are:

```
<Provider>
    <Name>FinancialAPI</Name>
    <Assembly>BaseLibrary</Assembly>
    <Class>BaseLibrary.Reports.BLFDatasourceProvider</Class>
</Provider>
```

But previous to the 2009.1 release, the settings were:

```
<Provider>
    <Name>FinancialAPI</Name>
    <Assembly>BaseAdapters.Reporting</Assembly>
    <Class>BaseAdapters.Reporting.DatasourceProvider.BLFDatasourceProvider</Class>
</Provider>
```

If you are upgrading from the 2009 release or prior, you can change the settings are needed for 2009.1 and above by editing the `client-session.xml` file accordingly.

Restoring Report Settings

After you make changes to report settings in Report Parameter Maintenance (36.4.21.3), you can change them back to the default settings. To do this, go to Report Settings Restore (36.4.21.23) and set Restore to Yes.

Appendix A

Implementing a Progress Data Source Program

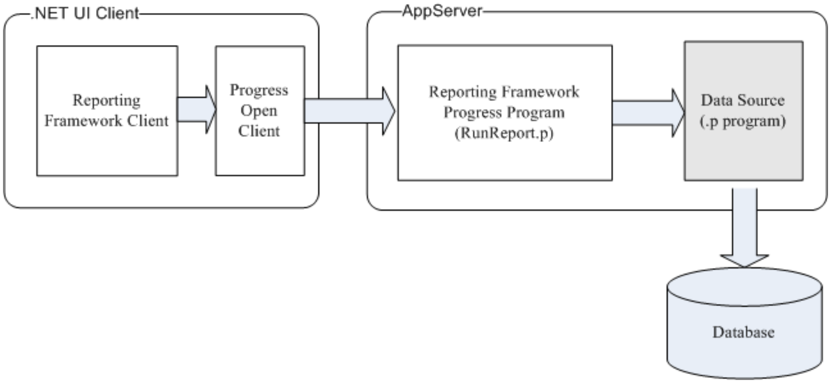
Overview

As discussed in “Choosing a Report Data Source” on page 11, there are three types of data sources that a report can use: Browse, FinancialAPI, and Proxy. The implementation of the Proxy data source is a Progress program that resides on the .NET Progress AppServer. This section describes how to create such a program.

This program serves as the data source provider that retrieves data from the database, constructs datasets, and passes them to Report Designer or Report Viewer through Progress Open Client for .NET to generate reports.

The following diagram illustrates the context in which a Progress data source program is run.

Fig. A.1
Data Source Program Context


.NET UI Client
Reporting Framework Client
Progress Open Client
AppServer
Reporting Framework Progress Program (RunReport.p)
Data Source (.p program)
Database

Implementing the data source program entails the following two steps:

1. Develop the data source program code.
2. Deploy the data source program on the QAD .NET UI Application Server.

Developing the Data Source Program Code

A data source program comprises four blocks of code:

- A data set definition (consisting of one or more temp-tables) that defines the data set structure for the report
- Statements to empty the temp-table(s) in the data set
- Metadata definition that defines attributes for the fields in the data set
- Data retrieval logic that populates the data set

In order for the program to be invoked properly by RunReport.p, it must have a specific structure. The following sample code illustrates this structure, showing where the four blocks of code should be inserted:

```
{mfdeclre.i}
{gplabel.i}

{com/qad/shell/report/dsReportRequest.i}
{com/qad/shell/report/ReportConstants.i}
```

```
/* Report data set definition block */

/* TODO Insert your data set code here */

/* Main Block */
define input parameter runReport as logical.
define input parameter reportHandle as handle.
define input parameter dataset for dsReportRequest.
define output parameter dataset-handle phReportResults.

{com/qad/shell/report/reporting.i}

define variable bufferName as character no-undo.

/* TODO empty the temp-table(s) */

for first ttReportRequest no-lock:

   run FillMetaData.

   if runReport then do:
      run RunReport
         (output dataset-handle phReportResults).
   end.

end.

/* Metadata definition block */

procedure FillMetaData:

   /* TODO Insert your metadata code here */

end procedure.

/* Data retrieval logic block */
procedure RunReport:

   define output parameter dataset-handle phReportResults.

   /* TODO Insert your data retrieval code here */

   phReportResults = dataset dsReportResults:handle.

end procedure.
```

The code contains several comments starting with `TODO`. These comments indicate where the code blocks should be inserted. The following sections will discuss how to write the code blocks to be inserted into the above structure.

Data Set Definition Block

The following code block from a sample program illustrates how to define a data set and its temp tables. A dataset may contain multiple temp-tables, which often have relations between them. In the following example, we create a data set with two simple temp-tables that have a master-detail relationship:

```
/* Temp-table definition block */
define temp-table ttSalesHeader
   field so_nbr like so_mstr.so_nbr
   field so_cust like so_mstr.so_cust
   field so_ord_date like so_mstr.so_ord_date
   field sales_order_slspsn1 like so_mstr.so_slspsn[1]
   field sales_order_slspsn2 like so_mstr.so_slspsn[2]
   field sales_order_slspsn3 like so_mstr.so_slspsn[3]
   field sales_order_slspsn4 like so_mstr.so_slspsn[4]
   index SalesHeaderIdx is primary so_nbr
.
define temp-table ttSoLine
   field sales_order_number like so_mstr.so_nbr
   field sales_detail_line like sod_det.sod_line
   field sales_detail_item like sod_det.sod_part
   field sales_detail_unit_measure like sod_det.sod_um
   field sales_detail_due_date like sod_det.sod_due_date
   index SoLineIdx is primary sales_order_number sales_detail_line.
.
define dataset dsReportResults for ttSalesHeader, ttSoLine
   data-relation drLine for ttSalesHeader, ttSoLine
   relation-fields (so_nbr, sales_order_number)
```

Note If you want to use some fields in the temp-table as search fields, you must use the same field names in the temp-tables as those in the database.

Note Indexes specified for the temp tables can be used to define unique constraints and the default sort order of the records.

Note There must be a dataset named dsReportResults defined for temp-tables for the program to work, even if they have no relations.

Note The use of Progress array fields (`extent` fields) is not supported in report data set temp tables. Instead, you can define several individual fields to hold the elements of the array. In the above example, this was done to handle the array `so_mstr.so_slspsn[]`.

Empty Temp-Table Block

Each temp-table in the report data set should be emptied before the main program logic is executed. This is necessary since temp-table contents could still be cached on the Application Server from previous client requests.

The following example illustrates this for the two temp-tables in our example data set:

```
/* Empty temp-table block */
empty temp-table ttSalesHeader no-error.
empty temp-table ttSoLine      no-error.
```

Metadata Block

You define metadata to specify which fields and tables the user can use to design the report. Every table in the data set needs a buffer header metadata section, as well as metadata describing its fields.

To facilitate the coding of metadata, there is a helper program (ReportHelper.p), which is accessible in your data source program through the persistent procedure handle `reportHandle`. This helper program contains several functions, described below, which populate the metadata data set that gets passed back to the client.

Defining Buffer Name and Creating BufferHeader

The CreateBufferHeader procedure creates the metadata that describes a table, and must be run once for each temp-table in your report data set.

The input parameters to this procedure are listed below; be sure to use the exact temp-table for the table name parameter.

Table A.1
CreateBufferHeader Parameters

Seq.	Name	Input/ Output	Data Type	Description
1	tableName	Input	Character	Temp-table name
2	tableLabel	Input	Character	Label displayed in Report Designer

Creating Fields for Each Temp-Table

Either of three predefined procedures can be used to create field metadata. The procedures are similar and have the same effect of creating a metadata record for one field. A description of each is given below.

- CreateField — this variant allows the programmer to explicitly pass in the values of each metadata parameter.
- CreateFieldForDBField — this variant can be used if the field is similar to one in the business database, in which case some of the metadata parameters will be determined by the system based on the database field. For example, the field label, data type, field format, and lookup name will be driven by the database field.
- CreateFieldLikeDBField — this variant is almost identical to CreateFieldForDBField except that it allows the field name of the report field to be different from that of the database field.

The following sections list the input parameters for each of these three procedures.

CreateField

Table A.2
CreateField Parameters

Name	Input/Output	Data Type	Description
bufferName	Input	Character	Temp-table name
fieldName	Input	Character	Report field name
fieldLabel	Input	Character	User-facing label for this field, as it will appear on the prompt page of the report viewer
dataType	Input	Character	Progress datatype of the field
fieldFormat	Input	Character	Progress format for the field. In the case of logical-typed fields, the field format string is used to specify the user-facing labels for the true and false values the field can hold; the syntax is: `<trueLabel>/<falseLabel>`. For example, `"Debit/Credit"`. Note: in a multi-language system, these values should not be hard coded but instead dynamically translated using the getLabel function. The format strings will be used in search condition values on the prompt page, as well as data values in the report output.

Name	Input/Output	Data Type	Description
lookupName	Input	Character	Program name of the lookup program (if any) that will be invoked from the lookup icon for this field on the prompt page of the report viewer; For example, `"gplu340.p"`. For more information on defining lookups, see “Defining Search Field Lookups” on page 99.
isSearchField	Input	Logical	Whether this field should be a search field on the prompt page
isReadOnlySearch	Input	Logical	Not used: use isEditable instead.
isVisible	Input	Logical	Whether this field is visible in Report Designer
isSingleEntry	Input	Logical	If set to True, then only a single search condition will be allowed for this field.
isOperatorChangeable	Input	Logical	Whether the operator can be changed on the prompt page
isRequiredCondition	Input	Logical	Whether the field is mandatory on the prompt page
isEditable	Input	Logical	Whether the field can be edited on the prompt page
defaultValue	Input	Character	Default value of the first search field
defaultOperator	Input	Character	Default operator of the first search field
defaultValueType	Input	Character	Default value type of the first search field
defaultValue2	Input	Character	Default value of the second search field
defaultValueType2	Input	Character	Default value type of the second search field

CreateFieldForDBField

Table A.3
CreateFieldForDBField Parameters

Name	Input/Output	Data Type	Description
bufferName	Input	Character	Temp-table name
tableName	Input	Character	QAD ERP database table name
fieldName	Input	Character	QAD ERP database field name
isSearchField	Input	Logical	Whether this field should be a search field on the prompt page
isReadOnlySearch	Input	Logical	Whether this field is read-only on the prompt page
isVisible	Input	Logical	Whether this field is visible in Report Designer
isSingleEntry	Input	Logical	Always set this to False
isOperatorChangeable	Input	Logical	Whether the operator can be changed on the prompt page
isRequiredCondition	Input	Logical	Whether the field is mandatory on the prompt page
isEditable	Input	Logical	Whether the field can be edited on the prompt page
defaultValue	Input	Character	Default value of the first search field
defaultOperator	Input	Character	Default operator of the first search field
defaultValueType	Input	Character	Default value type of the first search field
defaultValue2	Input	Character	Default value of the second search field
defaultValueType2	Input	Character	Default value type of the second search field

CreateFieldLikeDBField

Table A.4
CreateFieldLikeDBField Parameters

Name	Input/Output	Data Type	Description
bufferName	Input	Character	Temp-table name
fName	Input	Character	Field name in the temp-table (report field name)
tableName	Input	Character	QAD ERP database table name
fieldName	Input	Character	QAD ERP database field name
isSearchField	Input	Logical	Whether this field should be a search field on the prompt page
isReadOnlySearch	Input	Logical	Whether this field is a read-only on the prompt page
isVisible	Input	Logical	Whether this field is visible in Report Designer
isSingleEntry	Input	Logical	Always set this to False
isOperatorChangeable	Input	Logical	Whether the operator can be changed on the prompt page
isRequiredCondition	Input	Logical	Whether the field is mandatory on the prompt page
isEditable	Input	Logical	Whether the field can be edited on the prompt page
defaultValue	Input	Character	Default value of the first search field
defaultOperator	Input	Character	Default operator of the first search field
defaultValueType	Input	Character	Default value type of the first search field
defaultValue2	Input	Character	Default value of the second search field
defaultValueType2	Input	Character	Default value type of the second search field

Example The following is a complete metadata definition code block illustrating a simple case consisting of a single table and three fields.

```
procedure FillMetaData:

    bufferName = "ttSalesHeader".

    run CreateBufferHeader in reportHandle
       (bufferName, "Sales Orders").


    /* The so_nbr field is set as a search field, so the seventh parameter is True. */

    run CreateField in reportHandle
       (bufferName,
        "so_nbr",
        getLabel("SALES_ORDER"),
        "character",
        "x(8)",
        "gplu239.p",
        true,
        false,
        true,
        false,
        true,
        false,
        true,
        "",
        {&ParameterOperator_Equals},
        {&ParameterValueType_Constant},
        "",
        {&ParameterValueType_Constant}).

    /* The following field will have some of its metadata attributes */
    /* driven from the so_cust database field. */
    run CreateFieldForDBField in reportHandle
       (bufferName,
       "so_mstr",
        "so_cust",
        true,
        false,
        true,
        false,
        true,
        false,
        true,
        "",
        {&ParameterOperator_Equals},
        {&ParameterValueType_Constant},
        "",
        {&ParameterValueType_Constant}).

    /* The following field will be called order_date in the report even though some */
    /* of its metadata attributes will be driven from the so_ord_date database field. */

    run CreateFieldLikeDBField in reportHandle
       (bufferName,
        "order_date",
        "so_mstr",
        "so_ord_date",
        true,
        false,
        true,
        false,
        true,
        false,
        true,
        "",
        {&ParameterOperator_Equals},
        {&ParameterValueType_Constant},
        "",
        {&ParameterValueType_Constant}).

end procedure.
```

Note The above code example is intentionally limited to a single table and three fields for the purpose of illustrating each of the metadata procedures. It does not define the entire metadata necessary to describe the master detail data set structure from the previous example. Please refer to the complete code example at the end of this appendix to see the entire metadata section.

Note The last example (`CreateFieldLikeDBField`) is just for the purpose of illustrating this procedure call; it is inconsistent with our temp-table definition in the previous section, whose field name would have to be changed to `order_date` in order to match the report field name in the procedure call.

Specifying a Discrete Set of Values for a Field

Fields of type `character` can optionally be defined such that their allowed values can be restricted to a discrete set. If a user creates a search condition on the prompt page for such a field, they will encounter a drop-down box containing the allowed values to select from. Furthermore, the system can display translated labels for these values that map to the actual underlying data values that are stored in the database. This is accomplished by setting the `valueList` metadata property for the field.

The `valueList` metadata attribute must be specified using the following comma-separated value format:

```
    <Label 1>,<DB value 1>,<Label 2>,<DB value 2>, ...
```

In the above format, `<Label 1>` specifies the user-facing label for the first allowed value; `<DB Value 1>` specifies the string used in the database for the first allowed value, and so on for the other allowed values.

Note To escape a comma character "`,`" in the display values for a value list, use two backslashes: "\\". For example: `"d\\,isp1,value1,disp2,value2,disp3,value3"`.

For example, consider a field called `status` that should be restricted to the set of values "New", "In Process", and "Completed", which are stored in the database as the values "1", "2", and "3", respectively. The `valueList` attribute can be specified by adding the following code statements after this field's `CreateField` procedure call:

```
define variable statusValueList as character no-undo.

statusValueList = "New,1,In Process,2,Completed,3".

run SetFieldMetaParameter in ReportHandle(
   bufferName,
   "status",
   "valueList",
   statusValueList).
```

The `SetFieldMetaParameter` procedure is used to set a field metadata attribute for a field that has already been created in the metadata. The first input parameter is the buffer name of the field, the second parameter is the field name, the third parameter is the name of the attribute to set (`"valueList"`, in this example), and the fourth parameter is the value to set the attribute to (the comma-separated list). Please note that in multi-language system environments, the strings "New", "In Process", and so on, should first be translated before setting the metadata attribute, as described in the next section.

Translating Metadata Content

When creating reports for systems with multiple languages, most of the translated labels are specified when designing the report layout using the Report Designer (see "Using Translated Labels" on page 45). However, the following attributes of field metadata can also contain strings that may need to be translated:

- The `fieldLabel` attribute
- The `valueList` attribute
- The `fieldFormat` attribute (in the case of logical-typed fields)

When the report client sends a request to the data source program, the language of the user is set in the `global_user_lang` Progress variable. The data source program should use this setting to determine the language in which to translate labels used in the metadata.

Beginning in the QAD Enterprise Applications 2010 release, a new utility function has been added to the ReportHelper.p program that facilitates looking up translated labels from the QAD system labels:

```
   FUNCTION getLabel returns character (
    input pTerm as character):
```

The `getLabel` function takes a label term as its input parameter, and returns the translated label for that term in the language specified in the `global_user_lang` variable. If no label is found for the input term, then the term itself is output.

In the example above illustrating how to use the `CreateField` procedure, this technique was used to translate the field label for the `so_nbr` field by using the function call `getLabel("SALES_ORDER")` that looks up the translated label for the term `"SALES_ORDER"`.

Specifying Financials Lookups in Field Metadata

When writing data source programs, you can specify Financials lookups in the metadata as opposed to standard browse lookups. To do this, specify the following for the `lookupName` in the metadata: <*lookup provider type*>:<*lookupID*>:<*lookup return field*>:<*lookup filter field*>.

<*lookup return field*> and <*lookup filter field*> are optional. In the example above, the two fields are the same as the calling field in the browse whose lookup button is pressed.

Here is an example for the Financials lookup:

```
BaseLibrary.Lookup.BLFLookupProvider:BJournalSAO.SelectJournal:tcJournal
Code:tJournal.JournalCode
```

Getting Report Code and Report Definition

Beginning in the QAD Enterprise Applications 2010.1 release, two new functions that can be called directly from proxy code to return the report code and report definition are available:

- GetReportCode — Returns the report code.
- GetReportDefinition — Returns the report definition. It only has a valid value when the report is run and does not have a value when the proxy code is accessed to get the metadata.

Displaying Hard-Coded Message Text

In the RunReport procedure of a report data source program, if you try to raise an error message using the following statement containing a hard-coded message, it does not stop the report from running and does not display the message:

```
{pxmsg.i &MSGTEXT=""Example Message"" &ERRORLEVEL=3}
```

Instead, you must use do the following:

```
DisplayTextMessage("this is a testing message",3)
```

Note that using MSGNUM for messages that are not hard-coded does work as expected:

```
{pxmsg.i &MSGNUM=28 &ERRORLEVEL=3}
```

Specifying Search Operator List

The `SetFieldMetaParameter` procedure can specify the search operator list. The first input parameter is the buffer name of the field, the second parameter is the field name, the third parameter is the name of the attribute to set (in this case, `searchOperatorList`), and the fourth parameter is the value to set the attribute to (the comma-separated list). The following example includes all the available search operators:

```
run SetFieldMetaParameter in ReportHandle(
   bufferName,
   "status",
   "searchOperatorList",
   "Equals,NotEquals,Contains,Range,GreaterThanEquals,GreaterThan,LessThan,IsNull,
   IsNotNull").
```

In this example, only the search operators for “equals” and “contains” are included:

```
run SetFieldMetaParameter in ReportHandle(
   bufferName,
   "status",
   "searchOperatorList",
   "Equals,Contains").
```

Defining Search Field Lookups

Fields appearing in the report search panel can optionally have a lookup specified. This is done by specifying the LookupName attribute in the field metadata (see “CreateField Parameters” on page 93). Many possibilities that are supported are described below.

Non-Component Based (MFG/PRO) Browses

Any browse created by Browse Maintenance can be used as a search field lookup, where the LookupName attribute in the field metadata (see “CreateField Parameters” on page 93) is the program name of the lookup program (if any) that is invoked from the lookup icon; for example, `gplu340.p`.

Specifying Financials Lookups

When writing data source programs, you can specify Financials lookups in the metadata as opposed to standard browse lookups. To do this, specify the following for the `lookupName` in the metadata: <*lookup provider type*>:<*lookupID*>:<*lookup return field*>:<*lookup filter field*>.

<*lookup return field*> and <*lookup filter field*> are optional. In the example above, the two fields are the same as the calling field in the browse whose lookup button is pressed.

Here is an example for the Financials lookup:

```
BaseLibrary.Lookup.BLFLookupProvider:BJournalSAO.SelectJournal:tcJournal
Code:tJournal.JournalCode
```

Passing Parameters to Lookups

The `<lookupID>` can also have parameters passed to it to dynamically change the behavior of the lookup. For example, a lookup of items may be defined to have a product line input parameter that would filter the item list according to the product line value passed in. The values that can be passed to lookups can either be hard-coded or obtained from values that the user has entered into any other search condition.

Parameters are currently only supported by the `QAD.Browse.MfgProLookupProvider` type (browses created by Browse Maintenance).

The syntax for passing parameters to lookups is as follows:

```
<lookupID>(paramName1,paramValue1,paramName2,paramValue2,...)
```

Literal `paramValue1` values are supported, as well as values that are dynamically determined by a search condition value in the search panel. The latter case is invoked by prefixing the `paramValue1` value with the `@` character. For example, a browse may have a search field pt_mstr.pt_prod_line, and another search field pt_mstr.pt_part, and the lookup for pt_part might be restricted by what the user entered in the value to restrict the product line by, with the pt_part search lookup (for example, *example*.p) expecting c-brparm1 parameter to contain the product line to filter by. This is accomplished by using the following lookupID string:

```
example.p(c-brparm1,@pt_mstr.pt_prod_line)
```

Specifying Conditional Lookups

In some cases, you may want to have a search field lookup invoke a different browse depending on the user-selected value in a different search condition.

For example, a report may have a vendor source field that is a value list with two items: PO (Purchase Order) and DO (Distribution Order). If the user selects PO, the lookup on the supplier field should be a browse against suppliers. If the user selects DO, the lookup on the supplier field should be a browse against sites.

The syntax for specifying conditional lookups is as follows:

```
<FieldName>|<Value1>,<LoopkupTarget1>,<Value2>,<LoopkupTarget2>,...|
<DefaultLookupTarget>
```

In the above, the search condition value of the condition on the field specified by <FieldName> is evaluated according to the map in the second section, which maps search values to lookup targets (browses) to invoke. If no match is found for the actual value of <FieldName> in the search panel, the default lookup target is used. If the type of search value is a ValueList or boolean, then the underlying values in the list (not the translated labels) should be specified in the <Value1> values.

Example This example invokes lookup targets that are browses created by Browse Maintenance, with the choice of lookup depending on the value entered into the search condition for the so_nbr (string) field:

```
so_mstr.so_nbr|czs-001,QAD.Browse.MfgProLookupProvider:gplu396.p,
SO-002,QAD.Browse.MfgProLookupProvider:gplu242.p|
QAD.Browse.MfgProLookupProvider:gplu340.p
```

This example invokes lookup targets that are browses defined in the CBF tool in Enterprise Edition, with the choice of lookup depending on the value entered into the search condition for the tPosting.PostingApproveStatus (value list) field:

```
tPosting.PostingApproveStatus|
APPROVEDANDCORRECTED,BaseLibrary.Lookup.BLFLookupProvider:BLayerSAO.Sele
ctLayer:tcLayerCode:tLayer.LayerCode:tLayer.LayerCode,APPROVED AND NOT
PASSED,BaseLibrary.Lookup.BLFLookupProvider:BUserSAO.SelectUser:tcUsrLog
in:tUsr.UsrLogin:tUsr.UsrLogin|
BaseLibrary.Lookup.BLFLookupProvider:BJournalSAO.SelectJournal:tcJournal
Code:tJournal.JournalCode
```

Example This example invokes lookup targets that are browses defined in the CBF tool in Enterprise Edition, with the choice of lookup depending on the value entered into the search condition for the tPosting.PostingIsAutoReversal (bool) field:

```
tPosting.PostingIsAutoReversal|
true,BaseLibrary.Lookup.BLFLookupProvider:BLayerSAO.SelectLayer:tcLayerC
ode:tLayer.LayerCode:tLayer.LayerCode,false,BaseLibrary.Lookup.BLFLookup
```

```
Provider:BUserSAO.SelectUser:tcUsrLogin:tUsr.UsrLogin:tUsr.UsrLogin|
BaseLibrary.Lookup.BLFLookupProvider:BJournalSAO.SelectJournal:tcJournal
Code:tJournal.JournalCode
```

Example This example illustrates how to specify a conditional lookup with one of the lookup targets having dynamic parameters passed to it. In addition to showing the lookup name string, this example also shows the entire code segment that would be used in a Progress data source program metadata section. This defines a lookup on product line that depends on the value picked by the user in `ttitem.pt_status`, and can have dynamic parameters passed in:

```
define variable prodLineLookup as character no-undo.
prodLineLookup = "ttitem.pt_status|
AC,gplu396.p,BASE,gplu242.p,NWISS,wllu003.p(c-
brparm1,@ttitem.pt_part)|gplu343.p".

run SetFieldMetaParameter in ReportHandle(
bufferName,
"pt_prod_line",
"lookupName",
prodLineLookup).
```

Report Data Retrieval Logic Block

The data retrieval code block is used to populate temp-tables with data. A dynamic query must be used if there are any search fields, since the search conditions come from the client request and can vary at run time based on the user’s selections on the prompt page of the report viewer. The data retrieving logic should be coded into the designated place in the data source program.

Defining the Dynamic Query

Here is an example that illustrates how to structure the code for the dynamic query. Note that the search fields all reside in the so_mstr table.

```
define variable queryString as character no-undo.
define variable hSOQuery as handle.
define query SOQuery for so_mstr.

hSOQuery = query SOQuery:handle.

queryString = "for each so_mstr no-lock "
   + " where so_mstr.so_domain = " + QUOTER(global_domain).

run FillQueryStringVariable in reportHandle (input "ttSalesHeader", input "so_nbr",
input-output queryString).
run FillQueryStringVariable in reportHandle (input "ttSalesHeader", input "so_cust",
input-output queryString).
run FillQueryStringVariable in reportHandle (input "ttSalesHeader", input "order_date",
input-output queryString).

queryString = queryString + ":".

hSOQuery:query-prepare(queryString).
hSOQuery:query-open().
hSOQuery:get-next().
```

Note The FillQueryStringVariable function will get the search conditions sent from the client request (each consisting of the search field, operator, and value entered by the user) and then construct the corresponding where clause fragment and add it to the query string dynamically at run time.

The following lists the input parameters for the FillQueryStringVariable function:

Name	Input/ Output	Data Type	Description
bufferName	Input	Character	Temp-table name
fieldName	Input	Character	Field name in the temp-table
queryString	Input-Output	Character	Dynamic query string

Note For the function to work, the temp-table fields defined as search fields in the metadata definition should use exactly the same names as those in the database.

Note If other static filter conditions are desired in the query string, they can be added in place of the “where true” part of the query string in the above example. For example, if we wanted this report to filter its query to only include orders whose so_site value is “SITE1”, we could use the following statement to begin the query string:

```
queryString = "for each so_mstr no-lock where so_site = " + QUOTER("SITE1").
```

The `QUOTER()` function returns the input string wrapped in quotes, and is useful when specifying text in a dynamic query string that must appear quoted in the final query string.

If static sorting is desired, a “by” clause can be added at the end of the dynamic query string. For example, the following change to our above example will cause records to be sorted primarily by so_cust:

```
queryString = queryString + " by so_cust:".
```

There is another procedure called FillQueryString that will automatically add the filter conditions for all the fields in a given temp-table. This is often the most concise way to implement filter fields. The following code statement illustrates how to use this procedure:

```
run FillQueryString in reportHandle (input "ttSalesHeader", input-output queryString).
```

The following lists the input parameters for the FillQueryString function:

Name	Input/ Output	Data Type	Description
bufferName	Input	Character	Temp-table name
queryString	Input-Output	Character	Dynamic query string

Retrieving Data

Loop over the so_mstr records to create the ttSalesHeader temp-table and retrieve detailed information from each record. The inner loop over so_det records is used to retrieve all detail records for each master record.

```
repeat while not hSOQuery:query-off-end:
      create ttSalesHeader.
      assign
         ttSalesHeader.so_nbr                = so_mstr.so_nbr
         ttSalesHeader.so_cust               = so_mstr.so_cust
         ttSalesHeader.so_ord_date           = so_mstr.so_ord_date
         ttSalesHeader.sales_order_slspsn1  = so_mstr.so_slspsn[1]
         ttSalesHeader.sales_order_slspsn2  = so_mstr.so_slspsn[2]
         ttSalesHeader.sales_order_slspsn3  = so_mstr.so_slspsn[3].
```

```
          ttSalesHeader.sales_order_slspsn4  = so_mstr.so_slspsn[4].

       for each sod_det no-lock
where sod_det.sod_nbr = so_mstr.so_nbr:
create ttSoLine.
assign
   ttSoLine.sales_order_number        = sod_det.sod_nbr
   ttSoLine.sales_detail_line         = sod_det.sod_line
   ttSoLine.sales_detail_item         = sod_det.sod_part
           ttSoLine.sales_detail_unit_measure = sod_det.sod_um
   ttSoLine.sales_detail_due_date     = sod_det.sod_due_date.
      end.

      hSOQuery:get-next().
   end. /* Repeat query */
```

Deploying the Data Source Program

After the data source program is developed, deploy it onto the report server. Compile the code and put the `.r` or `.p` file in the specified directory so that Appserver can run this file.

Both the `.p` and `.r` files are needed when the server runs the code. Compile the `sosorp_Finished.p` file.

1. Connect to the QAD ERP production database (qaddb) and administration database (qadadm).
2. Compile the program, adding the following to the Propath:

- Report data source program directory:
  <desktop source code directory>`/com/qad/shell/report/reports`

  Where *<desktop source code directory>* usually is

  `/qad/web/server/docs/`*<ENVname>*`/ebdesktop2/`*<WEBAPPNAME>*/
- QAD ERP installation directory.

Here is a Propath sample:

```
propath = propath + "," +
"/qad/web/server/docs/93/ebdesktop2/dev93ui/com/qad/shell/report/reports".

propath = propath + "," +
"/qad/web/server/docs/93/ebdesktop2/dev93ui".

propath = propath + "," +
"/qad/web/server/docs/93/ebdesktop2/dev93ui/com/qad/shell/report".

propath = propath + "," +
"/qad/mfgpro/93/stage".
```

Compile `com/qad/shell/report/reports/TestReport.p` and save.

Complete Data Source Program Sample Code

```
{mfdeclre.i}
{gplabel.i}

{com/qad/shell/report/dsReportRequest.i}
{com/qad/shell/report/ReportConstants.i}

/* Report data set definition block */
define temp-table ttSalesHeader
   field so_nbr like so_mstr.so_nbr
   field so_cust like so_mstr.so_cust
   field so_ord_date like so_mstr.so_ord_date
   field sales_order_slspsn1 like so_mstr.so_slspsn[1]
   field sales_order_slspsn2 like so_mstr.so_slspsn[2]
   field sales_order_slspsn3 like so_mstr.so_slspsn[3]
   field sales_order_slspsn4 like so_mstr.so_slspsn[4]
   index SalesHeaderIdx is primary so_nbr
.
define temp-table ttSoLine
   field sales_order_number like so_mstr.so_nbr
   field sales_detail_line like sod_det.sod_line
   field sales_detail_item like sod_det.sod_part
   field sales_detail_qty_ord like sod_det.sod_qty_ord
   field sales_detail_unit_measure like sod_det.sod_um
   field sales_detail_due_date like sod_det.sod_due_date
   index SoLineIdx is primary sales_order_number sales_detail_line.
.

define dataset dsReportResults for ttSalesHeader, ttSoLine
   data-relation drLine for ttSalesHeader, ttSoLine
   relation-fields (so_nbr, sales_order_number)
.

/* Main Block */
define input parameter runReport as logical.
define input parameter reportHandle as handle.
define input parameter dataset for dsReportRequest.
define output parameter dataset-handle phReportResults.

{com/qad/shell/report/reporting.i}

define variable bufferName as character no-undo.

/* Empty temp-table block */
empty temp-table ttSalesHeader no-error.
empty temp-table ttSoLine no-error.

for first ttReportRequest no-lock:

   run FillMetaData.

   if runReport then do:
      run RunReport
         (output dataset-handle phReportResults).
   end.

end.

/* Metadata definition block */
procedure FillMetaData:

   bufferName = "ttSalesHeader".
   run CreateBufferHeader in reportHandle
      (bufferName, "Sales Orders").

   /* Create field for ttSalesHeader */
   run CreateFieldLikeDBField in reportHandle
      (bufferName,
      "so_nbr",
      "so_mstr",
      "so_nbr",
      true,
      false,
      true,
      false,
      true,
      false,
      true,
      "",
      {&ParameterOperator_Equals},
      {&ParameterValueType_Constant},
      "",
      {&ParameterValueType_Constant}).
```

```
run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "so_cust",
   "so_mstr",
   "so_cust",
   true,
   false,
   true,
   false,
   true,
   false,
   true,
   "",
   {&ParameterOperator_Equals},
   {&ParameterValueType_Constant},
   "",
   {&ParameterValueType_Constant}).

run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "so_ord_date",
   "so_mstr",
   "so_ord_date",
   true,
   false,
   true,
   false,
   true,
   false,
   true,
   "",
   {&ParameterOperator_Equals},
   {&ParameterValueType_Constant},
   "",
   {&ParameterValueType_Constant}).

run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "sales_order_slspsn1",
   "so_mstr",
   "so_slspsn",
   false,
   false,
   true,
   false,
   true,
   false,
   true,
   "",
   {&ParameterOperator_Equals},
   {&ParameterValueType_Constant},
   "",
   {&ParameterValueType_Constant}).

run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "sales_order_slspsn2",
   "so_mstr",
   "so_slspsn",
   false,
   false,
   true,
   false,
   true,
   false,
   true,
   "",
   {&ParameterOperator_Equals},
   {&ParameterValueType_Constant},
   "",
   {&ParameterValueType_Constant}).
```

```
run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "sales_order_slspsn3",
   "so_mstr",
   "so_slspsn",
   false,
   false,
   true,
   false,
   true,
   false,
   true,
   "",
   {&ParameterOperator_Equals},
   {&ParameterValueType_Constant},
   "",
   {&ParameterValueType_Constant}).

run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "sales_order_slspsn4",
   "so_mstr",
   "so_slspsn",
   false,
   false,
   true,
   false,
   true,
   false,
   true,
   "",
   {&ParameterOperator_Equals},
   {&ParameterValueType_Constant},
   "",
   {&ParameterValueType_Constant}).

/* Create buffer header for ttSoLine */
bufferName = "ttSoLine".
run CreateBufferHeader in reportHandle
   (bufferName, "Sales Order Lines").

/* Create fields for ttSoLine */

run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "sales_order_number",
   "so_mstr",
   "so_nbr",
   false,
   false,
   true,
   false,
   true,
   false,
   true,
   "",
   {&ParameterOperator_Equals},
   {&ParameterValueType_Constant},
   "",
   {&ParameterValueType_Constant}).

run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "sales_detail_line",
   "sod_det",
   "sod_line",
   false,
   false,
   true,
   false,
   true,
   false,
   true,
   "",
   {&ParameterOperator_Equals},
   {&ParameterValueType_Constant},
   "",
   {&ParameterValueType_Constant}).

run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "sales_detail_item",
   "sod_det",
   "sod_part",
   false,
   false,
   true,
   false,
   true,
   false,
   true,
   "",
   {&ParameterOperator_Equals},
   {&ParameterValueType_Constant},
   "",
   {&ParameterValueType_Constant}).

run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "sales_detail_qty_ord",
   "sod_det",
   "sod_qty_ord",
   false,
   false,
   true,
   false,
   true,
   false,
   true,
   "",
   {&ParameterOperator_Equals},
   {&ParameterValueType_Constant},
   "",
   {&ParameterValueType_Constant}).

run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "sales_detail_unit_measure",
   "sod_det",
   "sod_um",
   false,
   false,
   true,
   false,
   true,
   false,
   true,
   "",
   {&ParameterOperator_Equals},
   {&ParameterValueType_Constant},
   "",
   {&ParameterValueType_Constant}).

run CreateFieldLikeDBField in reportHandle
   (bufferName,
   "sales_detail_due_date",
   "sod_det",
   "sod_due_date",
   false,
   false,
   true,
   false,
   true,
      false,
      true,
      "",
      {&ParameterOperator_Equals},
      {&ParameterValueType_Constant},
      "",
      {&ParameterValueType_Constant}).

end procedure.

/* Data retrieval logic block */
procedure RunReport:

   define output parameter dataset-handle phReportResults.

   /* Retrieve the data from database */
   define variable queryString as character no-undo.
   define variable hSOQuery as handle.
   define query SOQuery for so_mstr.

   hSOQuery = query SOQuery:handle.

   queryString = "for each so_mstr no-lock "
       + " where so_mstr.so_domain = " + QUOTER(global_domain).

   run FillQueryString in reportHandle (input "ttSalesHeader", input-output
queryString).

   queryString = queryString + ":".

   hSOQuery:query-prepare(queryString).
   hSOQuery:query-open().
   hSOQuery:get-next().

   repeat while not hSOQuery:query-off-end:
      create ttSalesHeader.
      assign
         ttSalesHeader.so_nbr = so_mstr.so_nbr
         ttSalesHeader.so_cust = so_mstr.so_cust
         ttSalesHeader.so_ord_date = so_mstr.so_ord_date
         ttSalesHeader.sales_order_slspsn1 = so_mstr.so_slspsn[1]
         ttSalesHeader.sales_order_slspsn2 = so_mstr.so_slspsn[2]
         ttSalesHeader.sales_order_slspsn3 = so_mstr.so_slspsn[3].
         ttSalesHeader.sales_order_slspsn4 = so_mstr.so_slspsn[4].

      for each sod_det no-lock
         where sod_det.sod_nbr = so_mstr.so_nbr:

         create ttSoLine.
         assign
            ttSoLine.sales_order_number = sod_det.sod_nbr
            ttSoLine.sales_detail_line = sod_det.sod_line
            ttSoLine.sales_detail_item = sod_det.sod_part
            ttSoLine.sales_detail_qty_ord = sod_det.sod_qty_ord
            ttSoLine.sales_detail_unit_measure = sod_det.sod_um
            ttSoLine.sales_detail_due_date = sod_det.sod_due_date.
      end.

      hSOQuery:get-next().
   end. /* Repeat query */

   phReportResults = dataset dsReportResults:handle.

end procedure.
```

Appendix B

External Metadata

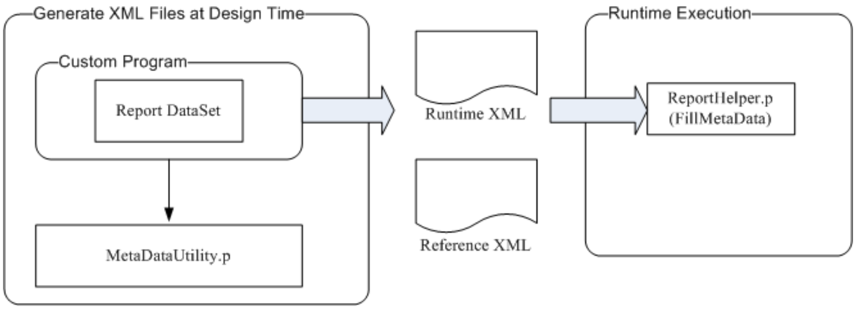
Overview

This appendix describes an alternative to hard-coding metadata in a data source program. This approach can greatly improve the ease and productivity of writing data source programs. This approach allows metadata information to be externalized to an XML file, which gets loaded at runtime. In addition to the runtime infrastructure required to implement this enhancement, a utility is also provided which allows for generating the following default XML files:

- A reference file containing base XML for each field in the report
- A reference file which provides runtime metadata for specified searchable fields

The process for making use of this new capability is illustrated below:

Fig. B.1
External Metadata



The above diagram shows the design-time and run-time steps to use the XML metadata utility.

1 At design-time, MetaDataUtility.p is used to generate XML metadata files, which can be manually edited to over-ride defaults. The XML files should then be deployed such that ReportHelper.p can read them at run-time to dynamically create the report metadata (typically in the same directory as the report data source program.)

2 The report data source program must call FillMetaData in ReportHelper.p, which creates metadata from the XML at run-time.

The following sections describe these steps in detail.

Generating Metadata XML Files

The MetaDataUtility.p program, which generates the two XML files, is invoked by a custom program that must be written. This program must include a copy of the report data set and call the MetaDataUtility.p program. An example program is shown below; its data set definition was copied from the example data source program in Appendix A.

```
/* Report data set definition block */
define temp-table ttSalesHeader
   field so_nbr like so_mstr.so_nbr
   field so_cust like so_mstr.so_cust
   field so_ord_date like so_mstr.so_ord_date
   field sales_order_slspsn1 like so_mstr.so_slspsn[1]
   field sales_order_slspsn2 like so_mstr.so_slspsn[2]
   field sales_order_slspsn3 like so_mstr.so_slspsn[3]
   field sales_order_slspsn4 like so_mstr.so_slspsn[4]
   index SalesHeaderIdx is primary so_nbr
```

```
.
define temp-table ttSoLine
   field sales_order_number like so_mstr.so_nbr
   field sales_detail_line like sod_det.sod_line
   field sales_detail_item like sod_det.sod_part
   field sales_detail_qty_ord like sod_det.sod_qty_ord
   field sales_detail_unit_measure like sod_det.sod_um
   field sales_detail_due_date like sod_det.sod_due_date
   index SoLineIdx is primary sales_order_number sales_detail_line.
.

define dataset dsReportResults for ttSalesHeader, ttSoLine
   data-relation drLine for ttSalesHeader, ttSoLine
   relation-fields (so_nbr, sales_order_number)
.

define variable vhDS as handle no-undo.
vhDS = dataset dsReportResults:handle.
run com/qad/shell/report/MetaDataUtility.p(
  vhDS,
  "UserGuideSampleReportMeta",
  "so_nbr,so_cust,so_ord_date").
```

The report dataset used in this custom program is copied from the report data source program.

MetaDataUtility Parameters

The call to the MetaDataUtility.p program has the following parameters:

Table B.1
MetaDataUtility Parameters

Name	Input/Output	Data Type	Description
ihDS	Input	Handle	DataSet handle to the report dataset — used to define default metadata settings for all of the fields
icBaseFileName	Input	Character	Base file name — specifies the base part of the generated XML file names. The runtime file, containing only the metadata for the specified field list is created as {basename}.meta.xml. The file containing the default metadata values for all the fields in the report is created in {basename}.meta.all. This latter file is generated for reference in case additional fields need to be added to the runtime XML file, to add additional search criteria, for instance.
icSearchableFields	Input	Character	Field list — specifies the list of fields which will be included as searchable fields in the runtime XML file. Note that the order in which these fields are specified in the list determines the order that they are generated, along with their criteria sort order, in the resulting XML file.

Metadata XML Elements

The specification for these metadata XML files is as follows:

Table B.2
Metadata XML Elements

Element	Required	Default	Data Type	Description
dsReportMetaData	Yes	N/A	N/A	Outermost root element.
BufferName	Yes	N/A	N/A	Outer element for each field metadata specification.
FieldName	Yes	N/A	String	Name of temp-table in report’s dataset
FieldLabel	No	“”	String	Name of field for temp-table in report’s dataset. The values for FieldLabel elements will be treated as translated label keys. If no corresponding label is found for the key, then the key itself will appear. For example, the term “SITE” should get translated in English to “Site.”
DBTableName	No	“”	String	Specifies the DB table name associated with this field.
DBFieldName	No	“”	String	Specifies the DB field name associated with this field.
DBFieldExtent	No	0	Integer	Number of elements in an array DB field. Set to 0 if not an array field.
IsSearchField	No	False	Logical	Determines whether this field is to be used for filter criteria.
ValueList	No	“”	String	Comma separated set of {term},{return value} pairs used to define selections for a filter field. The values ValueList elements will be treated as translated label keys. If no corresponding label is found for the key, then the key itself will appear. For example, the term “SITE” should get translated in English to “Site.”
IsReadOnlySearch	No	False	Logical	Specifies that this filter constraint cannot be modified.
IsVisible	No	True	Logical	Whether this field can be displayed
IsOperatorChangeable	No	True	Logical	Whether this filter field's operator can be modified by the user.
IsRequiredCondition	No	False	Logical	Specifies whether user must enter a constraint for this filterable field.
IsEditable	No	True	Logical	Whether this filter field's constraint can be modified.
DefaultValue1	No	“”	String	Default value for first filter criteria setting.
DefaultValue1Type	No	“constant”	String	The type for the first filter criteria.
DefaultOperator	No	“equals”	String	The default operator for a filter field.
DefaultValue2	No	“”	String	Default value for second filter criteria setting.
DefaultValue2Type	No	“constant”	String	The type for the second filter criteria.

An example of an XML file is provided below:

```
<dsReportMetaData>
  <ttField>
    <BufferName>ttSalesHeader</BufferName>
    <FieldName>so_nbr</FieldName>
    <FieldSequence>1</FieldSequence>
    <DBTableName>so_mstr</DBTableName>
    <DBFieldName>so_nbr</DBFieldName>
    <IsSearchField>true</IsSearchField>
  </ttField>
  <ttField>
    <BufferName>ttSalesHeader</BufferName>
    <FieldName>so_cust</FieldName>
    <FieldSequence>2</FieldSequence>
    <DBTableName>so_mstr</DBTableName>
    <DBFieldName>so_cust</DBFieldName>
    <IsSearchField>true</IsSearchField>
  </ttField>
  <ttField>
    <BufferName>ttSalesHeader</BufferName>
    <FieldName>so_ord_date</FieldName>
    <FieldSequence>3</FieldSequence>
    <DBTableName>so_mstr</DBTableName>
    <DBFieldName>so_ord_date</DBFieldName>
    <IsSearchField>true</IsSearchField>
  </ttField>
</dsReportMetaData>
```

Note The elements must be provided in the order identified in the previous table, but intermediate optional elements need not be provided. Any missing elements will result in the default value being used as specified in the previous table.

Implementing XML Metadata in the Report Data Source Program

The report data source program should invoke the FillMetaData procedure in ReportHelper.p to create metadata at run-time from the information in the XML file. Once this is done, there is no need for any hard-coded metadata statements.

The following code sample illustrates how to invoke FillMetaData from a data source program:

```
define variable vhDS as handle no-undo.

for first ttReportRequest no-lock:
  vhDS = dataset dsReportResults:handle.
  run FillMetaData in reportHandle (
    vhDS,"UserGuideSampleReportMeta.meta.xml").

  if runReport then do:
    run RunReport
      (output dataset-handle phReportResults).
  end.
end.
```

The lines highlighted in bold are the lines modified from the original code from the example in Appendix A. Recall that reportHandle is a handle to ReportHelper.p. The name of the metadata XML file in this example is `UserGuideSampleReportMeta.meta.xml`. It is recommended to use the following naming convention for metadata XML files:

<*data source program name*>.meta.xml

The input parameters to the FillMetaData procedure are as follows:

Table B.3
FillMetaData Parameters

Name	Input/Output	Data Type	Description
ihDS	Input	Handle	Handle to the report dataset — used to create default metadata records for all fields in the report dataset
icMetaDataFile	Input	Character	Name of a runtime XML metadata file — provides overrides to the default metadata values. This file is located on the appserver’s propath with “com/qad/shell/report/reports/” prepended (it should be co-located with the report proxy).

When processing each field for each temp-table in the dataset the metadata information is used to determine how these fields will be created in the report helper. If a DBFieldName is specified and is the same as the FieldName value then the CreateFieldForDBField call is made to retrieve metadata values for the specified DB field, such as an associated lookup. If a DBFieldName is specified but the DBFieldName is different then the FieldName value then the CreateFieldLikeDBField call is made, which will also populate additional values, such as the lookup. If there is no DBFieldName specified for the field then the CreateField call is made.

Implementing Filter Conditions in the Dynamic Query

In addition to loading and applying the metadata, it is also necessary to apply filter criteria constraints to dynamic queries. ReportHelper.p provides several procedures for adding in these constraints. These procedures have been enhanced to allow for adding in conditions to a DB array field (field which has DBFieldExtend metadata value greater than one), as well as allowing a temp-table field to be associated with a specific index of a DB array field.

- AddAllConditions — Used to add all filter constraints for a specified DB table name to a query string.
- AddSomeConditions — Adds all filter constraints for a specified DB table name to a query string except for the fields provided.
- AddSpecificConditions — Adds filter constraints only for specified fields for a given DB table name to a query string.

To specify a filter field which is associated with a DB array field the meta.xml is specified as follows:

```
<ttField>
    <BufferName>ttSQHeader</BufferName>
    <FieldName>qo_slspsn</FieldName>
    <FieldSequence>3</FieldSequence>
    <DBTableName>qo_mstr</DBTableName>
    <DBFieldName>qo_slspsn</DBFieldName>
    <DBFieldExtent>4</DBFieldExtent>
    <IsSearchField>true</IsSearchField>
  </ttField>
```

Note that qo_slspsn is a field in qo_mstr which is defined as having an extent of 4. When creating a query for which the qo_slspsn is constrained to the “jp” salesperson the following condition will be added to the query:

```
   and  ( ( ( ( qo_slspsn[1] = 'jp' ) ) ) or
   ( ( ( qo_slspsn[2] = 'jp' ) ) ) or
   ( ( ( qo_slspsn[3] = 'jp' ) ) ) or
   ( ( ( qo_slspsn[4] = 'jp' ) ) ) )
```

To specify a filter field which is associated with a specific index for a DB array field the meta.xml is specified as follows:

```
<ttField>
    <BufferName>ttSQHeader</BufferName>
    <FieldName>qo_slspsn_1</FieldName>
    <FieldSequence>3</FieldSequence>
    <DBTableName>qo_mstr</DBTableName>
    <DBFieldName>qo_slspsn[1]</DBFieldName>
    <IsSearchField>true</IsSearchField>
  </ttField>
```

In this case the temp-table field (qo_slspsn_1) is associated with a specific indexed field (qo_slspsn[2]) which results in a condition only for that specific indexed field being added as shown below:

```
    and ( ( ( qo_slspsn[2] = 'jp' ) ) )
```

Currently the MetaDataUtility which can be used to generate meta.xml files does not support the ability to map temp-table fields to indexed DB fields.

The AddAllConditions is used to loop through all filter fields associated with a specified DB table in the metadata and add in the filter constraints for these fields. The parameters to this procedure are the name of the DB table to retrieve filter constraints for and an input-output character string for the query to append to. This procedure has been modified to automatically include proper handling of extent and indexed DB fields.

Following is an example of the use of AddAllConditions:

```
queryString = "for each qaddb.sod_det no-lock where
   sod_det.sod_domain = " + quoter(global_domain).
queryString = queryString + " and (not sod_sched)".
queryString = queryString +
   " and (sod_qty_ord > sod_qty_ship)"
run AddAllConditions in reportHandle(
   "sod_det",input-output queryString).
queryString = queryString + " and sod_compl_stat = ''".
queryString = queryString + ",each qaddb.so_mstr no-lock where
   so_domain = " + quoter(global_domain) +
       " and so_nbr = sod_nbr".
run AddAllConditions in reportHandle(
   "so_mstr",input-output queryString).
queryString = queryString + " and so_compl_stat = '':".
```

The first call to AddAllConditions will find all the searchable constraints for report fields associated with the sod_det DB table and append them to the query string. The second call does the same thing for constraints for searchable report fields associated with the so_mstr DB table. Since the DB table associated with the searchable field is the same as the DB table being queried there is no need to perform the mapping which is provided in the AddSpecificConditions procedure described below.

The AddSomeConditions builds on the previous AddAllConditions but has an additional parameter between the DB table and query string which contains a comma separated list of DB fields to exclude from the filter constraints to be appended to the query string. This variant can be used when it is desirable to manually add some filter constraints within the proxy. This procedure has been modified to automatically include proper handling of extent and indexed DB fields.

Below is an example of how this procedure can be used:

```
queryString = "for each qaddb.sod_det no-lock where
   sod_det.sod_domain = " + quoter(global_domain).
Run AddCondition in reportHandle(
   "ttOrderData","sod_site",input-output queryString).
queryString = queryString + " and (not sod_sched)".
queryString = queryString +
   " and (sod_qty_ord > sod_qty_ship)"
run AddSomeConditions in reportHandle(
   "sod_det","sod_site",input-output queryString).
queryString = queryString + " and sod_compl_stat = ''".
queryString = queryString + ",each qaddb.so_mstr no-lock where
   so_domain = " + quoter(global_domain) +
       " and so_nbr = sod_nbr".
run AddAllConditions in reportHandle(
   "so_mstr",input-output queryString).
queryString = queryString + " and so_compl_stat = '':".
```

The previous AddCondition call is used to insert any constraints on the sod_site DB field at a specific location in the query (possibly to address performance concerns). The subsequent call to AddSomeConditions will process all the filter constraints for filterable fields except for sod_site. If multiple fields are to be excluded then each name is provided in a comma separated list.

The AddSpecificConditions is almost the opposite of the AddSomeConditions in that it takes a comma separated list of fields to include as filter conditions. However, this procedure supports the ability to specify each field as “{DBFieldName}[:{DB Field}].” This allows for specifying a list of fields for the specified DB table and have the field's name mapped from the original {DBFieldName} to a different {DB Field}. So a constraint on a {DBFieldName} of “sod_part” could be converted to a {DB Field} of “ds_part” to apply the constraint against a ds_det record instead of the sod_det record. This procedure has been modified to automatically include proper handling of extent and indexed DB fields.

```
queryString = "for each qaddb.ds_det no-lock where
   ds_det.ds_domain = " + quoter(global_domain).
run AddSpecificConditions in reportHandle(
   "sod_det",
   "sod_part:ds_part,sod_due_date:ds_shipdate," +
   "sod_req_date:ds_due_date,so_nbr:ds_nbr,sod_site:ds_site",
   input-output queryString).
run AddSpecificConditions in reportHandle(
   "so_mstr",
   "so_nbr:ds_nbr ",
   input-output queryString).
queryString = queryString + " and ds_qty_conf >= 0".
queryString = queryString + " and ds_qty_ship < ds_qty_conf".
queryString = queryString + ",each qaddb.dss_mstr no-lock where
   dss_domain = " + quoter(global_domain) + " and so_nbr = sod_nbr".
queryString = queryString + " and dss_nbr = ds_nbr".
run AddSpecificConditions in reportHandle(
   "so_mstr",
   "so_ship:dss_rec_site ",
input-output queryString).
run AddSpecificConditions in reportHandle(
   "sod_det",
   "sod_site:dss_shipsite",
   input-output queryString).
queryString = queryString + ",each qaddb.pt_mstr no-lock where
   pt_domain = " + quoter(global_domain).
queryString = queryString + " and pt_part = ds_part".
run AddAllConditions in reportHandle(
   "pt_mstr",input-output queryString).
queryString = queryString + ":".
```

In this example the filter criteria for the specified sod_det and so_mstr DB table fields are being used to filter the resulting ds_det and dss_mstr records returned. Note the use of the "{DBFieldName}:{DB Field}" notation to provide mapping between the DBFieldName specified for a reporting field's metadata and the actual DB field name in the query - this is used to convert the name of the field in the query from the metadata DB field name to the correct field for the table used in the query.

The AddExtentConditions is used to bring in all the conditions which apply to an extent DB field. Since this procedure is automatically called from the previous procedures, it will probably not be needed to call directly, but is included here for documentation purposes. The input parameters to this procedure are the name of the temp-table buffer, name of the temp-table field, name of the DB field and number of indexed elements in the DB array field. There is also an input-output parameter for the query string which gets updated with the constraints for this filterable array. This procedure is only intended to be used when an extent DB field is being referenced, it should not be called for non-extent DB fields.

Following is an illustration of how this procedure could be called (note that this example is contrived, since similar behavior would occur from calling AddSpecificConditions):

```
queryString = queryString + "for each qo_mstr no-lock where
   qo_mstr.qo_domain = " + quoter(global_domain).
run AddExtentConditions in reportHandle(
   "qo_mstr","qo_slspsn","qo_slspsn",4,input-output queryString).
queryString = queryString + ":".
```

The call to AddExtentConditions will cause the filter condition applied to the qo_slspsn field to be applied to each indexed element in the DB qo_slspsn field.

Implementing Filter Conditions in the Example Program

In the example program we have been developing, the filter conditions can most easily be implemented by using the following code block in the RunReport procedure.

```
   hSOQuery = query SOQuery:handle.

   queryString = "for each so_mstr no-lock "
      + " where so_mstr.so_domain = " + QUOTER(global_domain).

   run AddAllConditions in reportHandle ("so_mstr", input-output queryString).

   queryString = queryString + ":".

   hSOQuery:query-prepare(queryString).
   hSOQuery:query-open().
   hSOQuery:get-next().
```

The `AddAllConditions` call will suffice to dynamically add the filter conditions for all of our searchable fields, since they are all in the same DB table (`so_mstr`). The entire source code for this example program is listed at the end of this appendix.

ReportHelper.p XML Metadata Procedure Reference

Following are the available procedures and functions related to XML metadata processing in the ReportHelper.p program.

SetIsTranslated

Used to set whether display values are to be treated as pre-translated values or as untranslated terms. A value of false will treat values as terms and attempt to convert them, whereas a value of true will bypass this conversion (for example, if the values are already translated).

Table B.4
SetIsTranslated Parameters

Name	Input/Output	Data Type	Description
ilIsTranslated	Input	Logical	Input logical specifies value for whether values are to be treated as already translated

FillMetaData

Procedure called from proxy to build report metadata.

Table B.5
FillMetaData Parameters

Name	Input/Output	Data Type	Description
ihDS	Input		Input handle to the report dataset which is used to create default metadata records for all fields in the report dataset
icMetaDataFile	Input	String	Input string for the name of a runtime XML metadata file which will provide overrides to the default metadata values. This file is located on the appserver’s propath with “com/qad/shell/report/reports/” prepended (it should be collocated with the report proxy).

SetMetaData

Wrapper to SetFieldMetaParameter procedure in ReportHelper program which provides for optional translation of field labels and display part of value lists. These values are only translated if the SetIsTranslated procedure is called with a false value, otherwise the values are passed directly to the ReportHelper without performing any translation.

Table B.6
SetMetaData Parameters

Name	Input/Output	Data Type	Description
icBufferName	Input	Character	Input name of the report temp-table to assign metadata value for
icField	Input	Character	Input name of the report temp-table's field to assign metadata value for
icMetaField	Input	Character	Input name of metadata field as specified by ReportHelper
icMetaValue	Input	Character	Input value to assign to the specified metadata field

AddAllConditions

Used to add all filter constraints for a specified DB table name to a query string.

Table B.7
AddAllConditions Parameters

Name	Input/Output	Data Type	Description
icDBTable	Input	Character	Input name of DB table to add filter constraints for
bcQueryString	Input	Character	Input-Output string to append filter constraints to

AddSomeConditions

Adds all filter constraints for a specified DB table name to a query string except for the fields provided.

Table B.8
AddSomeConditions Parameters

Name	Input/Output	Data Type	Description
icDBTable	Input	Character	Input name of DB table to add filter constraints for
icExclusionList	Input	Character	Input comma separated string of fields to exclude from constraints
bcQueryString	Input/Output	Character	Input-Output string to append filter constraints to

AddSpecificConditions

Adds filter constraints only for specified fields for a given DB table name to a query string.

Table B.9
AddSpecificConditions Parameters

Name	Input/Output	Data Type	Description
icDBTable	Input	Character	Input name of DB table to add filter constraints for
vcFieldData	Input	Character	Input comma separated string of fields for which to add constraints. Syntax of each field is as follows: {DBFieldName}[:{DB Field}] Where {DBFieldName} is the field’s metadata DBFieldName value and {DB Field} is the name of the field as it will be used in the resulting filter constraint
entryName	Input/Output	Character	Input-Output string to append filter constraints to

AddExtentConditions

Adds filter constraints for a specific extent DB field, applying the constraint to each element in the DB field array. (This procedure is typically called internally from the ReportHelper.p program, but is listed here for reference.)

Table B.10
AddExtentConditions Parameters

Name	Input/Output	Data Type	Description
icBufferName	Input	Character	Input name of the temp-table
icFieldName	Input	Character	Input of the temp-table field
icDBFieldName	Input	Character	Input name of corresponding database field
iiExtent	Input	Integer	Number of elements in the array for the database field
bcQueryString	Input/Output	Character	Input-Output string to append filter constraints to

Complete Data Source Program Sample Code (Using XML Metadata)

```
{mfdeclre.i}
{gplabel.i}

{com/qad/shell/report/dsReportRequest.i}
{com/qad/shell/report/ReportConstants.i}

/* Report data set definition block */
define temp-table ttSalesHeader
   field so_nbr like so_mstr.so_nbr
   field so_cust like so_mstr.so_cust
   field so_ord_date like so_mstr.so_ord_date
   field sales_order_slspsn1 like so_mstr.so_slspsn[1]
   field sales_order_slspsn2 like so_mstr.so_slspsn[2]
   field sales_order_slspsn3 like so_mstr.so_slspsn[3]
   field sales_order_slspsn4 like so_mstr.so_slspsn[4]
   index SalesHeaderIdx is primary so_nbr
.
define temp-table ttSoLine
   field sales_order_number like so_mstr.so_nbr
   field sales_detail_line like sod_det.sod_line
   field sales_detail_item like sod_det.sod_part
   field sales_detail_qty_ord like sod_det.sod_qty_ord
   field sales_detail_unit_measure like sod_det.sod_um
   field sales_detail_due_date like sod_det.sod_due_date
   index SoLineIdx is primary sales_order_number sales_detail_line.
.

define dataset dsReportResults for ttSalesHeader, ttSoLine
   data-relation drLine for ttSalesHeader, ttSoLine
   relation-fields (so_nbr, sales_order_number)
.

/* Main Block */
define input parameter runReport as logical.
define input parameter reportHandle as handle.
define input parameter dataset for dsReportRequest.
define output parameter dataset-handle phReportResults.

{com/qad/shell/report/reporting.i}

define variable bufferName as character no-undo.

/* Empty temp-table block */
empty temp-table ttSalesHeader no-error.
empty temp-table ttSoLine no-error.

define variable vhDS as handle no-undo.

for first ttReportRequest no-lock:
  vhDS = dataset dsReportResults:handle.
  run FillMetaData in reportHandle (
    vhDS,"UserGuideSampleReportMeta.meta.xml").

  if runReport then do:
    run RunReport
      (output dataset-handle phReportResults).
  end.
end.

/* Data retrieval logic block */
procedure RunReport:

   define output parameter dataset-handle phReportResults.

   /* Retrieve the data from database */
   define variable queryString as character no-undo.
   define variable hSOQuery as handle.
   define query SOQuery for so_mstr.

   hSOQuery = query SOQuery:handle.

   queryString = "for each so_mstr no-lock "
      + " where so_mstr.so_domain = " + QUOTER(global_domain).

   run AddAllConditions in reportHandle ("so_mstr", input-output queryString).

   queryString = queryString + ":".

   hSOQuery:query-prepare(queryString).
   hSOQuery:query-open().
   hSOQuery:get-next().

   repeat while not hSOQuery:query-off-end:
      create ttSalesHeader.
      assign
         ttSalesHeader.so_nbr = so_mstr.so_nbr
         ttSalesHeader.so_cust = so_mstr.so_cust
         ttSalesHeader.so_ord_date = so_mstr.so_ord_date
         ttSalesHeader.sales_order_slspsn1 = so_mstr.so_slspsn[1]
         ttSalesHeader.sales_order_slspsn2 = so_mstr.so_slspsn[2]
         ttSalesHeader.sales_order_slspsn3 = so_mstr.so_slspsn[3].
         ttSalesHeader.sales_order_slspsn4 = so_mstr.so_slspsn[4].

      for each sod_det no-lock
         where sod_det.sod_nbr = so_mstr.so_nbr
         and sod_det.sod_domain = global_domain:

         create ttSoLine.
         assign
            ttSoLine.sales_order_number = sod_det.sod_nbr
            ttSoLine.sales_detail_line = sod_det.sod_line
            ttSoLine.sales_detail_item = sod_det.sod_part
            ttSoLine.sales_detail_qty_ord = sod_det.sod_qty_ord
            ttSoLine.sales_detail_unit_measure = sod_det.sod_um
            ttSoLine.sales_detail_due_date = sod_det.sod_due_date.
      end.

      hSOQuery:get-next().
   end. /* Repeat query */

   phReportResults = dataset dsReportResults:handle.

end procedure.
```

Index