TOC

About This Guide **34**

Element Naming Conventions **35**

Element ID Notation **36**

CamelCase Notation 36

Hungarian Notation 36

Standardization Is Good 37

Formulating Mnemonics **38**

Suggested Mnemonics for Common Elements **39**

Element Position **42**

About Flow and Absolute Positioning 42

Selecting the Positioning Scheme **46**

Switching Back from Absolute to Flow 47

Using the Layout Editor **48**

Overlaying Elements 50

Show Modes **51**

About Show Modes 51

Show Modes in Action 52

Show Modes vs. Conditions 53

Using Built-in Show Modes Values **55**

Setting the Report-Level Default Show Modes **58**

Overriding Default Show Modes with Action.Report **59**

Toggling Show Modes with Action.Show Element **61**

Local Data is Available Everywhere **63**

Procedure.XML Modifier **66**

About Procedure.XML Modifier 66

Procedure.XML Modifier - Child Modifier Elements **68**

Procedure.XML Modifier - Usage Example: Set Attribute **71**

Procedure.XML Modifier - Usage Example: Append Child Element **73**

Procedure.XML Modifier - Usage Example: Prepend Child Element **78**

Procedure.XML Modifier - Usage Example: Set Element Text **82**

Data Calendar **83**

About the Data Calendar Element 83

Data Calendar Attributes **85**

Working with the Data Calendar **88**

Globalization and Localization **91**

Overriding the Default Style Class **93**

The Aggregate Column **95**

About the Aggregate Column 95

Dynamic Application 95

Aggregate Column Attributes **97**

The Concat Function 98

Aggregate Column Usage Example **99**

The DsAggregate Column **101**

About the DsAggregate Column 101

DsAggregate Column Attributes **102**

DsAggregate Column Usage Example **103**

Dynamic Application 104

The DsCalculated Column **106**

About the DsCalculated Column 106

DsCalculated Column Attributes **107**

DsCalculated Column Usage Examples **108**

The DsCondition Filter **112**

Applying the DsCondition Filter **113**

Using Multiple and Dynamic DsCondition Filters **115**

Use with DsCompare Filters **117**

DsGroup **119**

Grouping Data **120**

Grouping on Multiple Columns 121

Conditional Grouping 122

Aggregating Grouped Data **124**

DsSort **126**

Using the DsSort Element 126

Dynamic Sorting 129

The Analysis Chart **130**

About the Analysis Chart 130

Studio Wizard Available 139

Retaining User Settings 141

Embedded Sub-Report Caveat 141

Analysis Chart - Attributes **142**

Analysis Chart - Add Analysis Chart Columns **145**

Analysis Chart - Adding Additional Series **149**

Analysis Chart - Adding Forecasting **156**

Analysis Chart - Saving Individual User Settings **159**

Analysis Chart - Adding Charts to Dashboards **160**

162

Analysis Chart - Customizing Analysis Chart Appearance ... **166**

Changing Appearance Using Style Classes ... 166

Changing Appearance Using Template Modifiers ... 166

The Analysis Filter ... **168**

About the Analysis Filter ... 168

Retaining User Settings ... 175

Analysis Filter - Using the Analysis Filter Wizard ... **176**

Analysis Filter - Attributes ... **179**

Analysis Filter - Column Attributes ... **183**

Analysis Filter - Using the Analysis Filter Elements ... **184**

Passing Parameters ... 189

Analysis Filter - Saving Individual User Settings ... **191**

Analysis Filter - Filtering with Query String Parameters ... **192**

Remove all Filter Definitions ... 192

Force a Page Refresh with Current Filter Values ... 192

Define a New Filter ... 192

Analysis Filter - Using Analysis Filters in Dashboards ... 195

Filtering Individual Panels ... 195

Adding a Global Filter ... 197

Global Filtering using Chart Data Points ... 202

Analysis Filter - Customizing the Analysis Filter Appearance ... 207

Changing Appearance Using Style Classes ... 207

Changing Appearance Using Template Modifiers ... 207

The Calculated Column ... 209

About the Calculated Column ... 209

Calculated Column Attributes ... 210

Calculated Column Usage Examples ... 212

NaN Errors ... 216

The Color Range Column ... 217

About the Color Range Column ... 217

Color Range Column Attributes ... **218**

Color Range Element Attributes ... 219

Color Range Column Usage Examples: Numeric Data ... **220**

Color Range Column Usage Example: Text Data ... **224**

The Compare Filter ... **228**

About the Compare Filter ... 228

Compare Filter Attributes ... **229**

Applying the Compare Filter ... **231**

Comparison Operator Options ... 231

Compare Filter Usage Example ... **237**

Using Multiple and Dynamic Compare Filters ... **239**

Use with a Condition Filter ... **241**

The Contain Filter ... **243**

About the Contain Filter ... 243

Applying the Contain Filter **244**

Using Multiple and Dynamic Contains **245**

The Condition Filter **248**

About the Condition Filter 248

Applying the Condition Filter **249**

Using Multiple and Dynamic Condition Filters **251**

Use with Compare Filters **253**

The Crosstab Filter **255**

About the Crosstab Filter 255

Applying the Crosstab Filter to a Crosstab Table **258**

Crosstab Grouping 264

Crosstab Summary 273

Summary Rows 273

Header Rows 278

Crosstab Extras 282

Applying the Crosstab Filter to a Chart **285**

Using Dynamic Crosstab Filters **289**

The DeDuplicate Filter **290**

About the DeDuplicate Filter 290

Applying the DeDuplicate Filter **291**

Using Dynamic DeDuplicate Filters **292**

The File Column (BLOBs) **294**

About BLOBs and CLOBs 294

Preparing Data in a Datalayer **295**

Working with the Retrieved Data **297**

Using the Data 298

Example: Retrieving Dashboard Configuration Data **300**

Requirements 300

Writing the Data 300

Reading the Data 301

Using the Data in the Dashboard 303

The Flattener **305**

About the Flattener 305

Flattener Attributes **306**

Flattener Usage Examples **308**

The Forecasting Elements **311**

About Forecasting 311

Forecasting Methodologies 313

Using Forecast.Current Time Period **314**

Using Forecast.Regression **316**

Using Forecast.Time Series Decomp **318**

The Formatted Column **320**

About the Formatted Column Element 320

Dynamic Formatting 321

Usage Examples 321

Format: Currency ... 322

Format: Short Date ... 324

Format: Month Abbreviations ... 325

The Group Filter ... 328

About the Group Filter ... 328

Grouping Data ... 329

Grouping for Unique Rows ... 331

Grouping on Multiple Columns ... 333

Conditional Grouping ... 335

Aggregating Grouped Data ... 337

Using Group DrillThrough ... 340

The RegEx Filter ... 343

About the RegEx Filter ... 343

Applying the RegEx Filter ... 345

Using Multiple and Dynamic RegEx Filters ... 346

The Relevance Filter **348**

About the Relevance Filter 348

Dynamic Filtering 348

Applying the Relevance Filter 349

The Sort Filter **356**

About the Sort Filter 356

Dynamic Sorting 356

Applying the Sort Filter 357

The Switch Column **362**

About the Switch Column 362

Applying the Switch Column **363**

Using Multiple and Dynamic Switch Columns **366**

Time Period Columns **368**

About the Time Period Column 368

Dynamic Application 368

Time Period Columns Attributes ... **370**

Globalization Settings ... 372

Culture Settings ... 373

Time Period Columns Simple Usage Example ... **374**

Time Period Columns Advanced Usage Example ... **379**

The UnCrosstab Filter ... **382**

About the UnCrosstab Filter ... 382

Applying the UnCrosstab Filter ... **384**

Using Dynamic UnCrosstab Filters ... **385**

The Unit Conversion Column ... **386**

About the Unit Conversion Column ... 386

Applying the Unit Conversion Column ... **388**

Using Multiple and Dynamic Unit Conversion Columns ... **389**

Customizing the Conversion Factors ... **391**

Auto Columns ... **393**

How Is This Useful? 394

Action Elements 396

About Action Elements 396

The Action.Refresh Element 400

Using the Wait Panel 401

Action Elements Example: Link to Another Report 402

Action Elements Example: Link to a URL 405

Opening a File 406

Action Elements Example: Link that Executes JavaScript 407

Action Elements Example: Export to PDF 408

Action Elements Example: Email Report as Attachment 409

Action Elements Example: Showing and Hiding Elements 411

Action Elements Example: With an Event Handler 414

Action Elements Example: With an Event Handler Popup Menu 415

Action.Show Element 417

About Action.Show Element ... 417

Action.Show Element Example: Show/Hide Divisions ... 420

Action.Show Element Example: Show/Hide a More Info Row ... 423

Shared Elements ... 426

About Shared Elements ... 426

Creating a Shared Element Definition ... 428

Using Shared Elements in a Report Definition ... 430

Passing Parameters to Shared Elements ... 432

Passing Elements to Shared Elements ... 436

Shared Element Usage Tips ... 438

Repeat Elements ... 440

About the Repeat Elements Element ... 440

Some Token & Element Restrictions ... 442

Repeat Elements Example: Building a Comma-Separated String of Values ... 443

Repeat Elements Example: Dynamic Data Table Columns ... 445

Special Crosstab Table Considerations ... 448

Repeat Elements Example: Dynamic Tables with Nested Repeat Elements ... 450

Repeat Elements Example: Dynamic Analysis Grid Columns ... 458

Responsive Design Elements ... 461

About Responsive Design ... 461

Using Responsive Rows and Columns ... 462

Implementation ... 466

Using Responsive Visibility ... 469

Implementation ... 471

Forecast Methodologies ... 473

About Forecasting ... 473

Which Method is Right? ... 474

Regression ... 476

Input Data Requirements ... 476

Results ... 477

Prorating **478**

Input Data Requirements 478

Method Implementation 478

Results 479

Time Series Decomposition **480**

Input Data Requirements 480

Data Validation 481

Time Series Iteration 482

Method Implementation 483

Forecast Calculations Example 484

Results 486

Logi OLAP **487**

About Logi OLAP 488

Logi OLAP - Elements **489**

Caveats and Limitations 493

Logi OLAP - Connecting to Data **494**

Logi OLAP - Creating a Basic OLAP Grid **495**

Logi OLAP - Other OLAP Grid-related Elements **499**

MDX Query Child Elements 499

OLAP Grid Child Elements 501

Logi OLAP - Using the OLAP Grid **505**

Logi OLAP - Using the OLAP Table **514**

Logi OLAP - Using OLAP Elements with Logi XOLAP **515**

Logi OLAP - Re-using the MDX Query **516**

Logi OLAP - Using Two-Dimensional Data **517**

Logi OLAP - Adding OLAP Grid Visualizations to Dashboards **518**

Logi OLAP - Customizing OLAP Grid Appearance **523**

Changing Appearance Using Style Classes 523

Changing Appearance Using Template Modifiers 523

Logi XOLAP **525**

About Logi XOLAP 525

Logi XOLAP - About Data Cubes **527**

Logi XOLAP - Logi XOLAP Data Cubes **529**

Logi XOLAP - The Dimension Grid **532**

Logi XOLAP - Using XOLAP with OLAP **534**

Dimension Grid **535**

About the Dimension Grid 535

Dimension Grid Attributes **540**

Dimension Grid Settings Panels **541**

Configuring the Data Cube **546**

Sorting Dimensions by Dates 548

Member Properties and Calculated Measures 549

Defining an Initial View **551**

Logi XOLAP on the Debugger Page **553**

Adding Visualizations to Dashboards **554**

Refresh, Reset, and Save Grid Settings **560**

Customizing Dimension Grid Appearance **564**

Changing Appearance Using Style Classes 564

Changing Appearance Using Template Modifiers 564

User Input Elements **566**

About Input Elements 567

Input Hidden **569**

Using It 569

Getting Its Data 570

More Information 570

Input Text **571**

Using It 571

Getting Its Data 573

Input Text Delimiter Child Element 574

More Information 575

Input Password ... **576**

Using It ... 576

Getting Its Data ... 576

More Information ... 576

Input Date ... **577**

Using It ... 579

Getting Its Data ... 581

Globalization and Localization ... 581

More Information ... 584

Date Picker ... **585**

Using It ... 585

Getting Its Data ... 587

More Information ... 588

Input Email ... **589**

Using It ... 589

Getting Its Data ... 590
More Information ... 590
Input Number (Mobile Only) ... **591**
Using It ... 591
Getting Its Data ... 592
More Information ... 592
Input Telephone (Mobile Only) ... **593**
Using It ... 593
Getting Its Data ... 594
More Information ... 594
Input Time ... **595**
Using It ... 595
Getting Its Data ... 597
Globalization and Localization ... 598
More Information ... 598

Input File Upload **599**

Using It 599

Getting Its Data 599

More Information 600

Input Text Area **601**

Using It 601

Getting Its Data 602

More Information 602

Input Check Box **603**

Using It 603

Getting Its Data 604

More Information 605

Input Select List **606**

Using It 606

Hierarchical Grouping 609

Adding Default Values ... 611
Getting Its Data ... 611
More Information ... 613
Input Check Box List ... **614**
Using the Input Check Box List ... 614
Displaying Hierarchical List Items ... 618
Getting Its Data ... 619
More Information ... 620
Input Combo List ... **621**
Using It ... 621
Getting Its Data ... 622
More Information ... 623
Input Radio Buttons ... **624**
Using It ... 624
Getting Its Data ... 626

More Information ... 626

Input Slider ... **627**

Using It ... 627

Getting Its Data ... 630

More Information ... 630

Input Color Picker ... **631**

Using It ... 631

Getting Its Data ... 632

More Information ... 633

Input Chart.List ... **634**

Using It ... 635

Getting Its Data ... 636

More Information ... 636

Input Chart.Range ... **637**

Using It ... 638

Getting Its Data ... 639

Special Event Elements ... 639

More Information ... 640

Input Selection Point and Range ... 641

Working with User Input Elements ... 642

When Are Input Values Available? ... 642

Working with Change Flags ... 644

Aligning Input Elements ... 645

Alignment Using Style Classes ... 648

Using AutoComplete ... 650

Wrapping in a Fieldset Box ... 652

Inclusion in Data Tables ... 654

Populating Input Element Collections ... 655

Setting Default Values ... 658

Re-displaying Entered Values ... 660

Setting the Default Value Attribute ... 660
Saving the Value in a Cookie ... 660
Saving the Value in Local Storage ... 661
Validating Input ... 662
Using Validation Elements ... 663
Validation.Date ... 664
Validation.Email ... 664
Validation.JavaScript ... 665
Validation.Length ... 667
Validation.Numeric ... 668
Validation.Required ... 668
Validation.Time ... 669
Using Event Handlers ... 670
Event Handler Mechanics ... 670
Using Event Handlers and Dependent Controls ... 671

Using Event Handlers with Javascript **678**

Example 1: Ensure that some data is entered 678

Example 2: Ensure some data is entered but not more than 4,000 characters 679

Example 3: Replace all occurrences of '@' with '#' 679

Example 4: Handling Links within Tables 680

Passing Input Select List Captions **683**

Super Elements **685**

About Super-Elements 685

Customizing Super-Elements 688

Table of Super-Elements 688

Data Menu **691**

About the Data Menu 691

Data Menu Attributes **694**

Data Menu Example: Simple Menu **696**

Navigating to Another Report 699

Data Menu Example: Complex Menu (with sub-menus) ... **701**

Changing Appearance with Template Modifiers ... **709**

The Wait Panel ... **710**

About the Wait Panel Element ... 710

Global Wait Panel ... 711

Using the Wait Panel ... 712

Tabbed Panels ... **715**

About Tabbed Panels ... 715

Basic Tab Elements ... **718**

Working with the Tabs Element ... **720**

Maintaining the Tab Selection Between Page Refreshes ... 722

Refreshing or Navigating to a Specific Tab ... 723

Working with the Tab Panel Element ... **724**

Changing Tab Panel Content Dynamically ... 725

Popup Panels ... **729**

About the Popup Panel ... 729

Popup Panel Attributes ... **733**

Using a Popup Panel with a Link ... **736**

Using a Popup Panel with the onMouseOver Event ... **741**

Passing Parameters to a Popup Panel ... **745**

Data Entry in Popup Panels ... **748**

The Tooltip Panel ... **751**

ReportCenter Menu ... **753**

About the ReportCenter Elements ... 753

Other Menu Variants ... 755

Including Reports in the Menu ... **756**

Creating a ReportCenter Menu ... **759**

Controlling the Report Links Dynamically ... 761

Ordering Menu Items ... 762

Ordering Menu Folder Items ... 762

Scheduling Report Distribution **764**

Popup Menus **768**

About Pop-up Menus 768

Other Menu Variants 769

Adding an Image to the Pop-up Options **770**

Using the onMouseOver Event **771**

Controlling Pop-up Location **772**

Controlling Pop-up Appearance **774**

Menu Tree Menus **776**

About the Menu Tree 776

Other Menu Variants 778

Menu Tree Attributes **779**

Menu Branch and Leaf Elements **781**

Creating a Menu Tree Menu **783**

Customizing Menu Appearance **784**

Adding a "Separator" Menu Item 785

Glossary **786**

About This Guide

This is an archived copy of the v23 documentation provided for Logi Info v23.3 and its service packs.

Notice: Archived Documentation

This documentation is provided as a courtesy reference for a version of our software that is no longer under active development or support. The information contained herein is offered without warranties of any kind, either expressed or implied, including but not limited to warranties of accuracy, completeness, or fitness for a particular purpose.

While this archived material may assist with understanding historical functionality, please be aware that the software described is no longer maintained at this version level and may contain outdated or inaccurate information. Images may not reflect currently supported modules, support sites, or third party products. This software may not be compatible with current versions of previously compatible third party products.

To access and upgrade to current software solutions and receive ongoing support, please contact our customer support team. They can assist you in migrating to the latest appropriate software version that meets your needs. Our support team is available to help ensure a smooth transition to actively maintained alternatives that provide the functionality and reliability you require.

Element Naming Conventions

Almost all elements have an ID (element identifier) attribute. This is an important attribute and should always be set to a unique value; if left blank, Logi Info will automatically provide an ID for some elements using a **standardnaming convention** when assigning IDs is good practice, as it results in a more easily understood definition.

The following topics discuss the naming conventions recommended by Logi Analytics:

- Element ID Notation
- Formulating Mnemonics
- Suggested Mnemonics for Common Elements

Element ID Notation

These are the rules for the use of characters in element IDs:

- Spaces within an ID are not allowed and Logi Studio will automatically remove them from the value when the focus changes.
- For an *optional* ID, you can use any characters except a space or a double-quote for its value.
- For a *required* ID, any combination of letters, numbers, and the underscore ("_"), dash ("-"), and period (".") characters may be used, but an ID value may not *begin* with a number. For example, XYZ9 is valid, but 9XYZ is invalid.

Because spaces are invalid, we recommend that you use a combination of two well-known ID formatting schemes, "CamelCase" and "Hungarian" notation, when naming your elements.

CamelCase Notation

CamelCase (sometimes called "CamelBack") notation is the practice of writing compound words or phrases in which the parts are joined without spaces, with each part's initial letter capitalized and the first letter in either upper or lower case. For example: SampleDataTable.

Hungarian Notation

IDs that use Hungarian notation identify elements by starting with one or more lower-case letters which are mnemonics for the type or purpose of the element, followed by whatever name you chose. An element that allows a user to input a starting date, for example, might be given the ID "inpStartDate", where "inp" indicates that it's a user input element. Similarly, a Label element that's used as a link to start an export to PDF might be given the ID "lblExportPDF". These examples also show how the two notations are combined in an ID.

Standardization Is Good

The standardized use of these notational schemes provides numerous benefits, including promoting **uniqueness** and **consistency**. This can be especially helpful for developers who are new to Logi reporting products and who have not worked with them long enough to recognize the element icons. When it comes to formulating element IDs, it's not necessarily important that you try to conform to some universal standard; what *is* important is that you use some consistent, meaningful standard, perhaps your own or your company's, regularly.

Formulating Mnemonics

Here are some suggested guidelines for formulating the mnemonics used as element ID prefixes:

Guideline	Example
Use the first 2-3 letters of the element name	ag = Analysis Grid div = Division inp = Input Text
Use the first few letters of each sub-part of the element name	dt = Data Table dl = DataLayer gm = Google Map sdac = SubData Aggregate Column
Use all or part of the element name without vowels (three letters max)	dsh = Dashboard img = Image lbl = Label
Use sound-related, three-letter combinations	btn = Button img = Image
Apply any of the above to the second part of a two-part element type	pie = Chart.Pie ndl = Animated Gauge.Needle

Suggested Mnemonics for Common Elements

This table presents suggested mnemonics for some commonly-used elements:

Element	Mnemonic
Action	act
Analysis Grid	ag
Animated Chart.Pie	apie
Animated Chart.XY	axy
Button	btn
Chart.Pie	cpie
Chart.XY	cxy
Column	col
Connection	conn
Crosstab Table	xtb
Dashboard	dash

Element	Mnemonic
DataLayer	dl
DataLayer Link	dlnk
Data MultiColumn List	dmc
Data Table	dt
Data Tree	dtr
Dimension Grid	dg
Division	div
Event Handler	evh
Google Map	gm
Image	img
Interactive Data View	idv
Label	lbl

Element	Mnemonic
Procedure	proc
Response	rsp
Shared Element	shr
Target	trg
Task	tsk

Element Position

Logi Info can use two methods for positioning elements on the report page: **Flow** positioning and **Absolute** positioning**.**

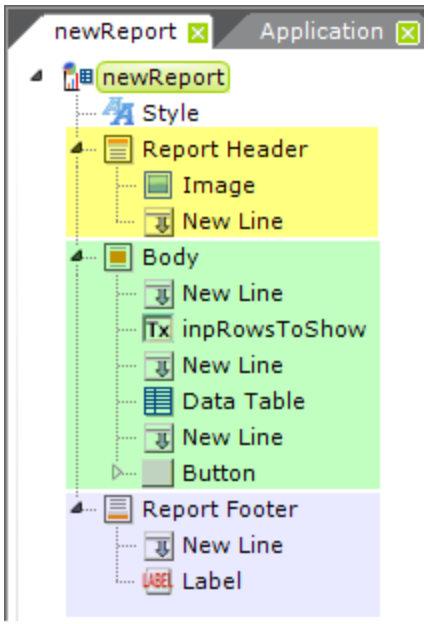
The following topics introduce you to these positioning schemes:

- Selecting the Positioning Scheme
- Using the Layout Editor

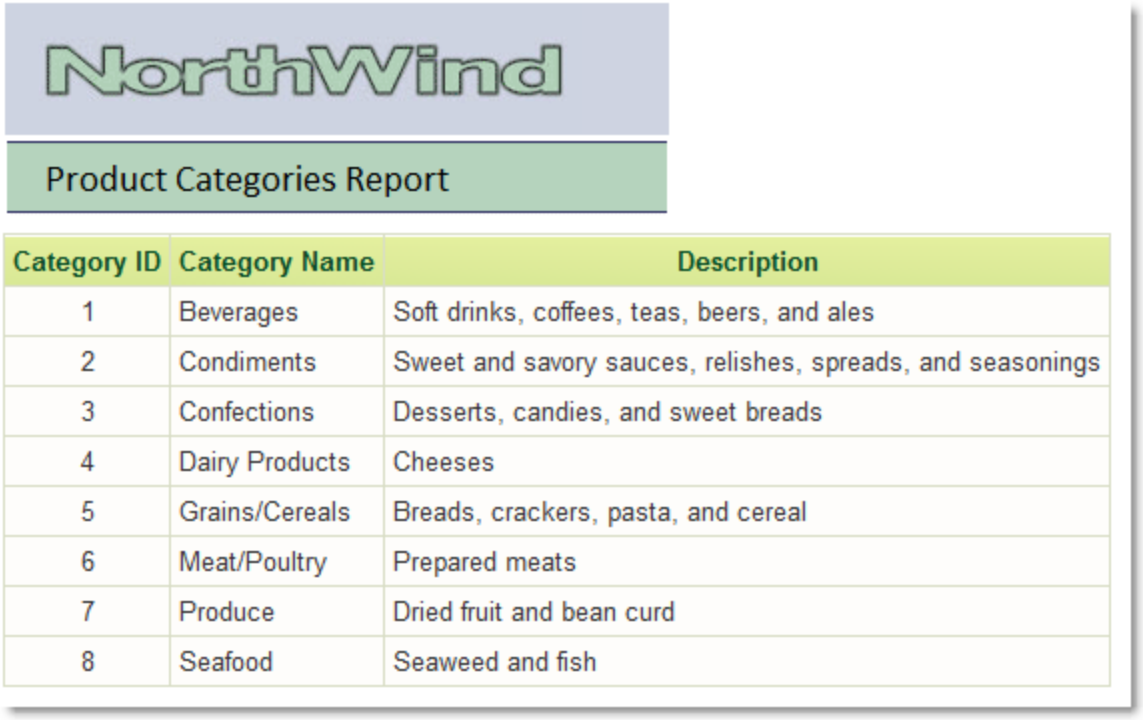

Do not switch layout modes, even just as an experiment, without reading the section Switching Back from Absolute to Flow "Selecting the Positioning Scheme" on page 46 first. Absolute positioning is retained in Logi Analytics products primarily for *backward compatibility* with earlier versions and we highly recommend that you *do not use it* for any new development work.

About Flow and Absolute Positioning

Logi application report definitions are developed in Studio using the Element Tree, which is a hierarchical arrangement of elements. **Flow positioning** determines the layout of the elements on the report page by order of appearance in the tree and each element's position is based on the previous element's position.



For example, as shown above, most report definitions contain elements in the element tree grouped beneath **Report Header**, **Body** and **Report Footer** elements, which are arranged from top to bottom. On the actual report page, elements in the Report Header will appear above elements in the Body and elements in the Body will appear above elements in Report Footer.



Category ID	Category Name	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

Flow positioning is the default layout for report definitions created in Logi Studio and works well with multiple browser types and mobile devices. The use of HTML tables (**Row**, **Rows**, and **Column Cell** elements in Logi definitions), responsive elements, and the careful application of style classes can achieve very accurate alignment of elements on the report page under Flow positioning and, for most situations, it's the best approach.

Absolute positioning, however, determines the location of the elements on the report page by assigning groups of elements exact pixel offsets from the top and left side of the browser window.

When Absolute positioning is in use, Studio's Layout editor enables you to place elements in absolute positions on the screen by dragging and dropping them (it's not completely WYSIWYG, but it's close).

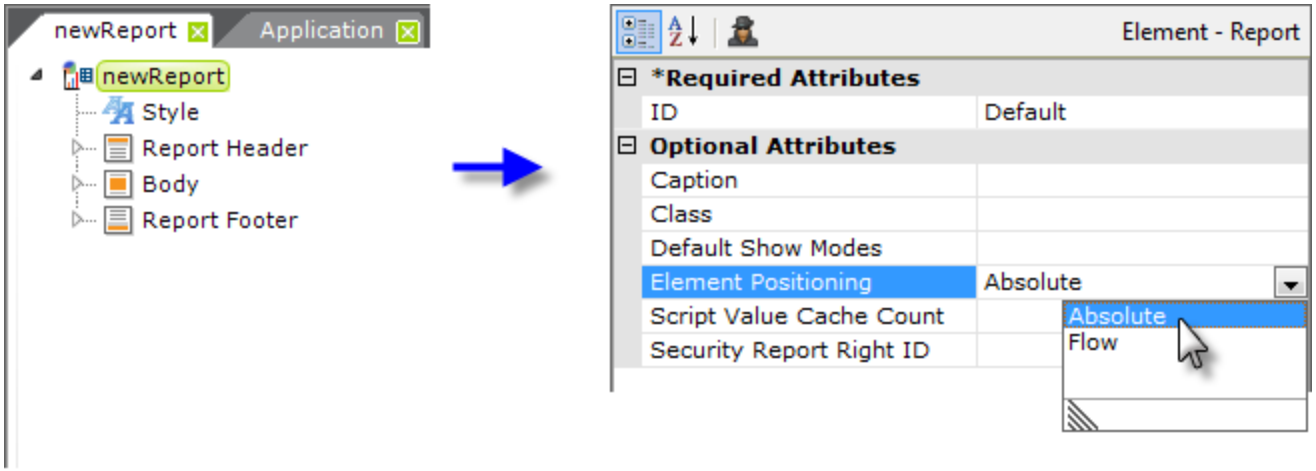


Category ID	Category Name	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

The example above shows the previous report after Absolute positioning has been enabled and elements have been positioned as desired.

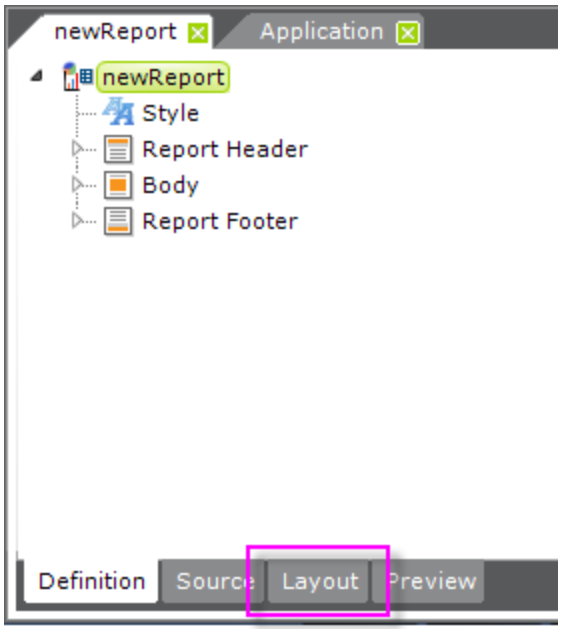
Selecting the Positioning Scheme

While Flow positioning is the default, the positioning scheme can be selected on a report-by-report basis.



The positioning scheme is controlled, as shown above, by selecting the **Report** (root or top) element in a definition and selecting the desired scheme in its **Element Positioning** attribute value. A blank entry selects the default scheme, *Flow* positioning.

Two things happen when **Absolute** positioning is selected. First, all elements that can contain other elements (Division, Body, Row, etc.) are given two "hidden" attributes: **absoluteTop**, and **absoluteLeft**. These attributes do not appear in the Attribute panel but are managed behind-the-scenes by Studio. The element will also have **Height** and **Width** attributes which may or may not be visible depending on the nature of the element. These attributes allow the Logi Server engine to accurately place elements on the page when a report is being generated.



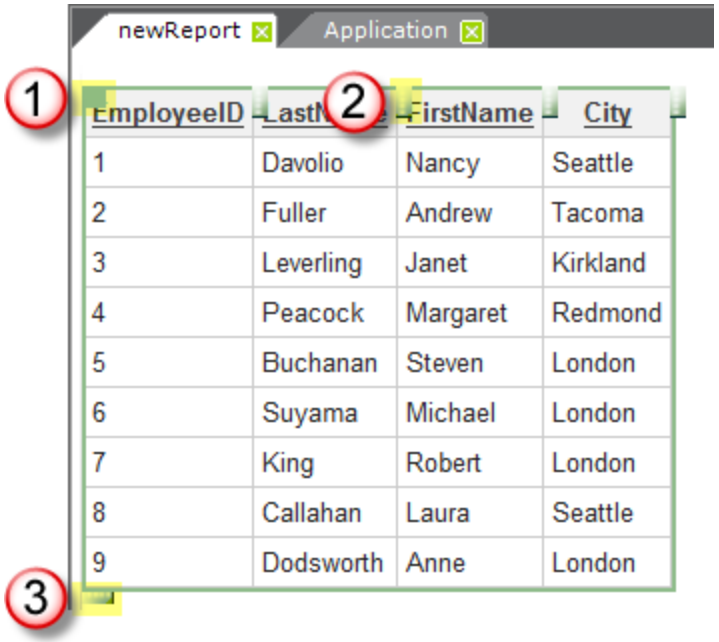
The second thing that happens is that the bottom of the Workspace panel acquires a new tab, the **Layout** tab, as shown above. Clicking this tab opens the **Layout editor**.

Switching Back from Absolute to Flow

You should use caution when converting to Absolute positioning. Switching back to Flow positioning later may not completely return your report to its former state. When you switch back to Flow positioning from Absolute positioning, Studio will remove *most* of the attributes mentioned above. However, blanket removal is not possible, as you may have added desired attributes, such as Height and Width, manually. So, don't switch to Absolute positioning unless you really, absolutely, have a need to do so.

Using the Layout Editor

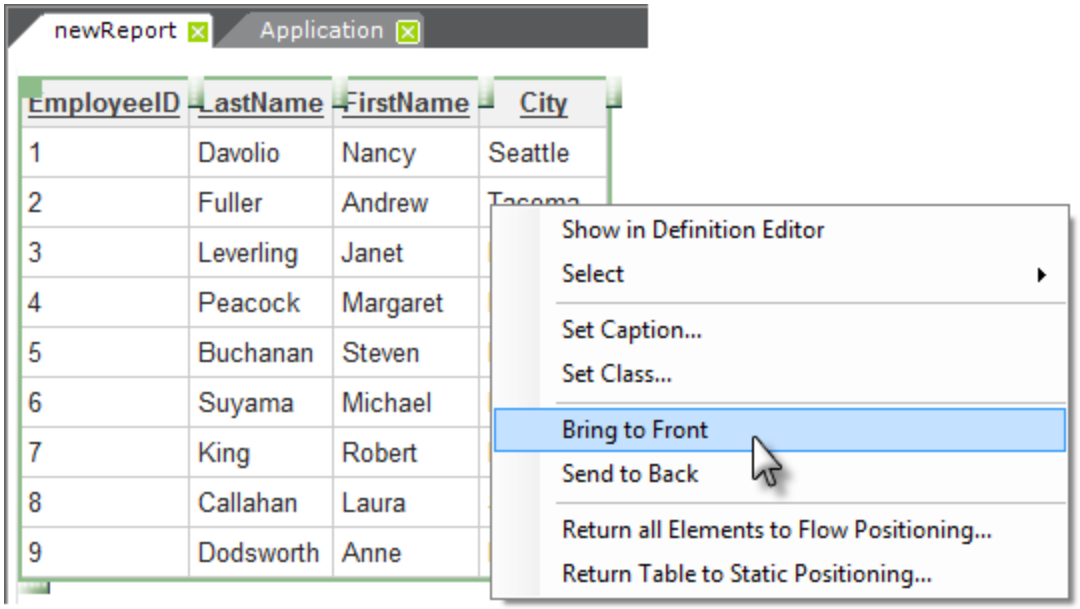
To begin using the Layout editor, click the Layout tab at the bottom of the Workspace panel. The Layout editor displays the report definition as it appears in a browser window.



Selecting certain types of objects in the layout encloses that object in a green outline, as shown above, with one or more "grab points". The table object shown above provides three types of grab points:

1. Click and drag here to reposition the entire table.
2. Click and drag here to resize the object horizontally (in this case, each table column can be resized).
3. Click and drag here to resize the object vertically.

The Layout editor also provides additional functionality beyond absolute positioning and resizing:



A selected object can be right-clicked to view its context menu, as shown above. Options here allow the following operations:

- Move the selected element in front of or behind other elements
- Set the text for Caption attributes
- Apply style sheet classes
- Locate hard-to-find elements by name
- Return the report to Flow Positioning

Menu options are dynamic and will change depending on the nature of the selected element.

Overlaying Elements

Absolute positioning provides a method of allowing one object to overlay another. This gives you the ability, for example, to create a foreground and a background for report definitions.


NorthWind

Product Categories Report

		Description
		Soft drinks, coffees, teas, beers, and ales
		Sweet and savory sauces, relishes, spreads
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

To overlay objects:

1. Position two objects such that one overlaps the other.
2. Select the overlaying (top) object, right-click it, and select **Bring to Front** from the popup menu.

The same results can be achieved with Flow positioning by using the CSS *position: absolute* property and value combination.

Show Modes

Show Modes is a feature in Logi applications that allows elements to be *shown* or *hidden*. This allows report content and appearance to be changed dynamically. Show Modes can be used to format reports for special purposes and even lead to a reduction in the number of report definitions you need.

The following topics discuss the use of Show Modes:

- Using Built-in Show Modes Values
- Setting the Report-Level Default Show Modes
- Overriding Default Show Modes with Action.Report
- Toggling Show Modes with Action.Show Element

About Show Modes

The Show Modes feature allows developers to control to the visibility of elements. This has many uses, from customizing reports based on individual user needs to changing the look of a report when it's exported. For example, if your report includes an "Export" button or link, you may not want that button or link to appear in the actual exported report. Or you may want to add special page headers to reports exported for printing purposes, but you don't need them when the report is viewed in a browser.

Many elements have a **Show Modes** attribute and the value of this attribute determines the element's visibility. If the value is left blank, the element will always be visible.

If they have a value, element Show Modes are compared against a list of values assigned to the entire report (also known as the "Report-level"). Elements with matching values are then handled appropriately (shown or not shown) when the report is displayed.

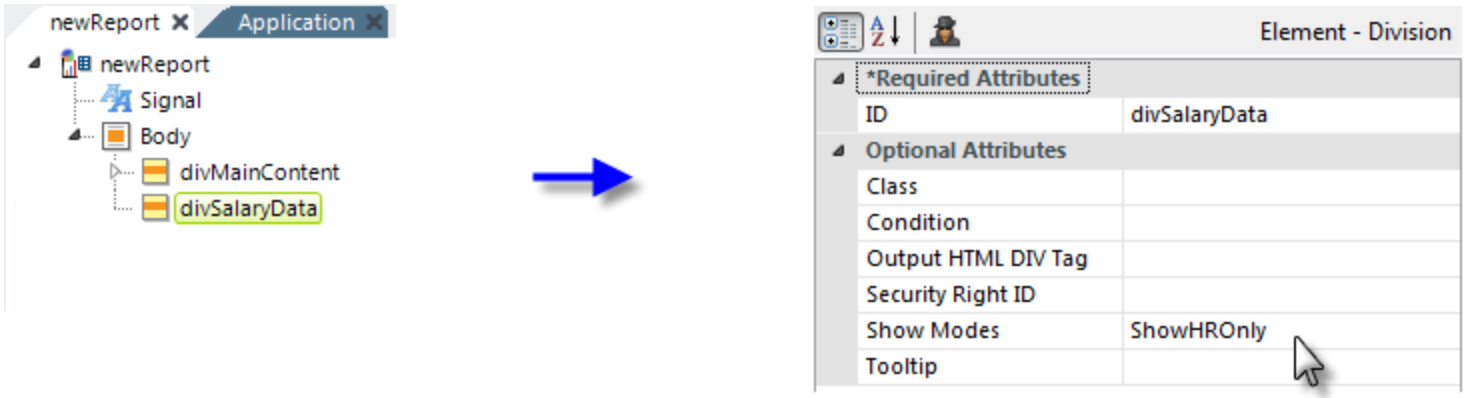
Don't use Show Modes to attempt to secure data. Show Modes controls element visibility by applying CSS at the client, which mean "hidden" information may actually be in the page's HTML source and available to anyone who looks for it.

Standardized *built-in* Show Modes values are automatically included and available at the Report-level. Developers can also provide their own *custom* values at the Report-level and work with them to handle circumstances not covered by the built-in values. Let's take a look at how Show Modes works.

Show Modes in Action

A custom Show Modes value can be any arbitrary string you choose. For example, "Show Documents" or "ExportMode"; anything that makes sense in the circumstances. These values can be defined at the Report-level in several ways, which are discussed separately in later sections.

When selecting a Show Modes value, keep in mind that embedded spaces are allowed and that the comparison between element Show Modes values and Report-level values is *case-sensitive*, so capitalization matters.



The example above presents Show Modes in action. Assuming that the first Division has no value in its Show Modes attribute, it and its contents will *always* be visible. But the second Division, which may display sensitive information, has a Show Modes attribute value of *ShowHROnly*. Therefore it will *only* be visible when *ShowHROnly* has been assigned as a Show Modes value at the Report-level.

Setting or removing this Report-level value dynamically at runtime will make the column appear or disappear. **Tokens** can be used to provide Show Modes attribute values, for both individual elements and at the Report-level, with one exception.

Additional flexibility is also available: *multiple values* can be entered in a Show Modes attribute, separated by commas. Matching *any* one of them with another Show Modes value will cause an element to be visible.

Show Modes vs. Conditions

Logi Info also includes another mechanism for conditionally showing or hiding elements: the **Condition** attribute. Many elements have this attribute and the way it works is that if the attribute value, which is typically an expression, evaluates to *True* or blank, the element is visible. If it's *False*, the element is hidden.

When should you use Show Modes and when should you use Conditions? Here are some factors to consider:

- Some elements have a Show Modes attribute but not a Condition attribute, and vice versa, so your choice will be dictated by the availability of the attributes. Some elements have both.
- For **export** purposes, the built-in Show Modes values for exports clearly offer the easiest configuration.
- Show Modes is a *client-side* effect that uses CSS to hide elements. Conditions, on the other hand, are applied at the *server*; elements hidden using Conditions are not rendered at all and do not appear in the page HTML. So the results of Show Modes are less secure but do not require a return trip to the server to change the page, while Conditions are more secure but their changes require a full or partial refresh of the page from the server.

- Expressions used with Conditions can include scripting functions so they make certain types of comparisons possible, especially those involving math and dates, and they generally use some kind of Data, Request, or Session token. Show Modes, on the other hand, use literal string values, don't generally use tokens, and provide fast string comparisons from a list.
- If you're using Show Modes in a definition with a "super-element", such as the **Analysis Grid** or **Dimension Grid**, you must exercise caution to treat the super-element in an independent manner. Super-elements include their own internal Show Modes that may interfere with, or be interfered with by, Show Modes on the report page. If possible, use Conditions when working with super-elements, instead of Show Modes.
- To show and hide **Data Table Column** elements, we recommend that you use Condition attributes, *not* Show Modes. In Logi Info v12.1+, Data Table Column elements no longer have a Show Modes attribute, though you may encounter it in earlier versions.

You can find out more about Conditions in *Work with Conditions*.

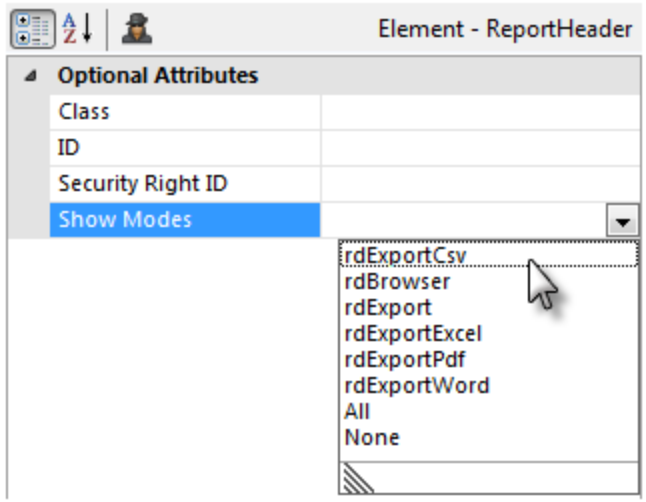
Using Built-in Show Modes Values

Built-in values provide the easiest method of working with Show Modes. They work like switches, automatically including or excluding elements from appearing under certain standard situations. These values are:

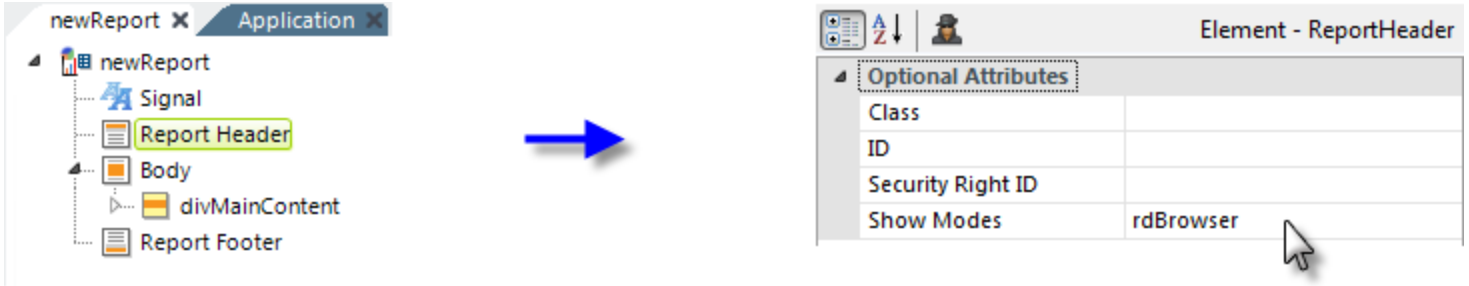
Built-in Value	Description
All	Element will always be visible
None	Element will never be visible
rdBrowser	Element will only be visible when report is viewed in a browser
rdExport	Element will only be visible when report is exported
rdExportCsv	Element will only be included when report data is exported to a CSV file
rdExportExcel	Element will only be included when report is exported to an Excel worksheet
rdExportPdf	Element will only be included when report is exported as a PDF
rdExportWord	Element will only be included when report is exported to a Word document

 When one of these built-in values is used, developers *do not* need to set a corresponding value at the Report-level. When a

specific action occurs, like an export, the application will compare these values to internal system values to determine what action to take.



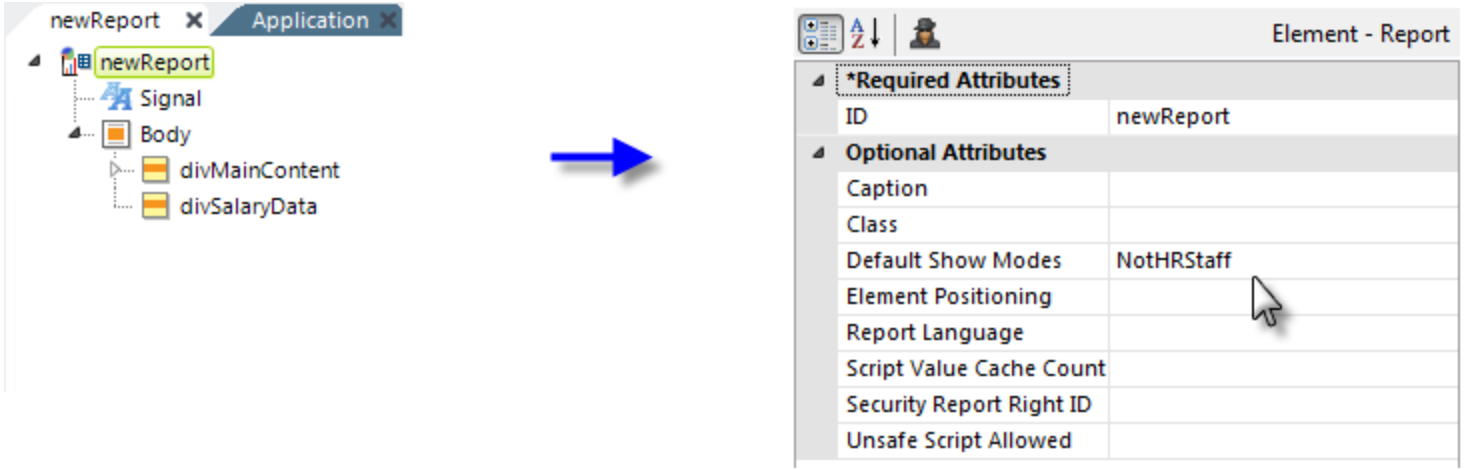
The built-in values are available for selection from a drop-down list in the Show Modes attribute value field, in the Attributes panel, as shown above. Here's an example of one of them in use:



In the example above, in the main content we provide a button that lets the user export the report to PDF. However, we don't want to include the page header and footer in the PDF. By setting the **Report Header** and **Report Footer** elements' Show Modes attributes to the standard *rdBrowser* value, we ensure that they will only appear when the report is browsed, but not when it's exported to PDF.

Setting the Report-Level Default Show Modes

When using *custom* Show Modes values, the most direct way to implement it at the report-level is to set the report's **Default Show Modes** attribute.

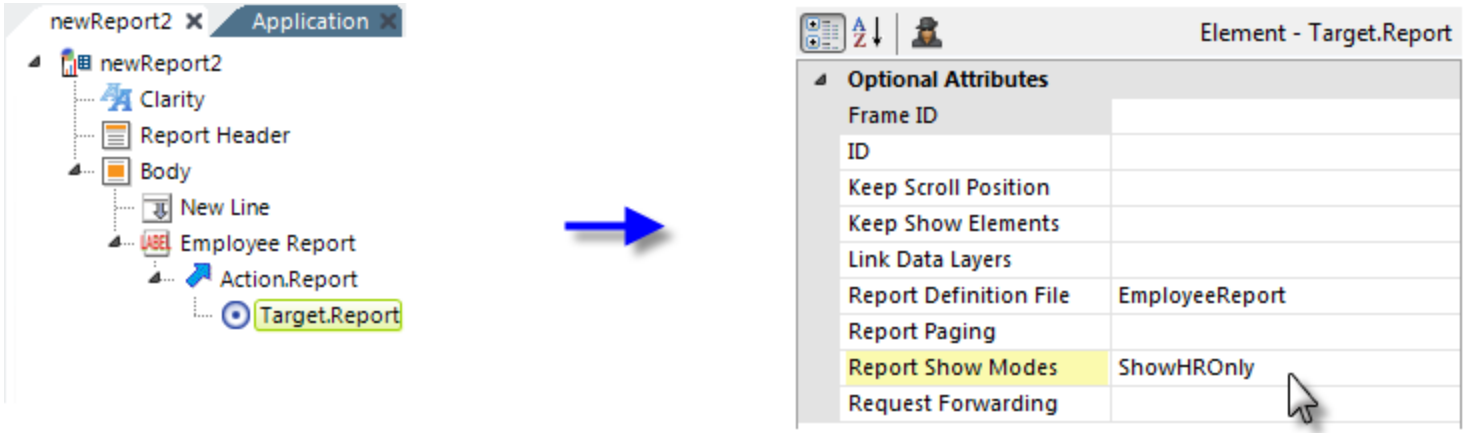


As shown above, the **root** element of a report definition has a **Default Show Modes** attribute. A string literal entered in this attribute's value will affect the rest of the report (as a "Report-level" value). Going back to our earlier example with the three Data Table columns, entering the "NotHRStaff" value here will ensure that the "colSalary" element will be *invisible* by default, because the Report-level Show Modes value and the element Show Modes value do not match.

Now suppose we want to alter the Show Modes (and column visibility) *dynamically*. We can do that with a special query string variable named "rdShowModes". For example, if the URL for this report includes `rdShowModes=ShowHROnly` this will *override* the Default Show Modes attribute's value and in our earlier example colSalarywill become visible. So, by altering the query string, parts of this report can be made visible on the fly.

Overriding Default Show Modes with Action.Report

In "Setting the Report-Level Default Show Modes" on the previous page we learned that use of the *rdShowModes* query string variable will override a report's Default Show Modes setting. The query string variable can be included when calling a report by using a number of methods, including **Link Parameters**. One of the easiest methods is available when using **Action.Report** and **Target.Report** elements to call the next report.



The example above shows a simple **Label** element, used as a link, with Action.Report and Target.Report elements beneath it. If the Target.Report element's **Report Show Modes** attribute is given a value, as shown above, that value will be automatically paired with the rdShowModes keyword and placed in the query string of the URL that calls the target report. The resulting URL would look like this:

```
http://localhost/myLogiApp/rdPage.aspx?rdReport=EmployeeReport&rdShowModes=ShowHROnly
```

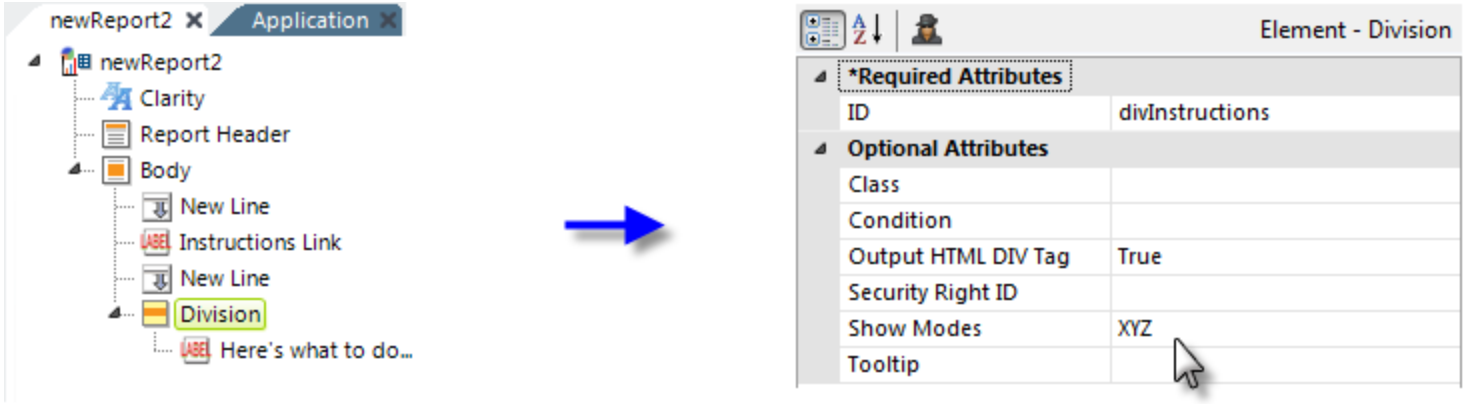
Great runtime flexibility is possible because the Report Show Modes attribute value can be an @Data, @Request, or other token. Multiple comma-separated values, or tokens, can also be entered.

The following elements also have a Report Show Modes attribute that can be used in the same way:

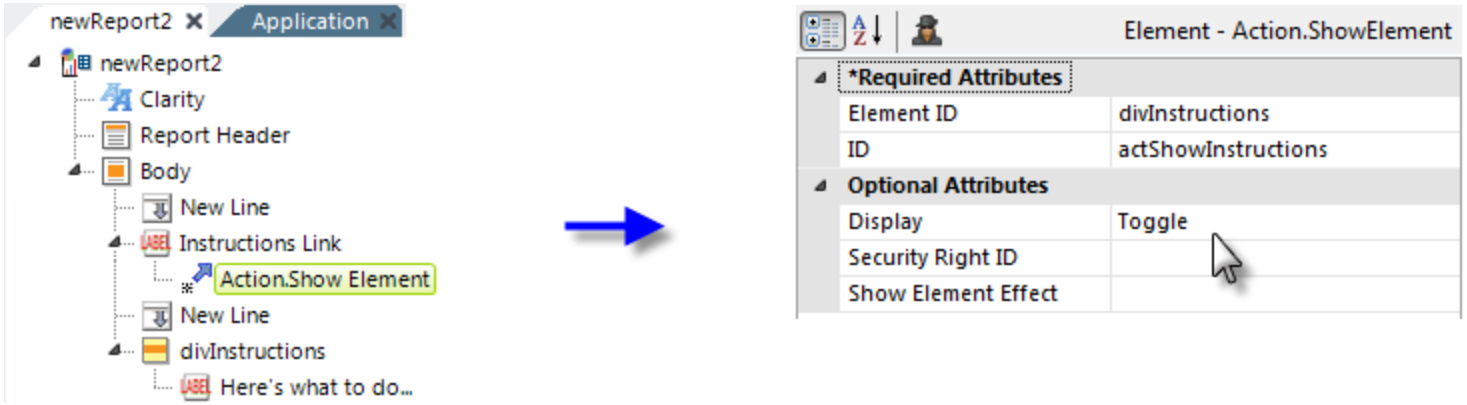
- Action.Refresh Element
- Target.CSV
- Target.NativeExcel
- Target.NativeWord
- Target.PDF
- Target.Widget

Toggling Show Modes with Action.Show Element

You may have a need to allow the user, at runtime, to alternately show and hide ("toggle") a report section. Two elements, the **Division** and **Action.Show Element**, using Show Modes, make this easy to do.



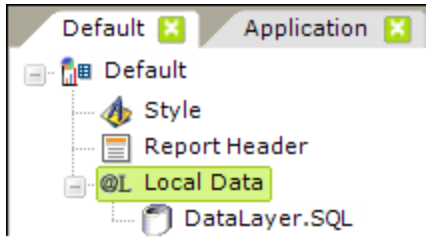
In the example above, a **Label** element has been placed in a report to be the link for showing and hiding instructions, which are in their own **Division**. The Division element's **Show Modes** attribute is set to an arbitrary value, *XYZ*, which causes it to be hidden initially.



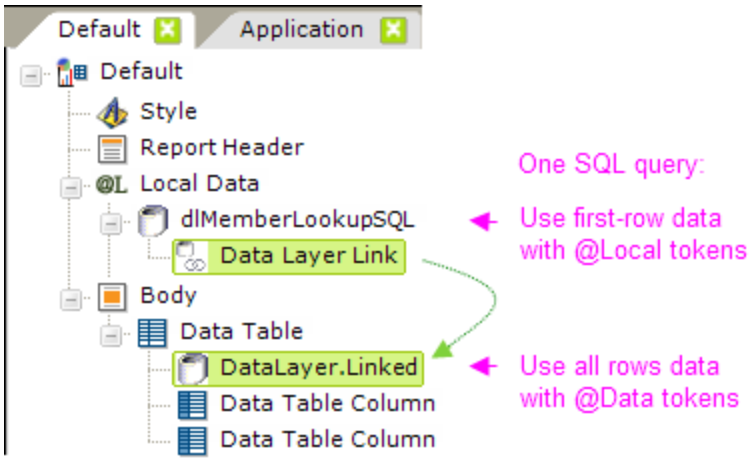
An **Action.Show Element** element has been added beneath the link Label and its attributes configured as shown above. Clicking the link will now result in the Show Modes of the Division element being manipulated, "toggling" it between being shown and being hidden.

Local Data is Available Everywhere

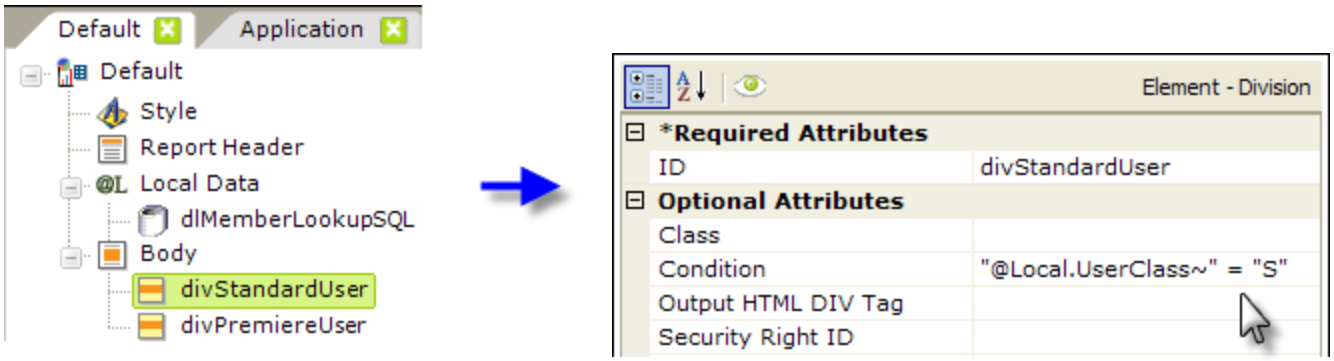
Logi Studio Data Tables are great for retrieving data but have a **scope limitation**: their data is only "available" to their child elements, usually within Data Table columns of some sort. The same applies to charts. However, you may need to use data in other places in your definition that may not be part of a Data Table, chart, or a similar element. This can be accomplished using the **Local Data** element.



The **Local Data** element, shown above, is a container for datalayers and related child data access and manipulation elements. The data in its child datalayer is available, using the **@Local** token, *anywhere* in the report definition. A Local Data datalayer can retrieve any number of rows, however, only the values of the *first* row will be available through the @Local token. This makes Local Data ideal for situations where a database "lookup", returning just a single row, is needed.



However, if you use a **Data Layer Link** element to save the Local Data data, as shown above, and later use a **DataLayer.Linked** element to re-use the data, then *all* of the rows are available for use, using the **@Data** token. The Local Data element has a **Condition** attribute. If the value of this attribute is left blank or contains a formula that evaluates to *True*, then the element's child elements (its datalayers) will run. If the value evaluates to *False*, the element is ignored and the datalayers are not run. This allows developers to dynamically determine when the Local Data should be run, or re-run if a page is refreshed (in this case, a @Request token can be used in the formula in the Condition attribute). Local Data datalayers are *not* re-run with all AJAX refresh requests. They are only re-run when the element being refreshed, using Action.Refresh Element, contains either a DataLayer.Linked element linked to the Local Data datalayer, or when it contains an @Local token. Because @Local tokens can be used *anywhere*, they can be used in many attributes to control element behavior. Here's an example of Local Data in action:



In the example above, Local Data is used to get lookup information about a user. Later in the report definition, that data is used to determine which Division to display. More information about datalayers and data retrieval is available in *Datalayer Introduction*.

Procedure.XML Modifier

The **Procedure.XML Modifier** family of elements allows you to use a Process Task to modify an XML file at runtime.

The following topics discuss the elements and their usage:

- Child Modifier Elements
- Usage Example: Set Attribute
- Usage Example: Append Child Element
- Usage Example: Prepend Child Element
- Usage Example: Set Element Text

About Procedure.XML Modifier

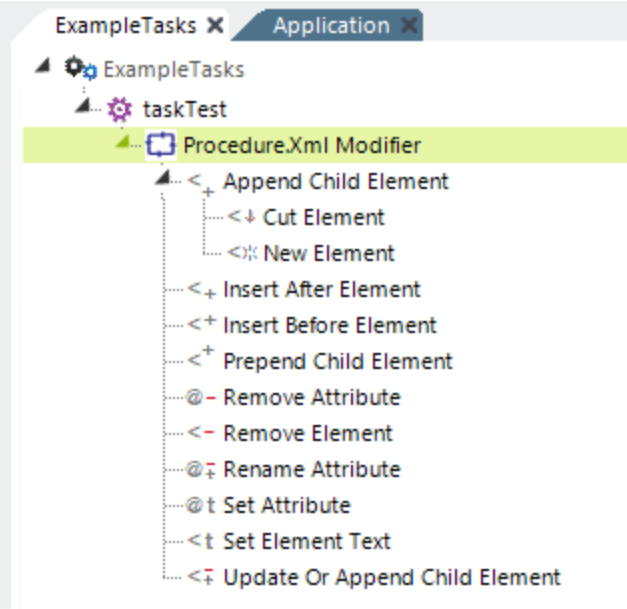
This topic discusses a Logi Info feature that allows you to update an XML file from a Process Task. Basic information about using tasks is available in *Process Tasks*. The techniques and syntax used to do this are similar to those used with a *Definition Modifier Files* (DMF) and reading about them may be helpful. A DMF applies a separate file containing special coded directives to a report definition in order to modify it at runtime. Procedure.XML Modifier operates in a similar manner but the "directives" are created by adding special elements to the Process Task, rather than coding them in a separate file.

 The proper terminology for an XML object is "element", which may cause confusion between the Logi Info *visualelements* in Logi Studio and XML document *elements*. We've attempted to clearly distinguish them in this topic, but please read carefully.
With Procedure.XML Modifier, XML elements and attributes in a file can be added, changed, or removed. "Target" elements can be identified by their ID attribute or by using XPath notation to locate them (learn more about XPath syntax). What kinds of XML files can this procedure element be used to modify? DMFs can modify Logi Info report definitions, which are XML documents, and Procedure.XML Modifier can do so, too, but in a much more programmable fashion. Other XML files, including data files, can also be modified. For example, you might export data to a file using Procedure.Export XML and then want to modify the file in some way

based on subsequent task operations. If you're inserting a new Logi Info element into a definition, you'll need to supply its XML source code. You can get that code by temporarily adding it to a definition, selecting it, configuring it, and copying it. Its XML source code (including that for any child elements) will be copied to your Clipboard and then you can paste it where needed. Don't forget to delete the element if it is indeed temporary. Through the use of Logi Info features such as "Shared Elements" on page 426, Procedure.XML Modifier and its child elements can be reused and even parameterized, if desired.

Procedure.XML Modifier - Child Modifier Elements

Procedure.XML Modifier uses a number of child elements:



All of the elements are shown above and they're described in the following table:

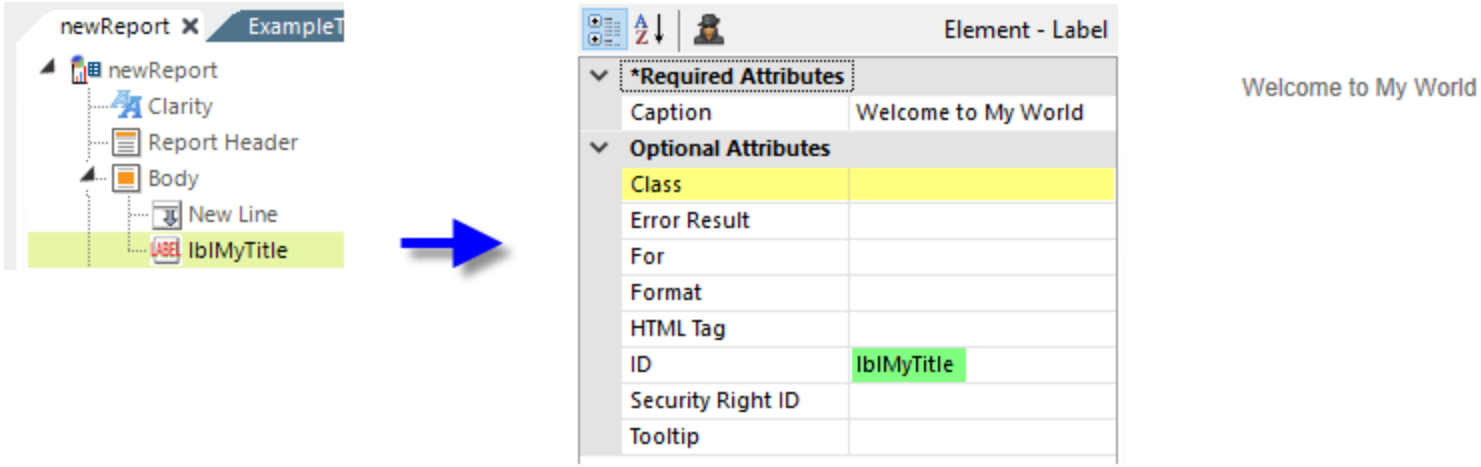
Element	Description
Procedure.XML Modifier	(Required) The parent element for all other related elements. Specifies the fully-qualified path and file name, with extension, of the XML file to be modified. The application must have Full Control or equivalent file access permissions to file. The @Function.AppPhysicalPath~ token can be used here.
Append Child Ele-	Specifies a target XML element by ID or XPath. Appends a new child XML element as the *last* child of the

Element	Description
ment	target element.
• Cut Element	Specifies a target XML element by ID or XPath. Used with any Logi element that appends or inserts a new child XML element, this element deletes the target element first, *before* the append or insert, then makes it available for insertion. This is used primarily to "move" an XML element from one place to another in the file, an operation similar to "Cut and Paste"in a file system. For more information, see this usage example.
• New Element	Used with any Logi element that appends or inserts a new child XML element, this is the "container" for the XML source code that you want to insert. Use Logi Studio's Source tab to directly edit this element, adding the source code for the XML element(s) to be inserted. Not needed when Cut Element is used. For more information, see this usage example.
Insert After Element	Specifies a target XML element by ID or XPath and inserts a new XML element *after* the target element.
Insert Before Element	Specifies a target XML element by ID or XPath and inserts a new XML element *before* the target element.
Prepend Child Element	Specifies a target XML element by ID or XPath and adds new child XML element as the *first* element beneath the target element.
Remove Attribute	Specifies a target XML element by ID or XPath and removes the XML attribute specified by name (required) in the target element.

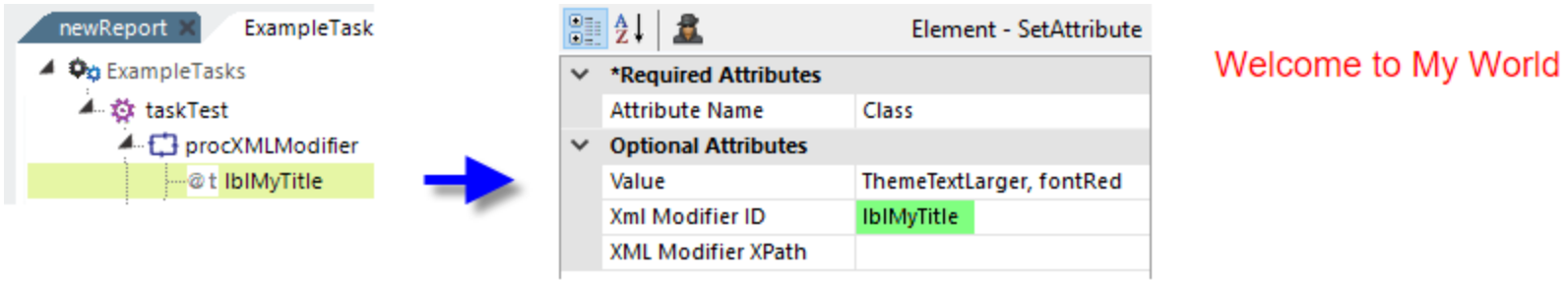
Element	Description
Remove Element	Specifies a target XML element by ID or XPath and removes it.
Rename Attribute	Specifies a target XML element by ID or XPath and renames its specified XML attribute with the specified new name.
Set Attribute	Specifies a target XML element by ID or XPath and replaces its specified XML attribute with the specified value. If the attribute doesn't exist, it's created and set to the value.
Set Element Text	Specifies a target XML element by ID or XPath and specifies its Inner Text value.
Update or Append Child Element	Specifies a target XML element by ID or XPath. If the specified child XML element *doesn*'t exist, it's appended as the *last* element beneath the target element. If the child element *does* exist, the specified attribute values replace the existing values.

Procedure.XML Modifier - Usage Example: Set Attribute

This example demonstrates one of the simplest operations: updating a report definition by changing an element's Class attribute value.



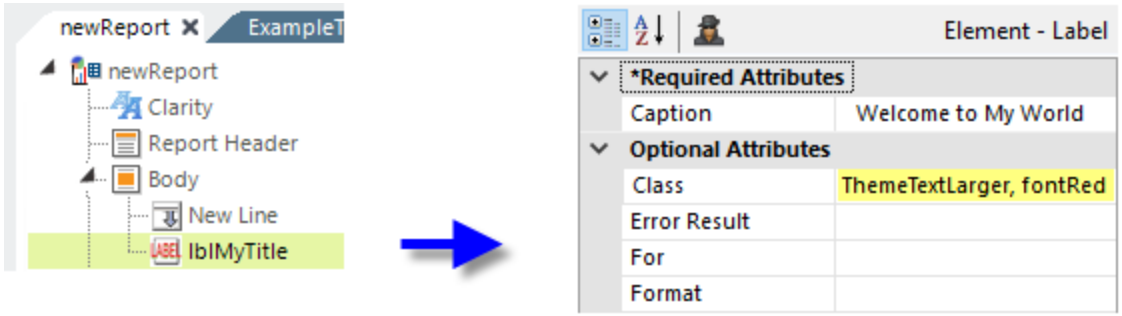
In the image above, we see a report definition on the left, the attributes for its **Label** element in the center (note that there is no Class attribute value), and the resulting output on the right.



And now we see the Process Task with **Procedure.XML Modifier** and **Set Attribute** elements above on the left. The Procedure.XML Modifier element's Filename attribute (not shown) has been set to:

```
@Function.AppPhysicalPath~\_Definitions\_Reports\newReport.lgx
```

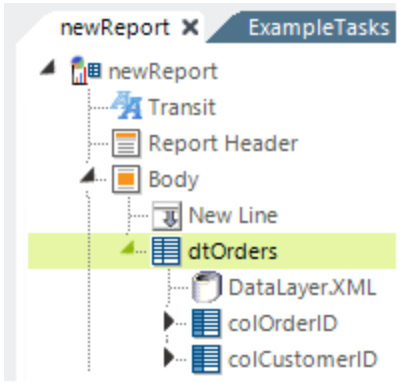
The Set Attribute element attributes are shown above in the center (note that the Label element's ID is used for the Xml Modifier ID value), and the resulting output is on the right.



If we refresh the report definition in Studio, we can see that it's been updated with the new Class attribute value, shown highlighted above. To change multiple attributes for the same element, use multiple child Set Attribute elements in the Process Task.

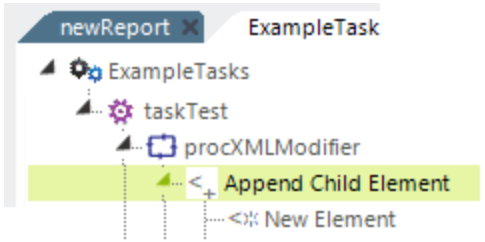
Procedure.XML Modifier - Usage Example: Append Child Element

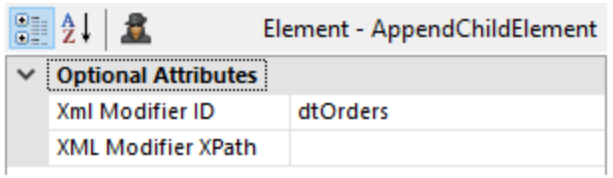
This example demonstrates appending a child XML element: we'll update a report definition to add a new column to a Data Table.



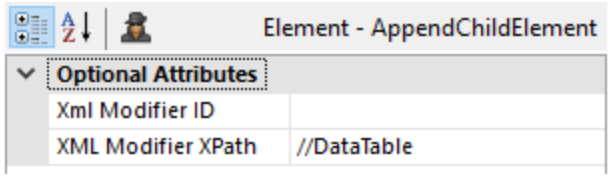
Order ID	Customer ID
10248	VINET
10249	TOMSP
10250	HANAR
10251	VICTE
10252	SUPRD

As shown above, our report definition has a Data Table ("dtOrders") and it currently has two columns.





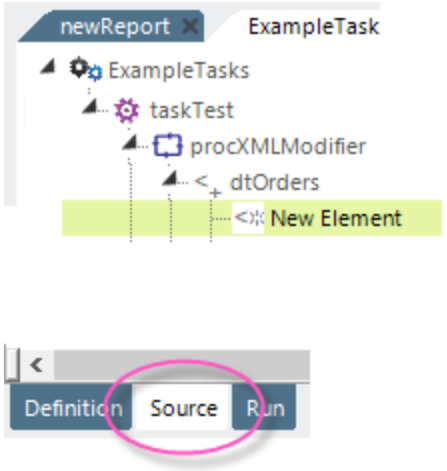
- or -



XPath is case-sensitive!

And now we see the Process Task with **Procedure.XML Modifier** and **Append Child Element** elements above. The Procedure.XML Modifier element's Filename attribute has been set as shown in the previous example. Two methods of using the Append Child Element element's attributes to identify the target Logi Info element (the Data Table) are shown.

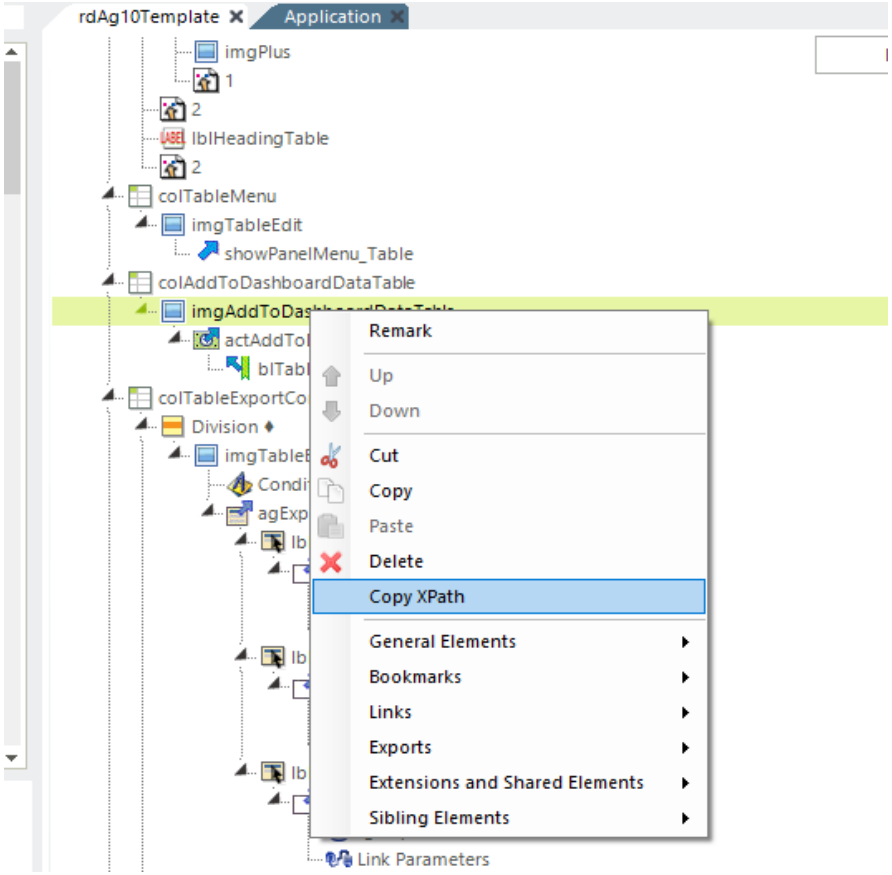
The XPath notation shown above will find the *first* Data Table in the definition.



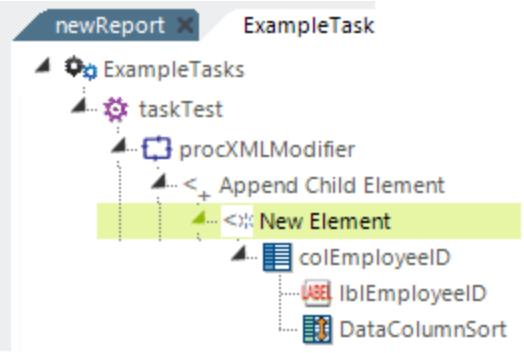

```
<AppendChildElement ID="dtOrders">
  <NewElement>
    <DataTableColumn ID="colEmployeeID" Header="Employee ID">
      <Label ID="lblEmployeeID" Caption="@Data.EmployeeID~" />
      <DataColumnSort DataColumn="EmployeeID" DataType="Number" />
    </DataTableColumn>
  </NewElement>
</AppendChildElement>
```

Next we need to manually enter or paste the source code for a Data Table Column into the New Element element, as shown above. Select the element and use the Source tab at the bottom of the editor in Studio to see its source code. Enter or paste your code as shown. If you only see a <NewElement /> tag, you'll also need to edit it into two tags, as highlighted in yellow above. Where can you get the source code to paste into the NewElement tag? You can select any element in Studio and copy it, then paste it into any text editor (or Studio's Source tab) and you'll get its complete XML source code, including its child elements. For example, you can select and copy one of the existing Data Table Column elements in the report definition, paste its code, and then edit it as needed.

In Studio, you can also find the XPath of an element in a report or template by right-clicking on the element and selecting **Copy XPath**:

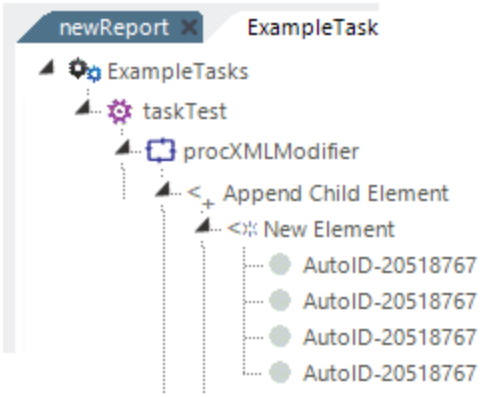


Selecting this option will copy the direct XPath of that element, the tag name, and ID attribute (if available) to the Clipboard, which can then be copied into a TMF.

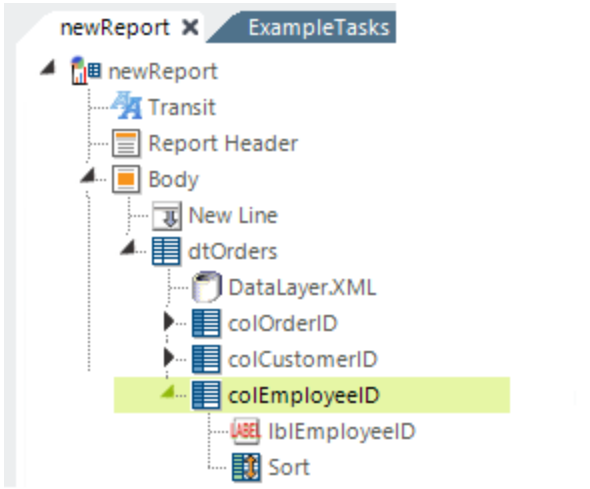


When you switch back to the Definition tab in Studio, you'll see the added Logi elements appear beneath the New Element element, as shown above.

```
<AppendChildElement ID="dtOrders">
  <NewElement>
      <AutoID-20518767 UnitPrice="14.0000" />
      <AutoID-20518767 UnitPrice="9.8000" />
      <AutoID-20518767 UnitPrice="34.8000" />
      <AutoID-20518767 UnitPrice="18.6000" />
  </NewElement>
</AppendChildElement>
```



If we were updating an XML *data* file, instead of a report definition, the resulting elements would look something like the above in Studio.

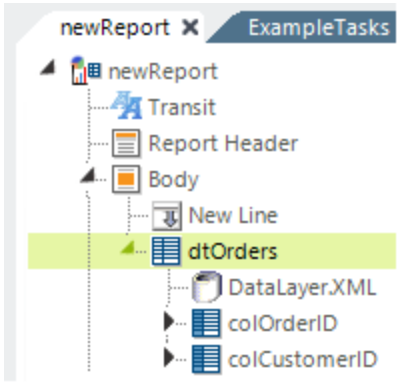


Order ID	Customer ID	Employee ID
10248	VINET	5
10249	TOMSP	6
10250	HANAR	4
10251	VICTE	3
10252	SUPRD	4

After the task runs, we can refresh the report definition in Studio, we can see that it's been updated to include the new Data Table column.

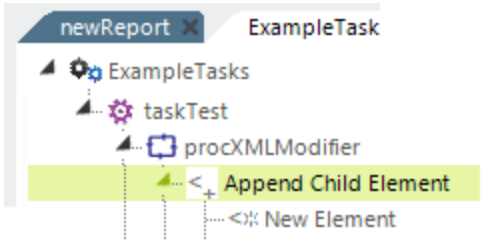
Procedure.XML Modifier - Usage Example: Prepend Child Element

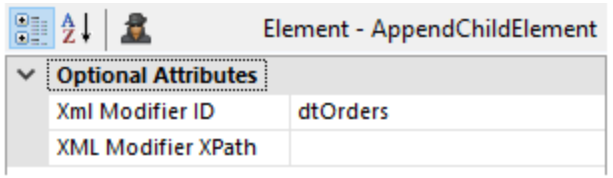
This example demonstrates appending a child XML element: we'll update a report definition to add a new column to a Data Table.



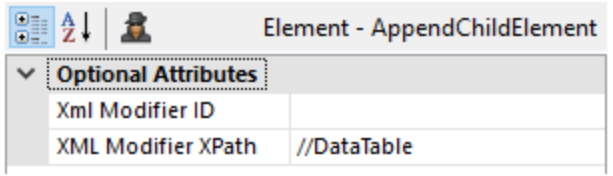
Order ID	Customer ID
10248	VINET
10249	TOMSP
10250	HANAR
10251	VICTE
10252	SUPRD

As shown above, our report definition has a Data Table ("dtOrders") and it currently has two columns.





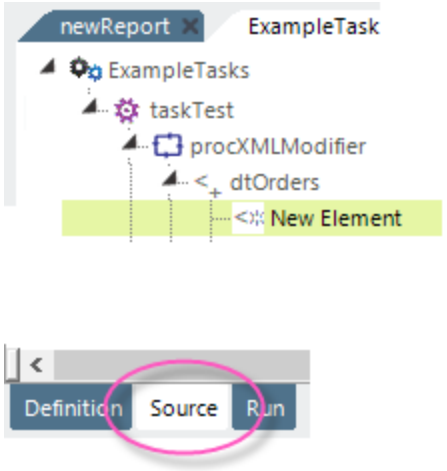
- or -



XPath is case-sensitive!

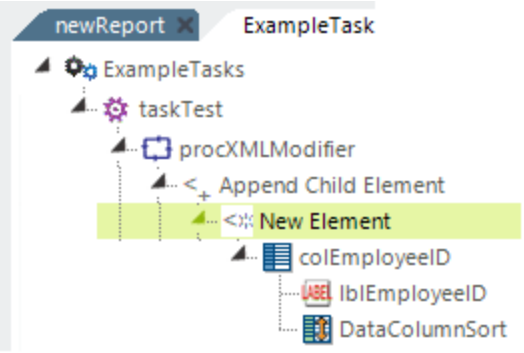
And now we see the Process Task with **Procedure.XML Modifier** and **Append Child Element** elements above. The Procedure.XML Modifier element's Filename attribute has been set as shown in the previous example. Two methods of using the Append Child Element element's attributes to identify the target Logi Info element (the Data Table) are shown.

The XPath notation shown above will find the *first* Data Table in the definition.



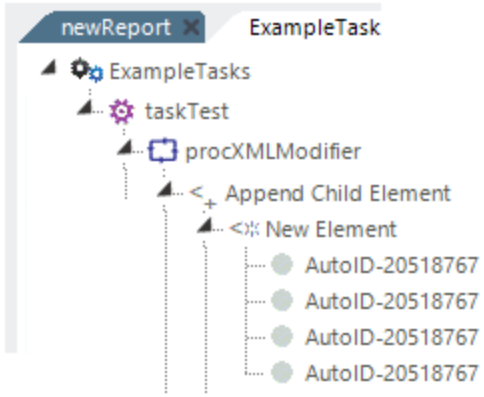

```
<AppendChildElement ID="dtOrders">
  <NewElement>
    <DataTableColumn ID="colEmployeeID" Header="Employee ID">
      <Label ID="lblEmployeeID" Caption="@Data.EmployeeID~" />
      <DataColumnSort DataColumn="EmployeeID" DataType="Number" />
    </DataTableColumn>
  </NewElement>
</AppendChildElement>
```

Next we need to manually enter or paste the source code for a Data Table Column into the New Element element, as shown above. Select the element and use the Source tab at the bottom of the editor in Studio to see its source code. Enter or paste your code as shown. If you only see a <NewElement /> tag, you'll also need to edit it into two tags, as highlighted in yellow above. Where can you get the source code to paste into the NewElement tag? You can select any element in Studio and copy it, then paste it into any text editor (or Studio's Source tab) and you'll get its complete XML source code, including its child elements. For example, you can select and copy one of the existing Data Table Column elements in the report definition, paste its code, and then edit it as needed.

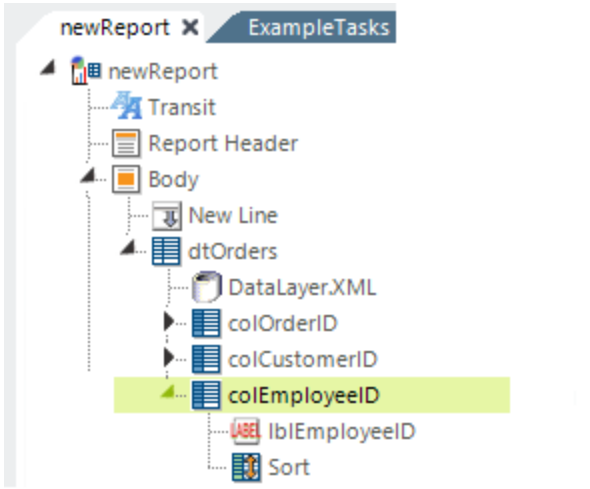


When you switch back to the Definition tab in Studio, you'll see the added Logi elements appear beneath the New Element element, as shown above.

```
<AppendChildElement ID="dtOrders">
  <NewElement>
      <AutoID-20518767 UnitPrice="14.0000" />
      <AutoID-20518767 UnitPrice="9.8000" />
      <AutoID-20518767 UnitPrice="34.8000" />
      <AutoID-20518767 UnitPrice="18.6000" />
  </NewElement>
</AppendChildElement>
```



If we were updating an XML *data* file, instead of a report definition, the resulting elements would look something like the above in Studio.



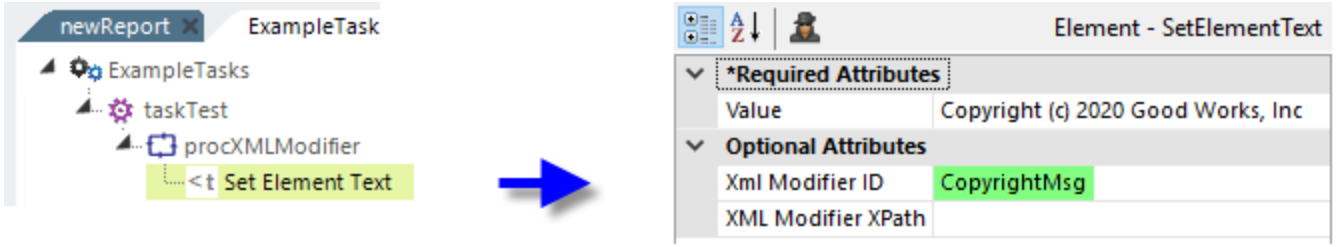
Order ID	Customer ID	Employee ID
10248	VINET	5
10249	TOMSP	6
10250	HANAR	4
10251	VICTE	3
10252	SUPRD	4

After the task runs, we can refresh the report definition in Studio, we can see that it's been updated to include the new Data Table column.

Procedure.XML Modifier - Usage Example: Set Element Text

This example demonstrates how to set the text for HTML tags that support text.

Now is time for all good men to come to the aid of their country
Copyright (c) 2018 Good Works, Inc.
Assume we have an HTML page with the code shown above in it, and we'd like to programmatically change the year in the copyright declaration.



In our Process Task, shown above, the Procedure.XML Modifier element's Filename attribute (not shown) has been set to the path and filename for the XML data file:

```
@Function.AppPhysicalPath~\_SupportFiles\PageFragment.html
```

The **Set Element Text** element's Xml Modifier ID attribute is set to the target node's ID and its value is set to the new text to be inserted or that will replace any existing text. Color highlighting shows the relationship between the data and the attribute values.

Now is time for all good men to come to the aid of their country
Copyright (c) 2020 Good Works, Inc.
After the task runs, we can look at the HTML file once again and see that the text has been changed, as shown above.

Data Calendar

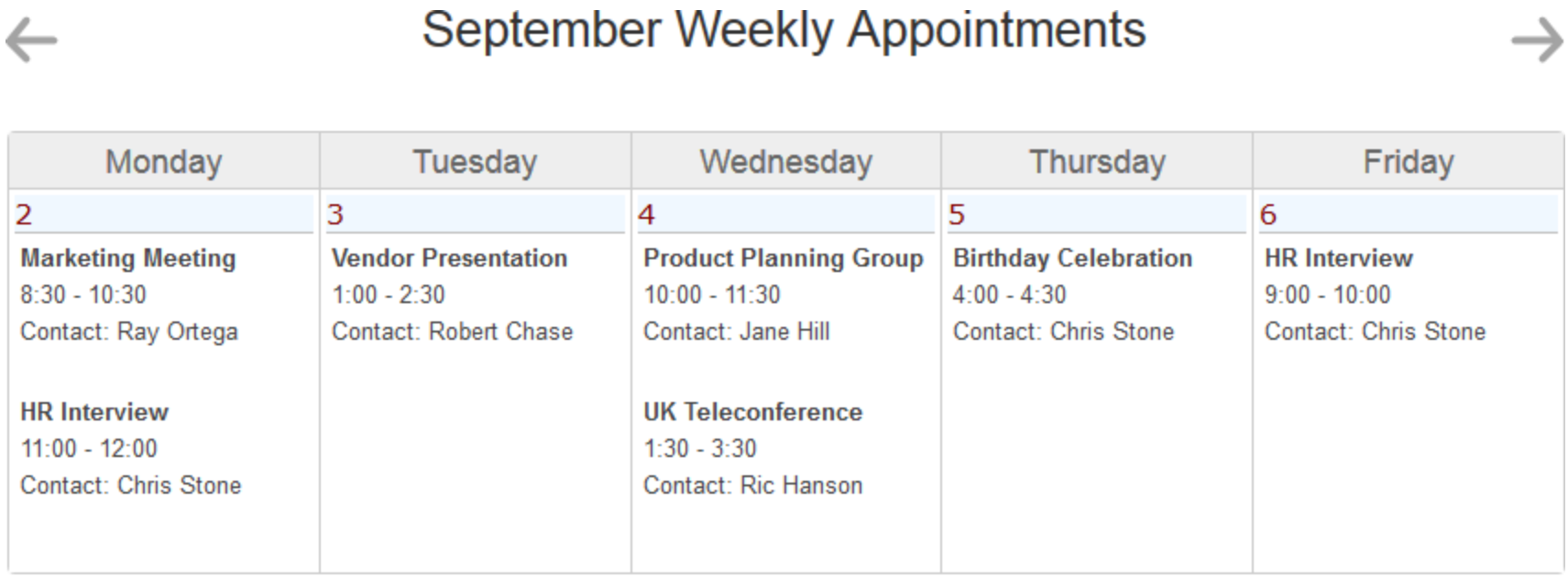
The **Data Calendar** element allows Logi Info developers to create data-driven calendars within reports. These can be weekly or monthly calendars, with a variety of customizable days and appearances.

The following topics discuss the use of the Data Calendar element:

- Data Calendar Attributes
- Working with the Data Calendar
- Globalization and Localization
- Overriding the Default Style Classes

About the Data Calendar Element

The Data Calendar element, like a Data Table or a chart, works with adatalayer element to retrieve data from a data source. Date-type data in the datalayer is used to generate the calendar days, which can then be populated with labels, links, tables, charts, etc. based on other columns in the data. Calendars can be *weekly* or *monthly*, with a variable number of days displayed; for example, a weekly calendar can be configured to show only the weekdays.



A calendar is rendered by the Logi Server engine as an HTML table; it's not an image. Two child elements, **Data Calendar Day** and **Data Calendar Rows**, are used to contain the data presented within each day in a calendar. A third, **Data Calendar Paging**, provides automatic next and previous calendar period paging.

The general appearance of a calendar is controlled by style sheet classes, which can be overridden in a custom style sheet.

Data Calendar Attributes

The Data Calendar element has the following attributes:

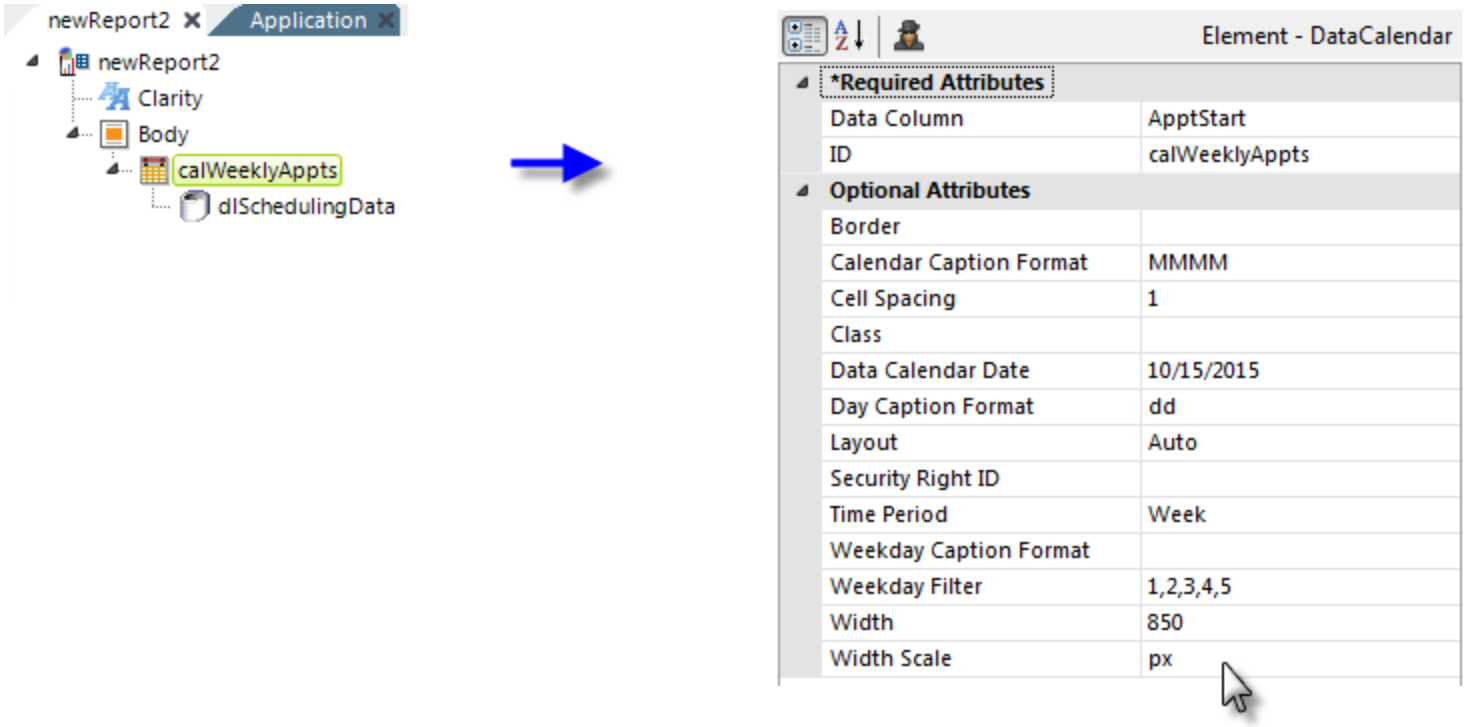
Attribute	Description
Data Column	(Required) Specifies the name of a column in the datalayer that contains *date-type* information. The dates in this column will each correspond to a single day in the calendar. This data should be in YYYY-MM-DD format. (As long as the data source thinks the data is date-type date, it will be retrieved into the datalayer in this format. Otherwise, a Calculated Column element should be used to reformat the data into the required format.)
ID	(Required) Specifies a unique element ID.
Border	Specifies the thickness of the border, in pixels, around the calendar and between each day on the calendar. Default (and minimum) value: *1*
Calendar Caption Format	By default, a caption will appear centered above the calendar consisting of a date range (weekly calendar) or the month and year (monthly calendar). This attribute specifies a format for that caption, using system formats such as *Short Date* (weekly) or these combinations (monthly): *None* - the caption is suppressed *yy* - two-digit year *yyyy* - four-digit year *MM* - month number *MMM* -three-letter month abbreviation (Jan, Feb, Mar, etc.) *MMMM* - full month name *Note: Only date-related formats should be selected here; use of other formats results in the format option itself appearing in the calendar caption.*Default: *Short Date* (weekly calendar); *yyyy MMMM* (monthly calendar)

Attribute	Description
Cell Spacing	Specifies the space, in pixels, between cells (boxes) in the calendar. Default: *0*
Class	Specifies the CSS class(es) to be used by the element. Generally not used if a Theme is being used.
Data Calendar Date	Specifies a date value that sets the current date for the calendar. The calendar will display the week or month, based on the Time Period attribute, containing the current date.
Day Caption Format	Controls the appearance of the day number in each box of the calendar. Set to "None" to suppress the numbering. Default: *%d* which provides the day number.
Layout	Determines whether the columns of days within the calendar size their widths automatically based on the widest data they contain or are set to a fixed width (based on required width of first row). Default: *Auto*
Security Right ID	When using Logi Security, specifies one or more Security Right IDs, separated by commas. Users with these Security Right IDs will be able to view the Analysis Grid, users without them will not.
Time Period	Specifies whether the calendar rendered will display a week or a month of days. Default: *Month*
Weekday Caption Format	This attribute is used to format the Monday, Tuesday, Wednesday, etc. headings that appear at the top of the calendar above each columns of days. These formats are available: *ddd* - three-letter day abbreviation (Mon, Tues, Weds, etc.) *dddd* - full day name *None* - the caption is suppressed Default - *dddd*

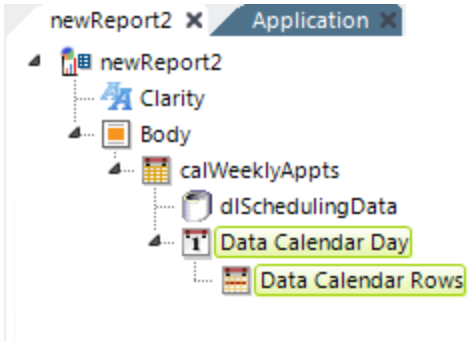
Attribute	Description
Weekday Filter	Specifies which days of the week will appear in the calendar; useful for suppressing display of weekend days. Enter values 0-6, separated by commas. Typically, a value of 1,2,3,4,5 will produce a calendar without Sunday (0) or Saturday (6) in it. The relationship of numbers 0-6 to days of the week can be set differently when using the Globalization element in your _settings definition by configuring its FirstDayOfWeek attribute. Default: *0,1,2,3,4,5,6* - showing all days of the week
Width Width Scale	Specifies the width of the Analysis Grid, in units set in the Width Scale attribute (either *px* for pixels or % for percent).

Working with the Data Calendar

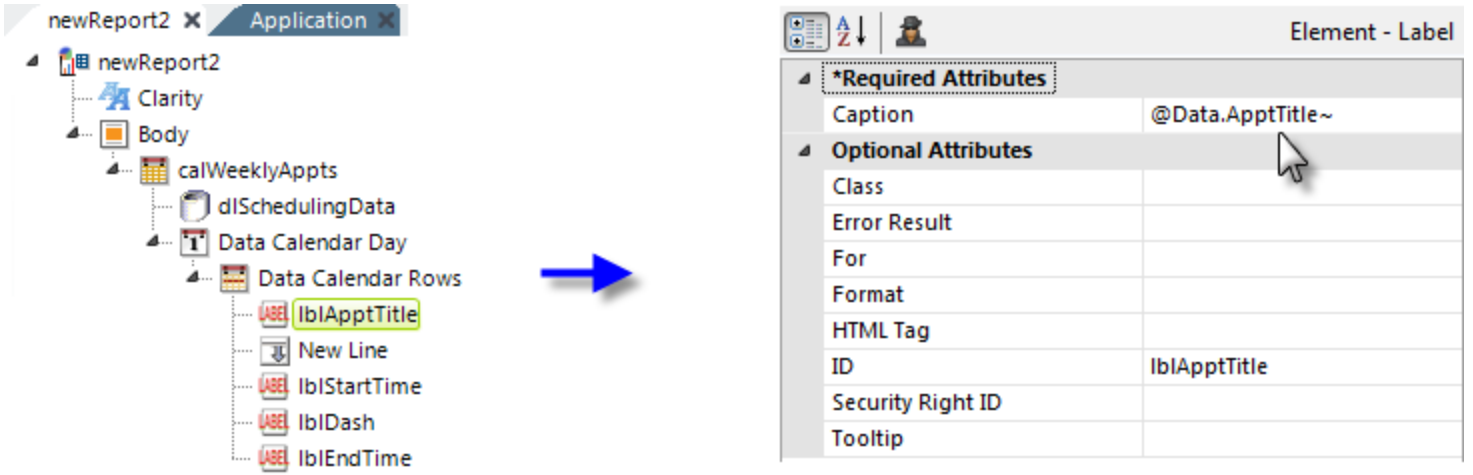
The following provides a step-by-step example of how to use the Data Calendar element to create a calendar:



1. Add a **Data Calendar** element to the body of your report definition and a child datalayer, as shown above, to retrieve the scheduling data.
2. Set its attributes as shown above, right. This will produce a calendar similar to the image in "Data Calendar Attributes" on page 85.



3. Next, as shown above, add a **Data Calendar Day** element as a child of the Data Calendar element. This element has no attributes to be set. Beneath it, add a **Data Calendar Rows** element, which also has no attributes to be set.



4. Finally, add **Label** and other elements as needed beneath the Data Calendar Rows element to display the data for each day on the calendar, using standard @Data tokens. A variety of elements, including images and charts, can also be used here to display data and Action elements can be used with them to create links, if desired.

The overall appearance of the calendar can be controlled by including a **style sheet** in this report and using the techniques described in "Overriding the Default Style Class" on page 93.

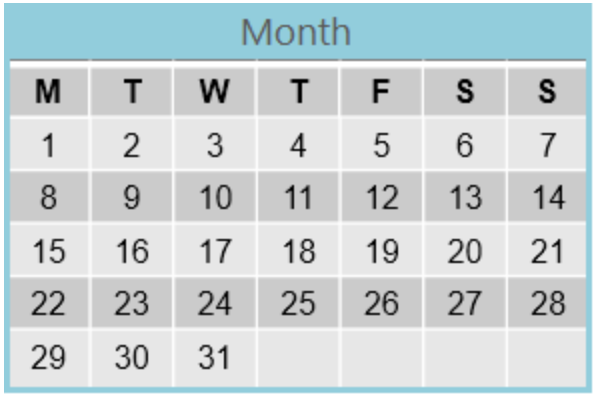
Globalization and Localization

The format and presentation of calendars and dates varies widely depending on country and/or language. Some aspects of the date-related input elements are controlled by the "preferred language" settings of the **browser** being used and others may be affected by the default date and time format settings of the **operating system**.

At the application level, date-related aspects of your Logi application can also be changed by adding and configuring a **Globalization** element in the application's _Settings definition. More information about it use can be found in *Internationalization and Localization* .

For example, setting the Globalization element's **First Day of Week** attribute will change the display of the calendars. If the attribute value is set to *1*, then *Monday* will be the first day in the calendar (the default is *Sunday*).

In addition, the browser's preferred **language setting** will affect the display of the month and day-of-week names:



Browser Language: en-us



Browser Language: fr-FR

In the examples shown above, a Globalization element has been used to change the first day of the week to Monday. Notice that the day-of-week **abbreviations** in the calendar also change based on the browser's preferred language setting.

In addition, the actual value returned after the user clicks on a date also reflects the language preference in any text it contains - "Jan" vs. "janv", for example.

Overriding the Default Style Class

The Data Calendar element is normally rendered at runtime using a default style sheet, which is:

```
<your application folder>\rdTemplate\rdCalendar\rdDataCalendarStyle.css
```

However, you can **override** the classes in this style sheet by copying them to your application's style sheet and modifying them. *Do not modify the original style sheet named above, as it will be overwritten and your changes lost if you upgrade to a new Logi Info version.*

```
.rdDataCalendarCaption
{
text-align: center;
background-color: white;
font-weight: bold;
font-size: 10pt;
padding-top: 10px;
padding-bottom: 10px;
}
```

For example, adding the style class shown above to your application style sheet will allow you to control the appearance of the calendar caption that's generated at the top of calendar,

```
.rdDataCalendarDay
{
vertical-align: top;
```

```
 background-color:khaki;
width: 20%;
height: 100px; /* default for any day's box */
}
```

while the one shown above controls the appearance of the boxes on the calendar that represent each day. There are classes that allow you to control all aspects of the calendar, including background colors and images, border colors, and captions.

The Aggregate Column

An Aggregate Column element adds a new column to the datalayer that represents an aggregated value of the data in one of the other columns in the datalayer.

The following topics discuss the Aggregate Column element:

- Aggregate Column Attributes
- Aggregate Usage Example

About the Aggregate Column

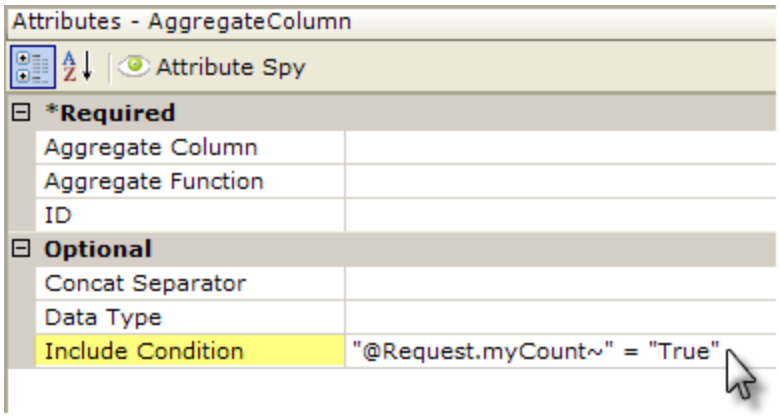
The Aggregate Column is similar to other elements designed to **extend** the data available in a datalayer. Like the **Calculated Column** element, the Aggregate Column is added as a child element beneath a datalayer and adds a **new column** to the datalayer. This column contains **derived data** which is accessible using an @Data token.

The new column's value is the result of aggregating the value of the data in one column, for all rows, in the datalayer. The aggregated value is populated in the new column in every row of the datalayer so that it can be used in further calculations, Summary Rows, Link Params, etc. It is also populated in the top-level row, allowing reference to the value anywhere in the report outside of a Data Table.

Similar aggregations can be performed within SQL queries and may offer faster performance when done by the database server. However, the Aggregate Column element allows aggregations to be performed on data from other sources too, such as web services and data files, that do not themselves offer aggregating functions.

Dynamic Application

The Aggregate Column element has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows developers to dynamically determine if the Aggregate Column will be added to the datalayer or not.

Aggregate Column Attributes

The attributes for the Aggregate Column element are described below:

Attribute	Description
Aggregate Column	(Required) The name of the column in the datalayer that contains the values to be aggregated.
Aggregate Function	(Required) The aggregating function to be applied to the column named in the previous attribute. The available functions include: • Average • Concat (see section below) • Count • Distinct Count • Max • Median • Min • Mode • Std Deviation • Sum Columns with Null values are *excluded* by default. If you want them to be included, create the constant `rdCalculationsIncludeNulls` and set it to *True.*
ID	(Required) The unique element ID for the Aggregate Column element.

Attribute	Description
Concat Separator	The character used to separate concatenated text values, if any. Default: *comma*
Data Type	The data type of the column named in the Aggregate Column attribute value.
Include Condition	Determines if the aggregation will be applied. If left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer.

The Concat Function

The Concat function builds a string of concatenated values by taking all the text in the specified data column and concatenating it together using the default (a comma) or specified separator character.

PoetName	Category	BirthYear
Whitman	102	1819
Plath	105	1932
Dickinson	103	1830
Frost	107	1874

Original Data

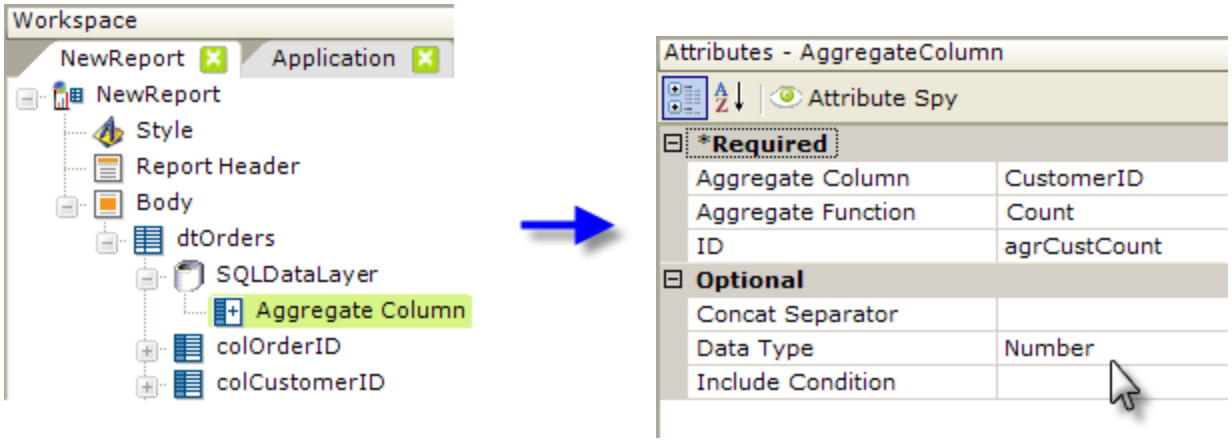
PoetName	Category	BirthYear	agrConcat
Whitman	102	1819	Whitman,Plath,Dickinson,Frost
Plath	105	1932	Whitman,Plath,Dickinson,Frost
Dickinson	103	1830	Whitman,Plath,Dickinson,Frost
Frost	107	1874	Whitman,Plath,Dickinson,Frost

with Aggregate Column, using Concat function

This can be useful for a variety of purposes, including creating default values for select lists.

Aggregate Column Usage Example

The following example, which provides a count of all customers, illustrates how the Aggregate Column works:



In the definition shown above, a Data Table and a datalayer element have been added. An **Aggregate Column** element has been added and its attributes have been configured so that it will count the number of customer IDs in the data retrieved into the datalayer.

OrderID	CustID	Ordered
10248	VINET	7/4/1996
10249	TOMSP	7/4/1996
10250	HANAR	7/8/1996
10251	VICTE	7/8/1996
10252	SUPRD	7/9/1996
10253	HANAR	7/10/1996
10254	CHOPS	6/3/1996
10255	RICSU	7/12/1996
10256	WELLI	7/15/1996
10257	HILAA	7/16/1996

Original Data

OrderID	CustID	Ordered	agrCustCount
10248	VINET	7/4/1996	10
10249	TOMSP	7/4/1996	10
10250	HANAR	7/8/1996	10
10251	VICTE	7/8/1996	10
10252	SUPRD	7/9/1996	10
10253	HANAR	7/10/1996	10
10254	CHOPS	6/3/1996	10
10255	RICSU	7/12/1996	10
10256	WELLI	7/15/1996	10
10257	HILAA	7/16/1996	10

with Aggregate Column, using Count function

The resulting count will be placed in a new column, named agrCustCount, in every row of the datalayer, and will be accessible as the token @Data.agrCustCount~. Because the value is in every row, it's available for use with every row of data in the table.

The DsAggregate Column

The **DsAggregate Column** element is used with DataLayer.Data Services to add a new column to the data containing an aggregated value of the data in one of the other columns.

The following topics discuss the DsAggregate Column elements:

- DsAggregate Column Attributes
- DsAggregate Column Usage Example

For more information about Dataviews, see *Dataviews*.

About the DsAggregate Column

The DsAggregate Column element discussed here is available in Logi Info v12.5 but has been deprecated in later Info versions; consult the Release Notes for specific details.

The **DsAggregate Column** is similar to other elements designed to *extend* the available data. Like the DsCalculated Column element, it's added as a child element beneath DataLayer.Data Services and adds a **new column** to the data. This column contains derived data which is accessible using an @Data token.

The new column's value is the result of aggregating the value of the data in one column, for all rows, in the data. The aggregated value is populated in the new column in every row of the data so that it can be used in further calculations and displays.

Similar aggregations can be performed in a Dataview definition but this element allows dynamic aggregations based on variable conditions at runtime.

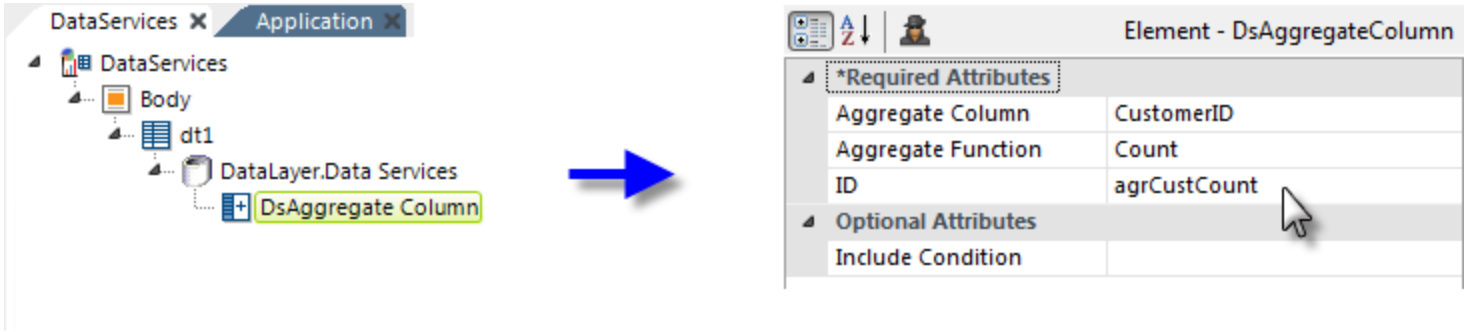
DsAggregate Column Attributes

The attributes for the DsAggregate Column element are described below:

Attribute	Description
Aggregate Column	(Required) Specifies the name of the column in the datalayer that contains the values to be aggregated.
Aggregate Function	(Required) Specifies the aggregating function to be applied to the column named in the previous attribute. The available functions include: • Avg • Count • Distinct • Max • Min • Sum
ID	(Required) Specifies a unique ID for the element.
Include Condition	Specifies if the aggregation will be applied. If left blank or contains a formula that evaluates to *True*, the element is applied to the data. If the value evaluates to *False*, the element is ignored and does not affect the data.

DsAggregate Column Usage Example

The following example, which provides a count of all customers, illustrates how the DsAggregate Column works:



In the example shown above, a Data Table and a DataLayer.Data Services element have been added. A **DsAggregate Column** element has been added beneath the datalayer and its attributes have been configured so that it will count the number of customer IDs in the data.

OrderID	CustID	Ordered
10248	VINET	7/4/1996
10249	TOMSP	7/4/1996
10250	HANAR	7/8/1996
10251	VICTE	7/8/1996
10252	SUPRD	7/9/1996
10253	HANAR	7/10/1996
10254	CHOPS	6/3/1996
10255	RICSU	7/12/1996
10256	WELLI	7/15/1996
10257	HILAA	7/16/1996

Original Data

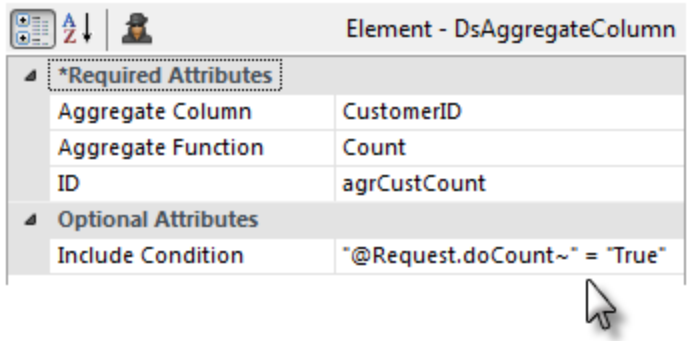
OrderID	CustID	Ordered	agrCustCount
10248	VINET	7/4/1996	10
10249	TOMSP	7/4/1996	10
10250	HANAR	7/8/1996	10
10251	VICTE	7/8/1996	10
10252	SUPRD	7/9/1996	10
10253	HANAR	7/10/1996	10
10254	CHOPS	6/3/1996	10
10255	RICSU	7/12/1996	10
10256	WELLI	7/15/1996	10
10257	HILAA	7/16/1996	10

Data after DsAggregate Column applied, using Count function

The resulting count will be placed in a new column, named "agrCustCount", in every row of the datalayer, and will be accessible as the token @Data.agrCustCount~. Because the value is in every row, it's available for use with every row of data in the table.

Dynamic Application

The DsAggregate Column element has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the Dataview. If the value evaluates to *False*, the element is ignored and does not affect the Dataview. This powerful feature allows you to dynamically determine if the DsAggregate Column will be added to the Dataview or not.

The DsCalculated Column

The **DsCalculated Column** element is used with DataLayer.Data Services to add a new column to a data containing a value that's the result of a expression. The expression can include data from other columns, tokens, and literals, and is evaluated for every row in the data.

The following topics discsus the DsCalculated Column element:

- DsCalculated Column Attributes
- DsCalculated Column Usage Examples

For more information about Dataviews, see *Dataviews*.

About the DsCalculated Column

The DsCalculated Column element discussed here is available in Logi Info v12.5 but has been deprecated in later Info versions; consult the Release Notes for specific details.

The DsCalculated Column is similar to other elements designed to *extend* the data available in a datalayer. Like the **DsAggregate Column** element, it's added as a child element beneath DataLayer.Data Services and adds a **new column** to the data. This column value is derived data which is accessible using an @Data token.

The new column's value results from evaluating an expression, for each row in the datalayer. This calculated value is populated in the new column in every row of the data and can be used in further calculations, Summary Rows, Link Params, etc.

DsCalculated Column Attributes

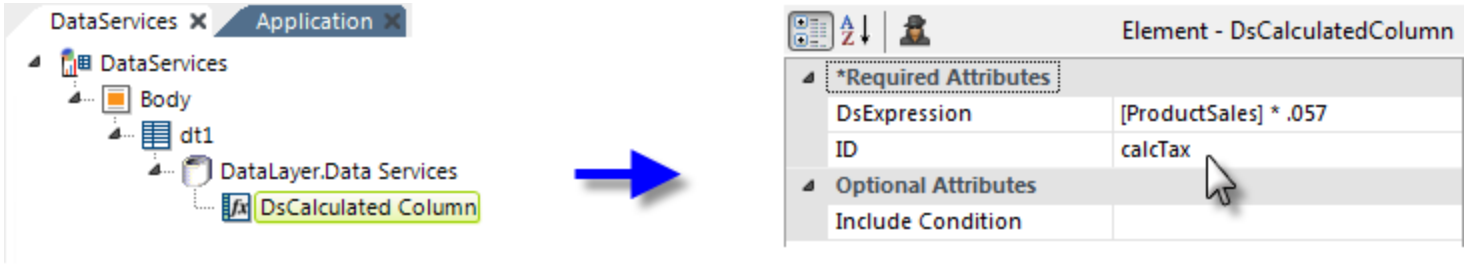
The attributes for the DsCalculated Column element are described below:

Attribute	Description
DsExpression	(Required) Specifies an expression that returns a string, numeric, or boolean value. Tokens, including data from other data columns, literals, and functions may be used here. The syntax you use to create expressions is discussed in detail in *Dsexpression Reference*.
ID	(Required) Specifies a unique ID for the element, and becomes the name of the column added to the data.
Include Condition	Specifies whether or not this element will be processed when the report runs. If left blank or if it contains an expression that evaluates to *True*, the element is processed and the new column is added to the data. If it evaluates to *False*, the element is ignored and does not affect the data.

DsCalculated Column Usage Examples

Here are some examples of the DsCalculated Column element in use.

1. The following shows a simple implementation - calculating sales tax:



In the definition shown above, a Data Table and a DataLayer.Data Services element have been added. A **DsCalculated Column** element has been added beneath the datalayer and its attributes have been configured so that it will multiply the values in the ProductSales column by a constant to produce the sales tax amount.

CategoryID	ProductName	ProductSales
1	Chai	$4,887.00
2	Aniseed Syrup	$1,724.00
3	Chocolade	$1,282.01
4	Camembert Pierrot	$20,505.40
5	Filo Mix	$2,124.15
6	Alice Mutton	$17,604.60
7	Longlife Tofu	$1,000.50
8	Boston Crab Meat	$9,814.73

Original Data

CategoryID	ProductName	ProductSales	calcTax
1	Chai	$4,887.00	$278.56
2	Aniseed Syrup	$1,724.00	$98.27
3	Chocolade	$1,282.01	$73.07
4	Camembert Pierrot	$20,505.40	$1,168.81
5	Filo Mix	$2,124.15	$121.08
6	Alice Mutton	$17,604.60	$1,003.46
7	Longlife Tofu	$1,000.50	$57.03
8	Boston Crab Meat	$9,814.73	$559.44

Data after using DsCalculated Column
to add calcTax column

The resulting sales tax amount will be placed in a new column, named calcTax, in every row of the data, as shown above. The data will be accessible as the token @Data.calcTax~. Instead of hard-coding the sales tax rate in the expression, it could be made dynamic using a token, like this:

```
[ProductSales] * @Request.TaxRate~
```

Here's a slightly different example: the following expression uses the IIF function to apply different tax rates, depending on other criteria:

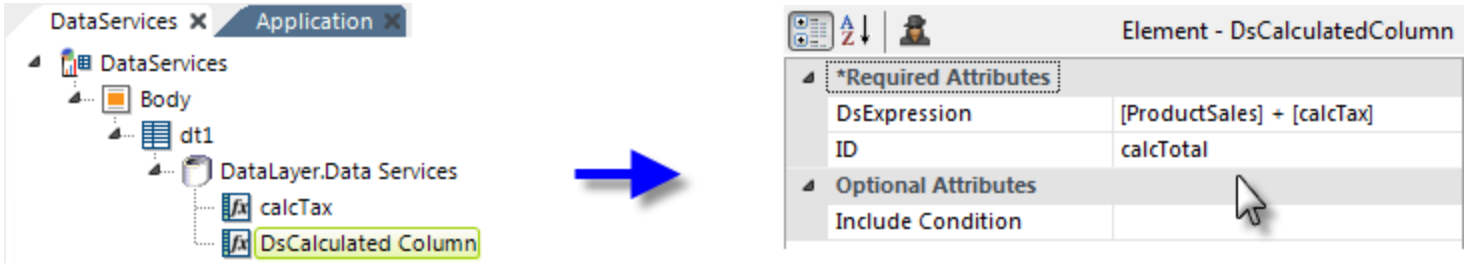
```
IIF([CategoryID] == 4, [ProductSales] * @Request.TaxRate1~, [ProductSales] * @Request.TaxRate2~)
```

Of course, your expression doesn't *have* to use data from the datalayer at all. The following example simply creates a new column and puts the elapsed seconds for every row for it:

```
DateDiff('Second', @Request.StartTime~, DatePart('Second', Now()))
```

The possibilities are endless and allow developers a great deal of flexibility in manipulating the datalayer.

2. This example illustrates how the value created by one DsCalculated Column can be used in another DsCalculated Column.



In the definition shown above, a second **DsCalculated Column** has been added beneath the datalayer. As you can see, its expression attribute uses data from one of the other data columns *and* the first calculated column.

CategoryID	ProductName	ProductSales
1	Chai	$4,887.00
2	Aniseed Syrup	$1,724.00
3	Chocolade	$1,282.01
4	Camembert Pierrot	$20,505.40
5	Filo Mix	$2,124.15
6	Alice Mutton	$17,604.60
7	Longlife Tofu	$1,000.50
8	Boston Crab Meat	$9,814.73

Original Data

CategoryID	ProductName	ProductSales	calcTax	calcTotal
1	Chai	$4,887.00	$278.56	$5,165.56
2	Aniseed Syrup	$1,724.00	$98.27	$1,822.27
3	Chocolade	$1,282.01	$73.07	$1,355.08
4	Camembert Pierrot	$20,505.40	$1,168.81	$21,674.21
5	Filo Mix	$2,124.15	$121.08	$2,245.23
6	Alice Mutton	$17,604.60	$1,003.46	$18,608.06
7	Longlife Tofu	$1,000.50	$57.03	$1,057.53
8	Boston Crab Meat	$9,814.73	$559.44	$10,374.17

Data after using a second DsCalculated Column
to add the ProductSales and calcTax column values together

The resulting total amount will be placed in a new column, named calcTotal, in every row of the data, as shown above. The data will be accessible as the token @Data.calcTotal~.

The DsCondition Filter

The **DsCondition Filter** element is used with DataLayer.Data Services to filter out data rows. It's analogous to using a *WHERE* clause in a SQL query and adds a filtering operation to a Dataview.

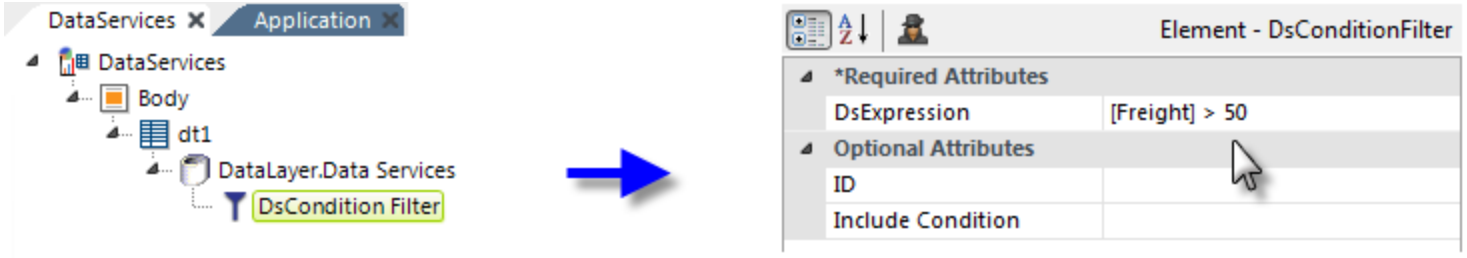
The following topics discuss the DsCondition Filter:

- Applying the DsCondition Filter
- Using Multiple and Dynamic DsCondition Filters
- Use with DsCompare Filters

For more information about Dataviews, see *Dataviews*.

Applying the DsCondition Filter

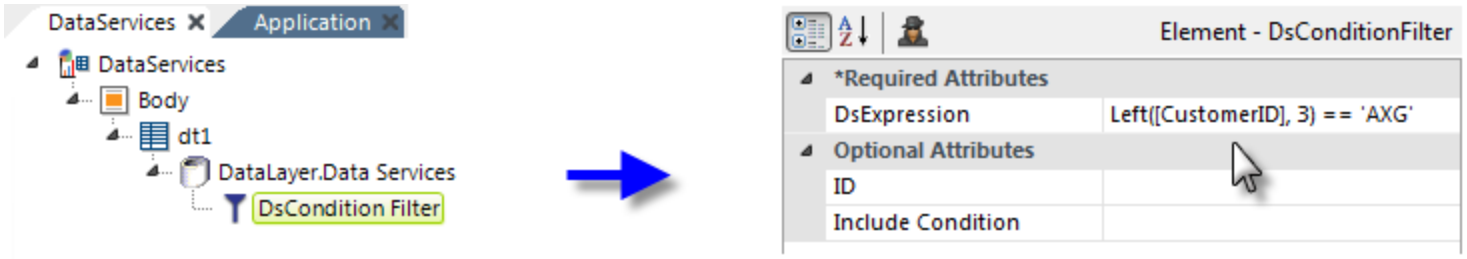
The DsCondition Filter element discussed here is available in Logi Info v12.5 but has been deprecated in later Info versions; consult the Release Notes for specific details. The following example illustrates how the **DsCondition Filter** element is used:



1. As shown above, a DsCondition Filter element is added as a child of the **DataLayer.Data Services** element..
2. Its **DsExpression** attribute value is set to an expression that compares the data to some value.

The element causes a filtering operation to be included in the processing of the Dataview referenced by the datalayer. All records where the expression evaluates to *False* are removed; in this case, where the values in the *Freight* column equal 50 or less.

The DsCondition Filter element's **DsExpression** attribute also supports script functions and can evaluate them to filter data:



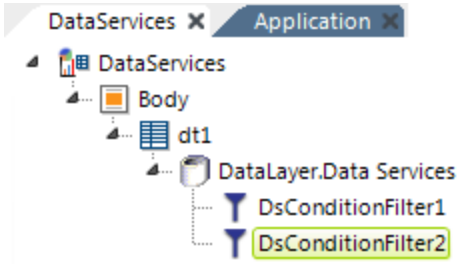
1. As shown above, a DsCondition Filter element is added as a child of the datalayer element.
2. Its DsExpression attribute value is set to an expression that uses the function *Left()* to compare the data to some value.

The element causes a filtering operation to be included in the processing of the Dataview referenced by the datalayer. All records where the expression evaluates to *False* are removed; in this case, where the values in the *CustomerID* column do not start with "AXG".

The syntax you use to create expressions is discussed in detail in *Dsexpression Reference*.

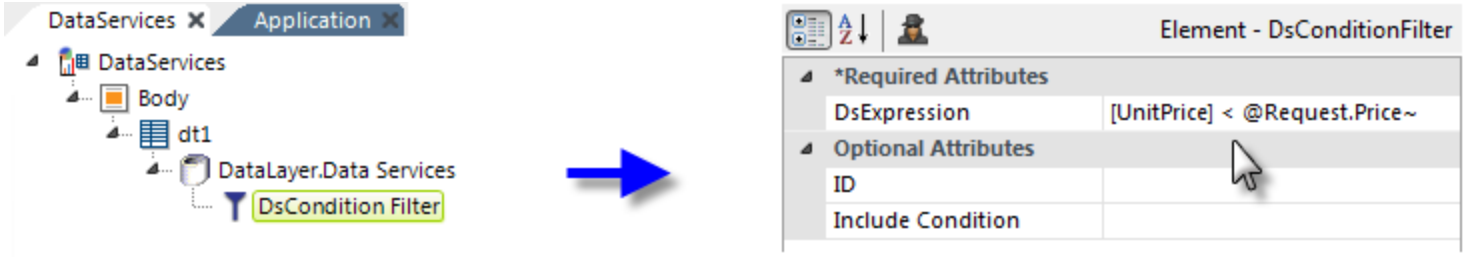
Using Multiple and Dynamic DsCondition Filters

Developers can use two or more DsCondition Filter elements:



In this case, they will be processed in the Dataview *sequentially*, based on their arrangement from top-to-bottom in the element tree. So, in the example above, after the first filter is applied, only records that meet its criteria will remain to be processed in the Dataview by the second filter.

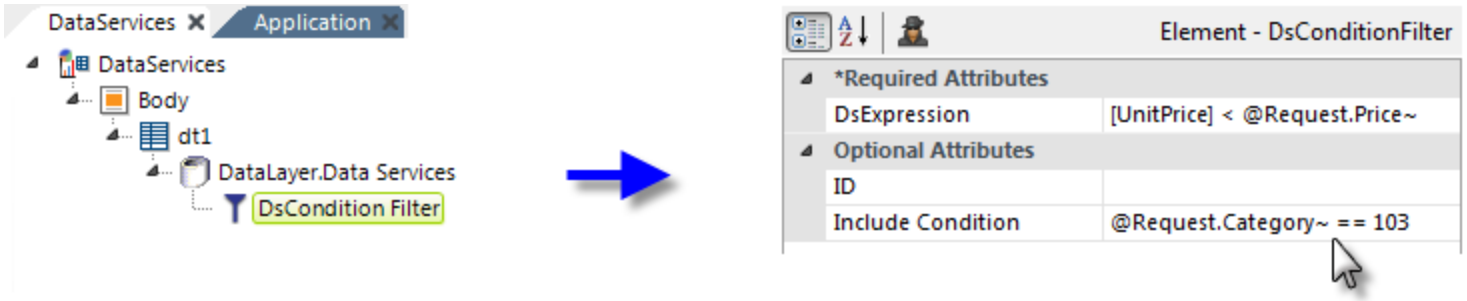
You can also make DsCondition Filter elements *dynamic* by using tokens within their expressions:



As shown above, DsExpression expressions can contain tokens such as @Request, so that comparison values are dynamic at runtime.

Sometimes it is necessary to ensure that tokens have default values when running a report. When using @Request tokens, the **Default Request Params** element can be used to create default values for each request variable.

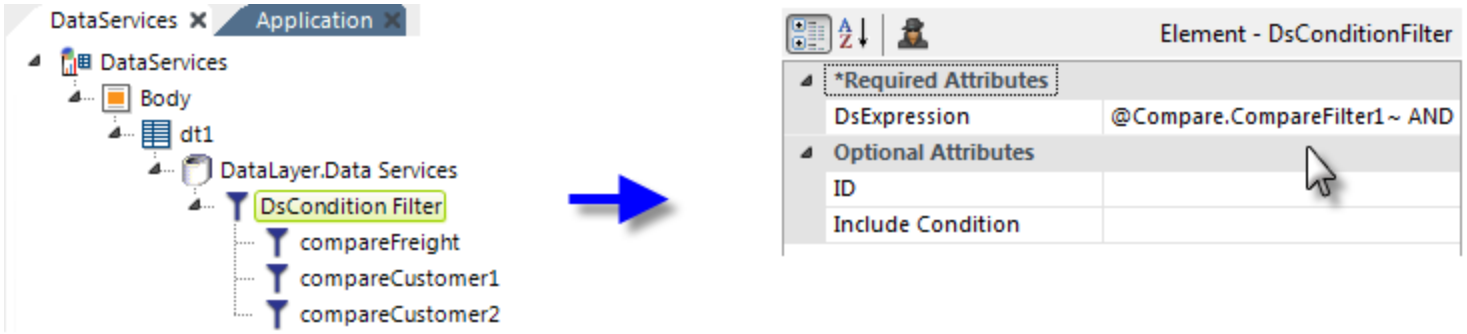
Like other filter elements, the DsCondition Filter has an **Include Condition** attribute:



If the value of this attribute is left blank or contains an expression that evaluates to *True*, the element is applied to the Dataview. If the value evaluates to *False*, the element is ignored and has no affect. This powerful feature allows you to dynamically determine if filtering will be applied.

Use with DsCompare Filters

For improved performance, DsCondition Filters can also be used in combination with **DsCompare Filter** elements to create complex expressions that include ANDs and ORs and operations grouping.



For example, consider the element arrangement shown above. For each row of the Dataview, the three **DsCompare Filters** will be evaluated first. Each of them will produce a *True* or *False* value as a result of their comparison operations. Those values will be assigned to @Compare tokens which include the element ID, for example: `@Compare.CompareFilter1~`. The scope of these tokens is limited to the parent DsCondition Filter.

After all of the DsCompare Filters are evaluated, the **DsCondition Filter** will be evaluated and its **DsExpression** attribute value can include expressions that use the @Compare tokens. For example, the complete DsExpression attribute value above might be:

```
@Compare.CompareFilter1~ AND (@Compare.CompareFilter2~ OR @Compare.CompareFilter3~)
```

And, of course, you can also use other types of tokens and literal values in the DsCondition attribute to get the expression you need, such as:

```
[UnitPrice] <= 100 AND (@Compare.CompareFilter1~ OR @Compare.CompareFilter2~)
```

You may be able achieve the same results with complicated scripting for the DsExpression attribute value, however, in most cases use of multiple DsCompare Filters as shown in this example will produce better performance.

DsGroup

The **DsGroup** element is used with DataLayer.Data Services to add a grouping operation to Dataview processing. It also gives you the ability to create aggregate values from the grouped rows.

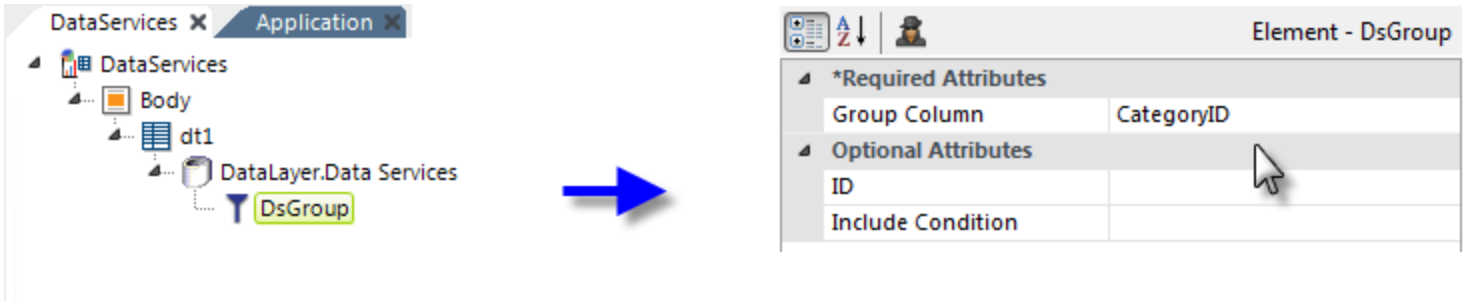
The following topics discuss the DsGroup element:

- Grouping Data
- Aggregating Grouped Data

For more information about Dataviews, see *Dataviews*.

Grouping Data

The DsGroup element discussed here is available in Logi Info v12.5 but has been deprecated in later Info versions; consult the Release Notes for specific details. Here's an example of the **DsGroup** element in action:



As shown above, a **DsGroup** element is added as a child of the DataLayer.Data Services element. Its **Group Column** attribute is set to the name of a column in the data. The data will be grouped based on the values in this column.

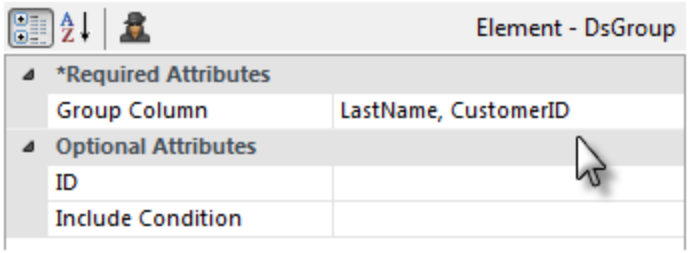
CategoryID
1
2
3
4
6
8

The results are shown above. The rows are grouped by CategoryID and one row for each category is included.

When grouping is applied, only values from the column named in the Group Column attribute are returned to the datalayer.

Grouping on Multiple Columns

Grouping can be accomplished on more than one column in the datalayer at the same time. This example shows the effects of grouping multiple columns:



As shown above, to group on multiple columns, enter their names in the Group Column attribute in a comma-separated list. The data will be grouped based on the values in these columns, in the order of the column names in the list.

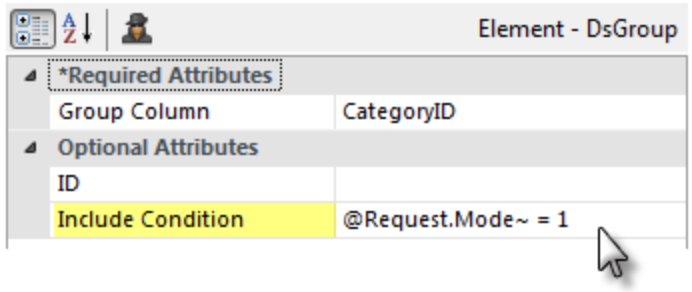
LastName	CustomerID
Callahan	CONSH
Callahan	ERNSH
Callahan	ERNSH
Callahan	QUEDE
Callahan	REGGC
Callahan	VICTE
Callahan	WARTH
Callahan	WARTH
Callahan	WARTH
Davolio	EASTC
Davolio	LINOD
Fuller	FAMIA
Fuller	FAMIA
Fuller	OTTIK
Fuller	OTTIK

The results are shown above. The rows are grouped first by LastName and then by CustomerID.

 When grouping is applied, only values from the columns named in the Group Column attribute are returned to the datalayer.

Conditional Grouping

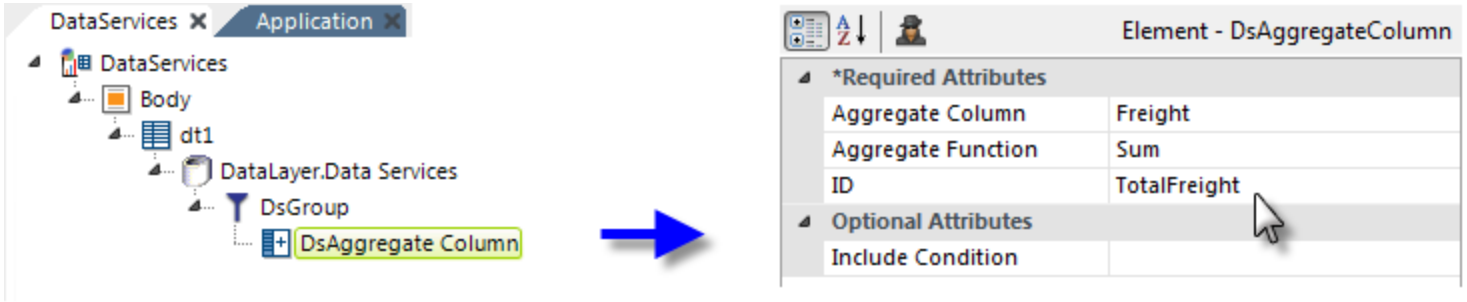
The DsGroup element can be employed dynamically using its **Include Condition** attribute.



If the value of this attribute is left blank or contains an expression that evaluates to *True*, the element is applied to the data. If the value evaluates to *False*, the element is ignored and does not affect the data. This allows you to dynamically determine if the data will be grouped or not.

Aggregating Grouped Data

A very common requirement when grouping data is to perform some kind of *aggregation*, such as summing or counting, of the grouped values. The **DsAggregate Column** element, when used as a child of DsGroup, gives you the ability to generate this summary data for the grouped columns in the data.



In the example above, the DsGroup element's Group Column has been set to ShipCity. A DsAggregate Column element has been added as a child element and configured to Sum the values found in the grouped Freight columns.

ShipCity	TotalFreight
Akron	12659455.0793
Albuquerque	12551758.6331
Anaheim	12688300.4631
Anchorage	12661705.261
Arlington	24844124.2264
Atlanta	12481864.7368
Aurora	12408596.7211
Austin	12623506.8993
Bakersfield	12567353.8584

And the results are shown above.

When grouping is applied and aggregation is used, only values from the columns named in the Group Column and the Aggregate Column attributes are returned to the datalayer.

DsSort

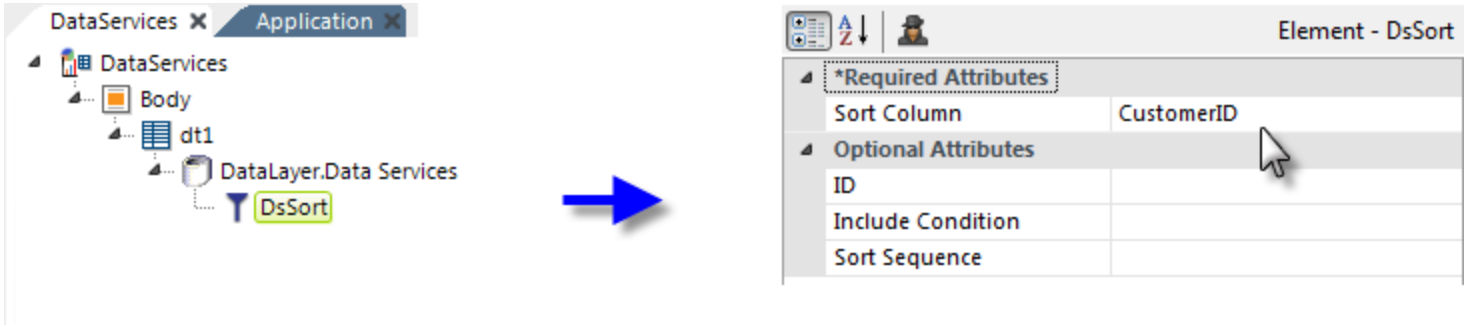
The **DsSort** element is used with DataLayer.Data Services to add a sorting operation to Dataview processing. This topic discusses the use of this element.
For more information about Dataviews, see *Dataviews*.

Using the DsSort Element

The DsSort element discussed here is available in Logi Info v12.5 but has been deprecated in later Info versions; consult the Release Notes for specific details. The following examples illustrate how the DsSort element can be used:

OrderID	CustomerID	EmployeeID	OrderDate	Freight	ShipName	ShipCountry	ShipCity
1	AXGMN	224	1/17/1994 8:02:39 PM	836.0464	Trevor876	United States	Pittsburgh
2	VKRFA	315	5/31/1986 7:18:32 PM	781.2219	Sam3	United States	St. Petersburg
3	CGHZU	622	8/18/1970 7:02:12 AM	116.4723	Gina343	United States	Seattle
4	EVMCI	469	7/3/1990 9:37:38 AM	535.6755	Nakia691	United States	Memphis
5	LUMLF	930	12/24/1973 5:59:46 AM	187.3961	Bennie	United States	Colorado
6	QICJM	377	10/19/1966 2:30:29 PM	746.4801	Regina278	United States	Virginia Beach

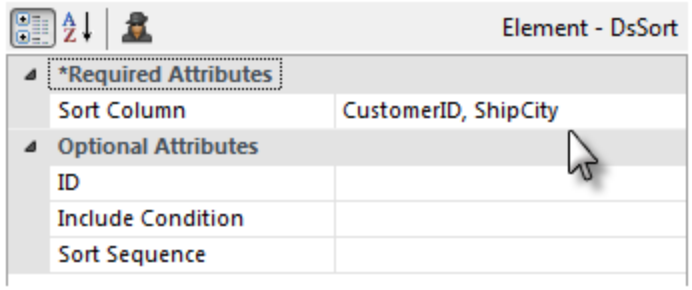
The table above shows the data as it appears without any sorting. Note the order of the CustomerID column data. Now let's sort that column:



1. As shown above, add **DsSort** element beneath a DataLayer.Data Services element.
2. Set its **Sort Column** attribute value to the name of the column in the data, *CustomerID*, to sort on.
3. The optional **Sort Sequence** attribute value, which defaults to *Ascending*, can be set to *Descending* if desired.

OrderID	CustomerID	EmployeeID	OrderDate	Freight	ShipName	ShipCountry	ShipCity
2352018	AAAAC	652	9/2/1990 3:03:28 AM	945.3282	Roman2	United States	Riverside
971668	AAAAD	579	6/30/2007 4:58:35 AM	670.128	Gena	United States	Lubbock
613002	AAAAF	76	6/26/1991 10:07:32 AM	309.333	Carlos26	United States	Chicago
1964489	AAAAG	406	4/26/1978 3:30:51 AM	54.9684	Evelyn386	United States	Fort Wayne
2178830	AAAAG	727	10/10/1986 3:30:35 PM	543.1911	Colin1	United States	Boston
931239	AAAAI	862	2/22/1987 3:28:22 PM	317.1819	Alejandro38	United States	St. Paul
2000400	AAAAL	248	11/1/1969 3:08:54 PM	445.8621	Jolene93	United States	Albuquerque

When the report is run again, the data will now be sorted in Ascending order on the *CustomerID* column, as shown above.



5. If a second column is added to the first in the **Sort Column** attribute value, separated by a comma as shown above, the sort will occur on multiple columns.

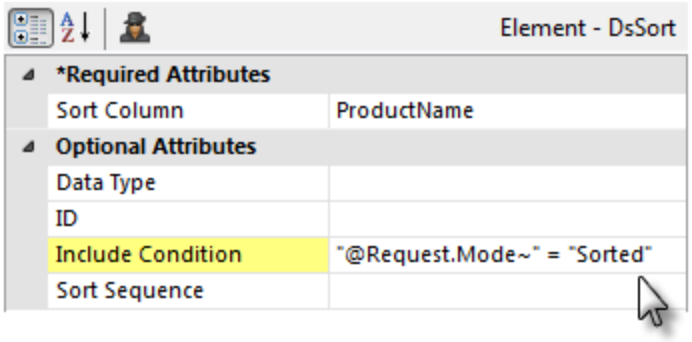
OrderID	CustomerID	ShipCity
2352018	AAAAC	Chicago
971668	AAAAC	Lubbock
613002	AAAAC	Riverside
2178830	AAAAG	Boston
1964489	AAAAG	Fort Wayne
931239	AAAAI	Albuquerque
2000400	AAAAI	St. Paul

The data will now be sorted first on one column and then on the second, as shown above (we've removed some intervening columns in the image to make the sorted columns easier to compare). You can also set different sort orders for different columns by entering the *Ascending* and *Descending* options in the Sort Column attribute, separated by commas (this assumes the columns are the same data type). In situations where the columns to be sorted are *not* the same data type and/or need to be sorted in opposite directions (one in *Ascending* order and one in *Descending* order) additional **DsSort** elements can be used, one for each column.

The *order* of the DsSort elements in the element tree will be significant in this case.

Dynamic Sorting

The DsSort element has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the data. If the value evaluates to *False*, the element is ignored and does not affect the data. This allows you to dynamically determine if the data will be sorted or not.

The Analysis Chart

The Analysis Chart is a dynamic chart set that provides interactive data analysis capabilities.

The following topics introduce developers to the **Analysis Chart** super-element:

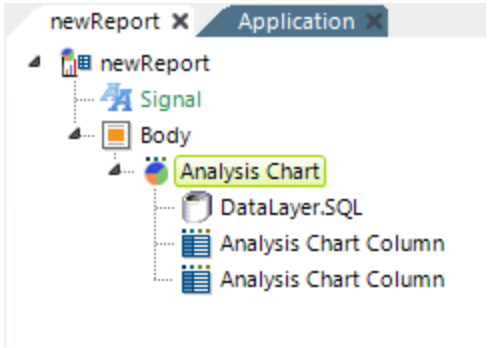
- Analysis Chart Attributes
- Adding Analysis Chart Columns
- Adding Additional Series
- Adding Forecasting
- Saving Individual User Settings
- Adding Charts to Dashboards
- Customizing Analysis Chart Appearance

About the Analysis Chart

The Analysis Chart allow users to display charts at runtime, based on a discrete data set and limited to chart types the developer chooses to include.

 Chart Canvas charts are used by default by the Analysis Chart, in all *new applications*. Older Logi applications that are *upgraded* to v12 will use the classic static charts for their Analysis Charts. To force upgraded apps to use Chart Canvas charts, add the constant rdFavorChartCanvas = *True* to your _Settings definition.

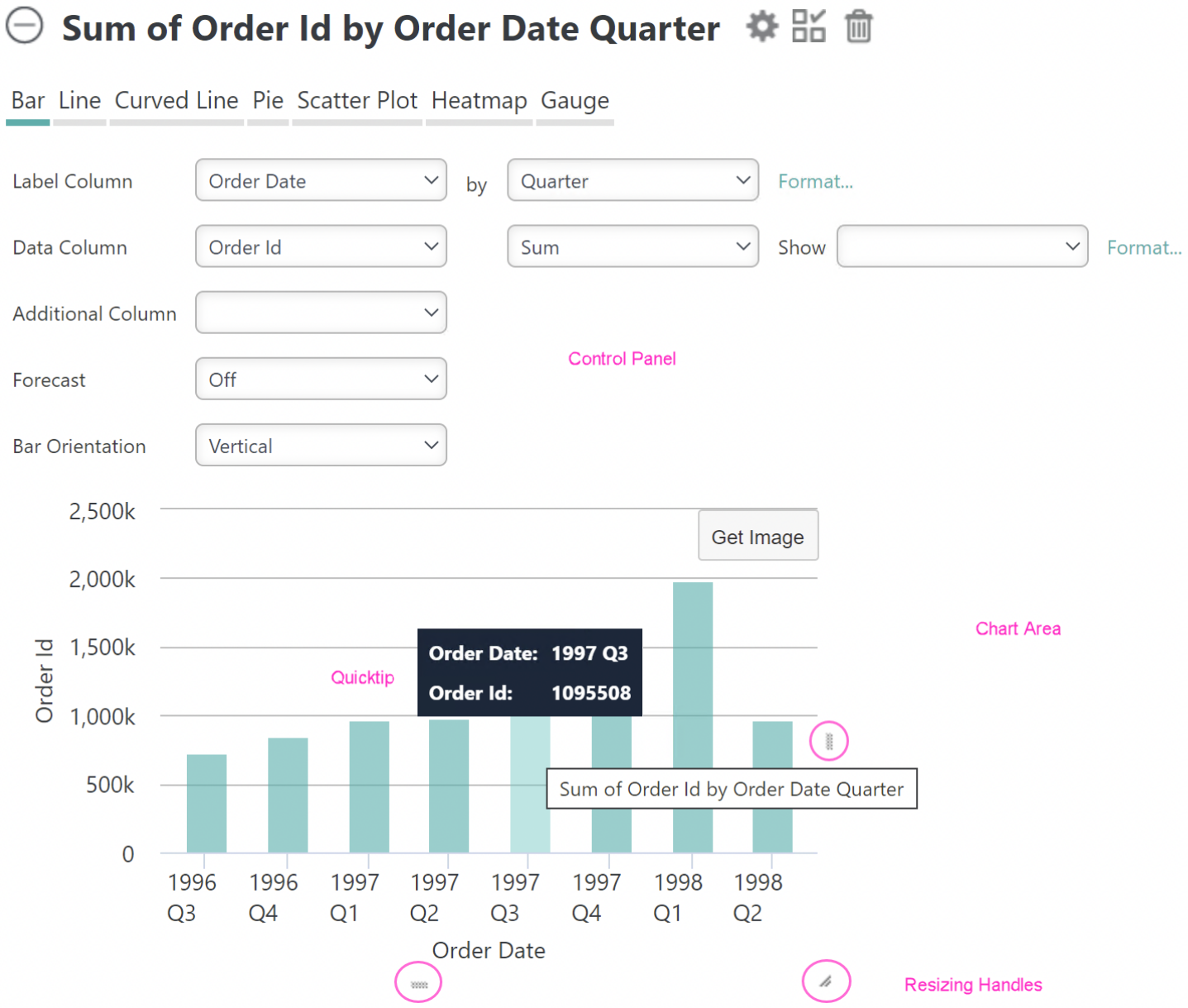
Like Data Tables, the Analysis Chart uses child column elements to include data for display.



The example above shows a typical set of Analysis Chart elements:

- An **Analysis Chart** element
- A **DataLayer** element to retrieve the data
- One or more **Analysis Chart Column** elements to make data available for analysis

Analysis Chart Column elements do not require any child elements, such as a Label element, in order to make data visible. The element's **Data Type** attribute determines how each column can be used. "Text"-type columns can be used as Label Columns for Pie and Bar charts. If your chart uses text-type columns, the **Relevance** option replaces Forecast. Relevance allows you to tune the data shown by *Automatic*, *Rank*, and *Percentage*. "Number"-type columns may be used in the X- and Y-axis of Line and Scatter charts. "Date" columns may be used in the X-axis. In addition, the **Format** attribute can be used to format the values that appear in tooltips displayed when the cursor is hovered over various chart areas. The **Format...** option, shown next to the Label Column and Data Column, allows you to format captions and labels for your charts. If you add an Additional Column(s), this option will also be available. For additional information, see Analysis Grid - Editing Chart Labels and Captions.


Sum of Order Id by Order Date Quarter
Bar Line Curved Line Pie Scatter Plot Heatmap Gauge
Label Column
Order Date
by
Quarter
Format...
Data Column
Order Id
Sum
Show
Format...
Additional Column
Forecast
Off
Bar Orientation
Vertical
Control Panel
Get Image
2,500k
2,000k
1,500k
1,000k
500k
0
Order Id
Order Date: 1997 Q3
Order Id: 1095508
Quicktip
Chart Area
Sum of Order Id by Order Date Quarter
1996 Q3
1996 Q4
1997 Q1
1997 Q2
1997 Q3
1997 Q4
1998 Q1
1998 Q2
Order Date
Resizing Handles

As shown above, the display produced by this element consists of a **Control Panel** and a **Chart Area**. By default, the Control Panel will appear *above* the Chart Area, as shown. However, the Analysis Chart can be configured to display the Control Panel to the *left* of the Chart Area instead.

The chart's **title** changes dynamically based on the selections the user makes at runtime; it cannot be set independently by the developer.

Resizing handles, which are only visible when the mouse hovers over the chart, are also provided so that the user can resize the chart at runtime.

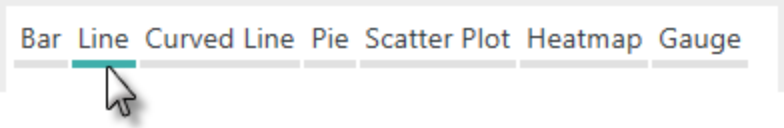
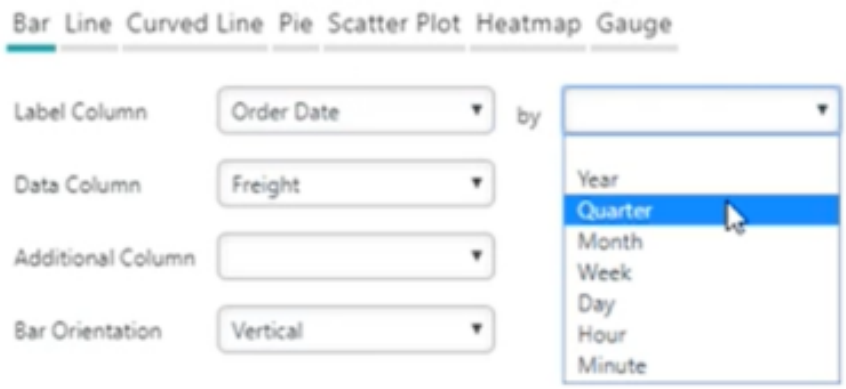


Chart type options, shown above, allow the user to select different chart types at runtime. The types available to the user can be restricted by the developer when configuring the element and will also be limited based on the nature of the data. For example, Pie and Bar chart types are only available when there is at least one Analysis Chart Column with its **Data Type** attribute set to *Text*.

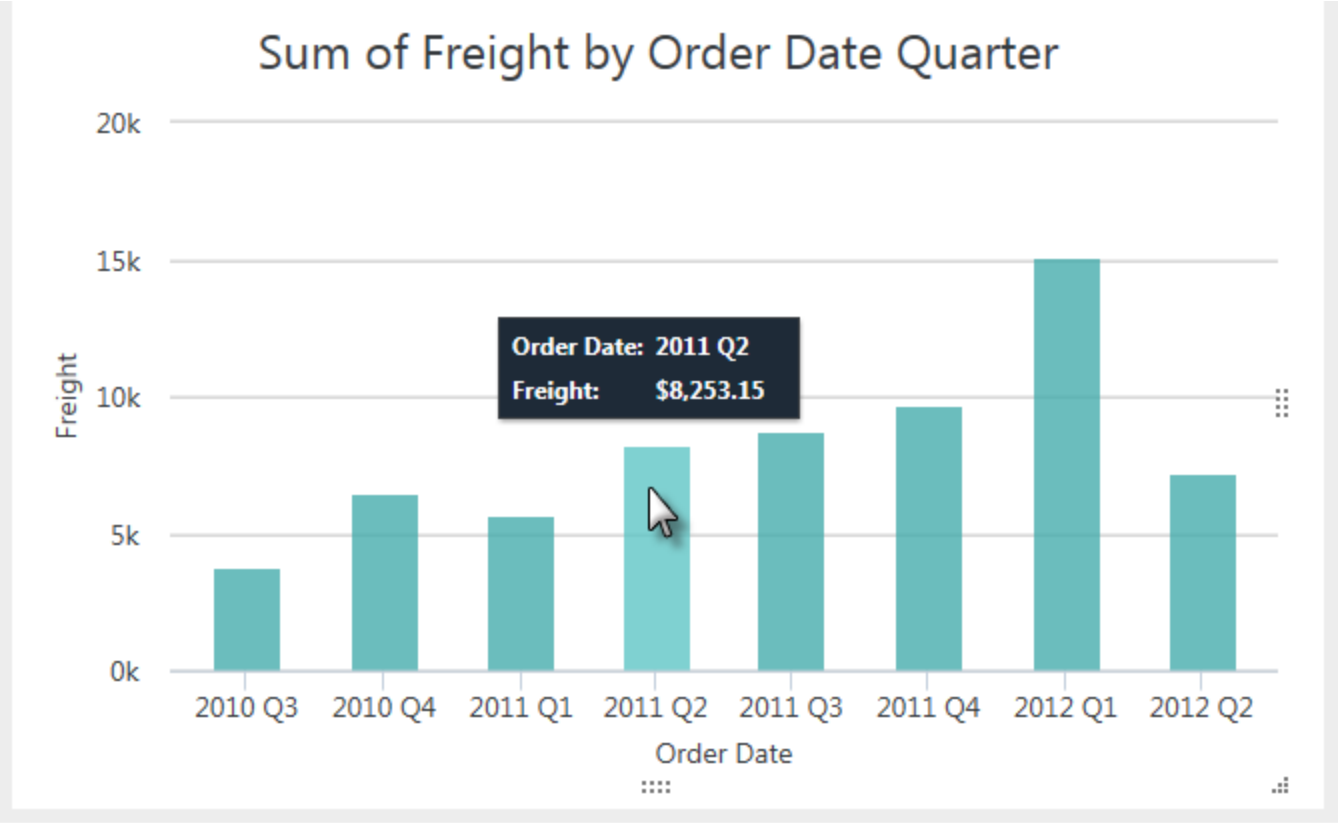
The default orientation of bar charts that are not based on a time series is *Horizontal*, to make it easier to read X-axis label data values. This can be changed at runtime by the user to Vertical.

A built-in **Group Filter** is available for X-axis (Label) values:



For example, if a DateTime type column is selected, the filter appears in the control panel automatically, as shown above. It allows users to group data by Year, Quarter, Month, Week, Day Hour, Minute time periods.

Quicktips are automatically included:



As shown above, this feature displays a pop-up "balloon", containing detail data, when the mouse is hovered over a data point. This feature is available for *all* chart types and no action on the developer's part is required to enable it.

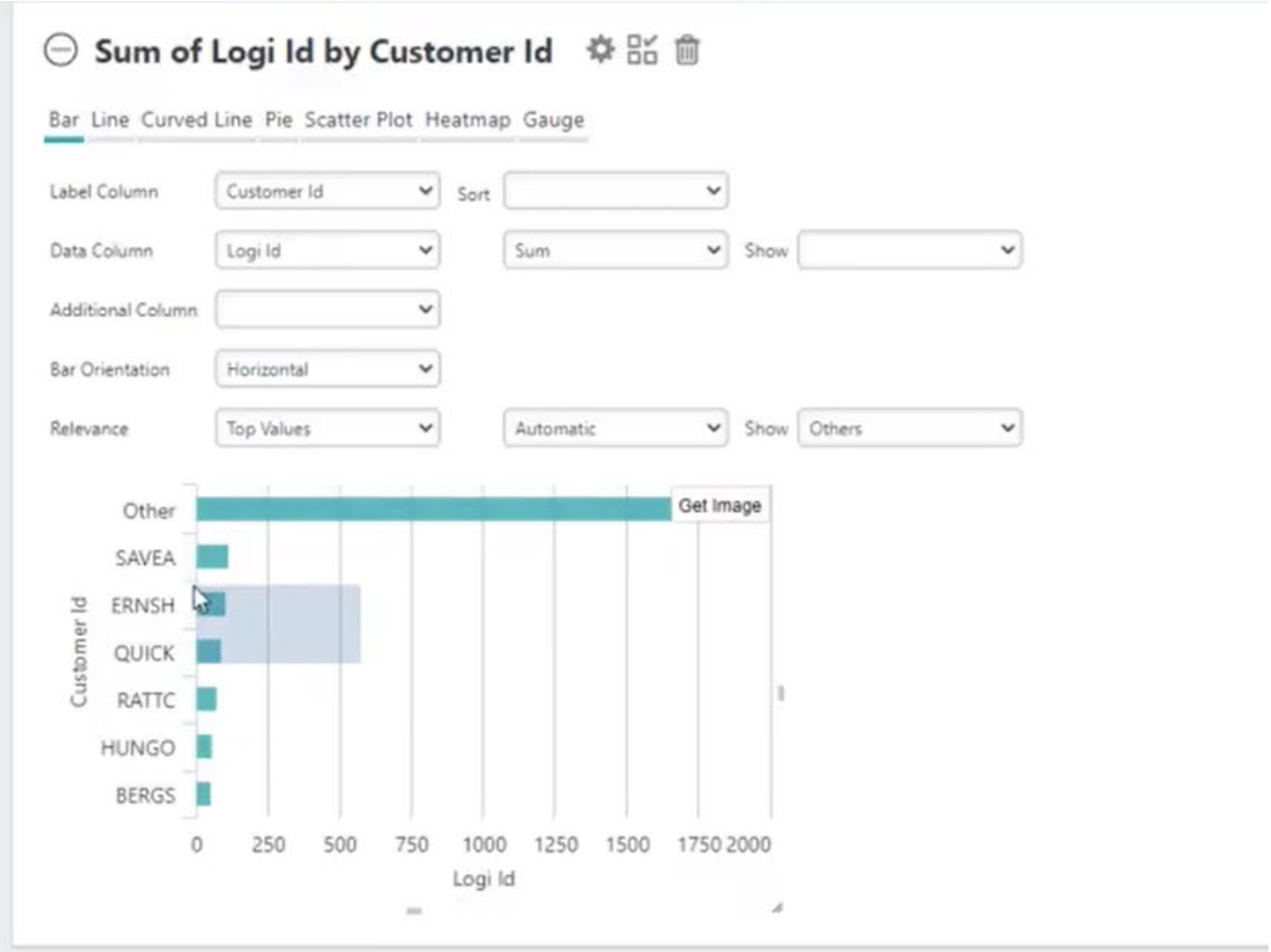
If your application has been configured for it, you can now use zoom filtering on the following charts:

- Bar
- Line

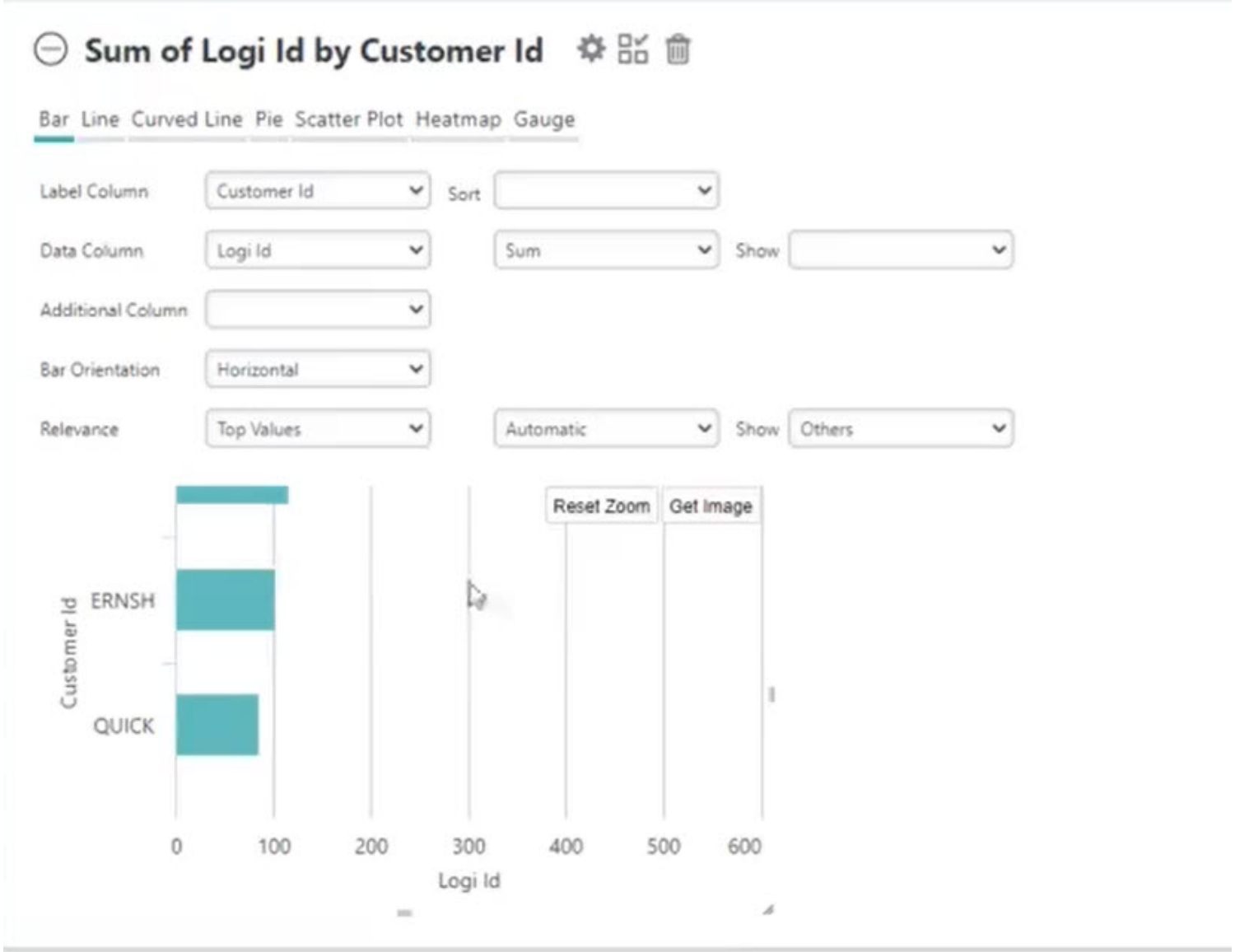
- Curved Line
- Scatter Plot

For information on configuring this feature, see *Analysis Grid Attributes*.

To utilize this feature, simply select the data you want to zoom in on, like below:



Upon releasing your cursor, Info will re-size the chart:



To remove the zoom and return the chart to its original state, select **Reset Zoom**.

Charts with zoom filtering can also be saved to the Dashboard for future analysis. For more information about Dashboards, see Dashboard for End Users.

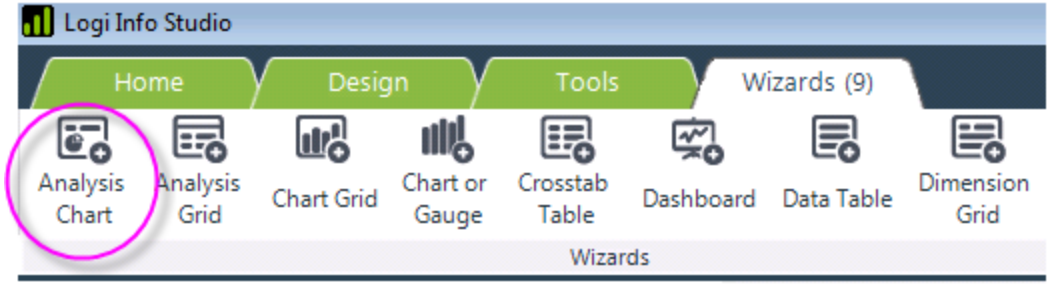
If you're working with **very large data sets**, you may be able to use a special-purpose datalayer for better performance. You can find out more about it and review the restrictions in *DataLayer.ActiveSQL*.

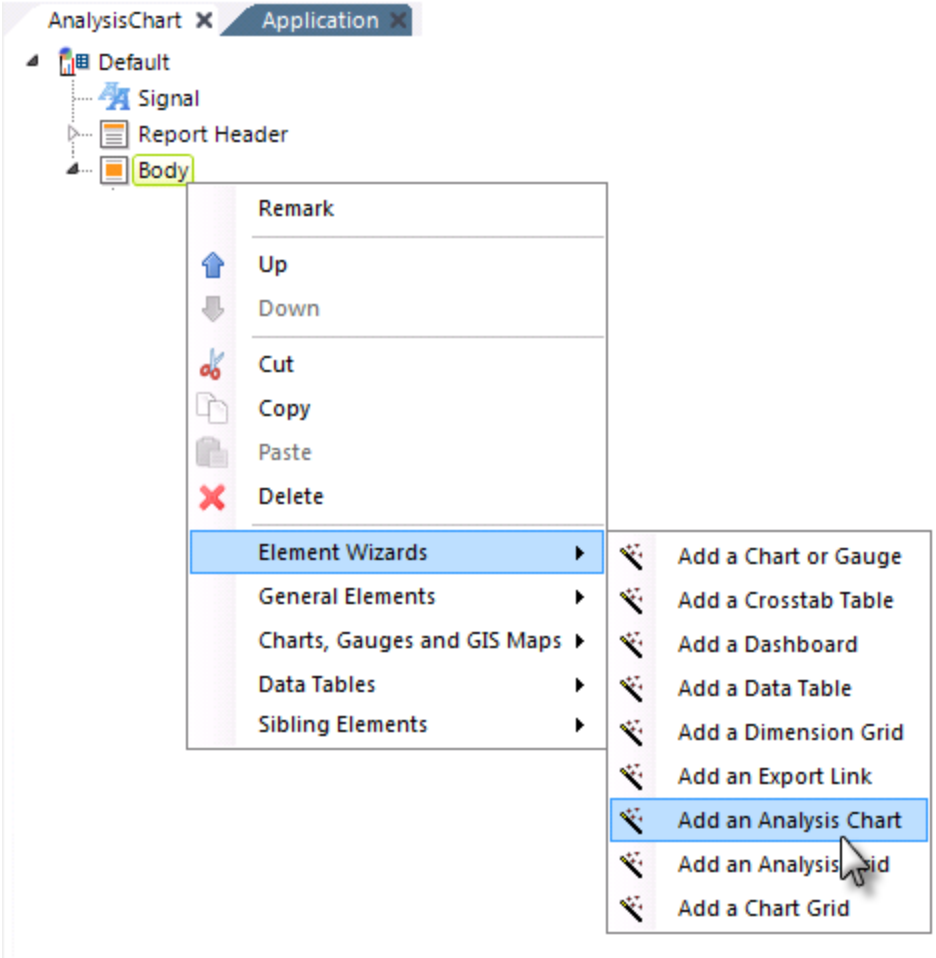
This datalayer provides the basic SQL query needed to retrieve data but doesn't initially retrieve and cache *all* of the result set. Instead, in response to runtime manipulations of the Analysis Chart interface, it dynamically modifies its own SQL query and retrieves the minimum required amount of data with each request. This increases the number and frequency of SQL queries the database server must handle (and, generally, does not prove to be a burden on the database server) but improves performance for the user.

A **Zero Rows Message** element is available as a child element of the Analysis Chart. This element allows developers to display a customized message if the datasource returns no data. The message text and its font type, size, and color can be set using the element's attributes.

Studio Wizard Available

Logi Studio includes a wizard that will help you to build and configure an Analysis Chart:


Logi Info Studio
Home
Design
Tools
Wizards (9)
Analysis Chart
Analysis Grid
Chart Grid
Chart or Gauge
Crosstab Table
Dashboard
Data Table
Dimension Grid
Wizards


AnalysisChart
Application
Default
Signal
Report Header
Body
Remark
Up
Down
Cut
Copy
Paste
Delete
Element Wizards
General Elements
Charts, Gauges and GIS Maps
Data Tables
Sibling Elements
Add a Chart or Gauge
Add a Crosstab Table
Add a Dashboard
Add a Data Table
Add a Dimension Grid
Add an Export Link
Add an Analysis Chart
Add an Analysis Grid
Add a Chart Grid

As shown above, you can launch the wizard from Studio's main menu Wizards tab, or by right-clicking a container element and using its context menus. The wizard will walk you through decisions about connecting to and retrieving data, and about which data columns to use, and will then insert the configured elements into your definition.

Retaining User Settings

With all these possibilities, it may be important for users to be able to **save** their **settings** between sessions. Settings can be automatically saved on the web server and made available to users during their next session (this is discussed in more detail below).

Embedded Sub-Report Caveat

 We *do not* recommend that you use this element within an embedded sub-report.

Analysis Chart - Attributes

The Analysis Chart element can be quickly configured and has the following attributes:

Attribute	Description
ID	(Required) Specifies a unique identifier for this element.
Allow Control Hiding	Specifies whether a "gear" icon will be visible that allows the user to hide the control panel. Default Value: *False.*
Allow Zoom Control	Specifies whether chart zoom is enabled/disabled for Bar Charts, Line Charts, Curve Lines, and Scatter Plots. Default: *False*.
Analysis Chart Types	Specifies a comma-delimited list of the chart types that will be available in the Analysis Chart. Possible values are *Pie*, *Bar*, *Line*, *Spline*, *Scatter*, *Heatmap* and *Gauge*. "Spline" is the Curved Line chart. The first type in the list will be the initial/starting chart type. The Pie chart option only appears when there is at least one Analysis Chart Column with Data Type=*Text*. Bar charts require a column with a *Text* or *Date* type.
Batch Selection	Specifies whether the chart will be updated immediately as each control panel item is changed (*False*) or only when a now-visible OK button is clicked (*True*). Setting this to *True* can speed up the user experience by reducing the number of updates requested from the server.
Class	Specifies a user-defined style sheet class to be used with the Analysis Chart. We recommend the use of a Theme for the best appearance.

Attribute	Description
Color	Specifies the chart colors. Enter a color by name, decimal RGB value, or hex RGB value. Prefix hex values with the pound sign, like #112233. Each bar in a Bar Chart can be made a different color by entering multiple colors, in a comma-separated list.
Control Panel Location	Specifies whether the Control Panel appears above (Top) or beside (Left) the Chart Area. Default value: *Top*
Height	Specifies the height, in pixels, of the chart image. Does *not* include the Control Panel. Leave blank for a default size, which can vary by chart type and data.
Save File	Specifies the fully-qualified file path and name of the file to which runtime user configuration changes will be saved. Must include the .xml file extension and can use an @Function token. Example: `@Function.AppPhysicalPath~\SaveFiles\AFSaveFile.xml` Example: `@Function.AppPhysicalPath~\UserSaveFiles\@Function.UserName~.xml` Save files can also be stored in a database, see *Bookmarks*.
Security Right ID	When using Logi Security, specifies one or more Security Right IDs, separated by commas. Users with these Security Right IDs will be able to view the Analysis Chart, users without them will not.
Template Modifier File	Identifies a custom XML file developers can create containing elements that will override the same elements in the template file, for language- and culture-customization of captions.

Attribute	Description
Width	Specifies the width, in pixels, of the chart image. Does *not* include the Control Panel. Leave blank for a default size, which can vary by chart type and data.

Analysis Chart - Add Analysis Chart Columns

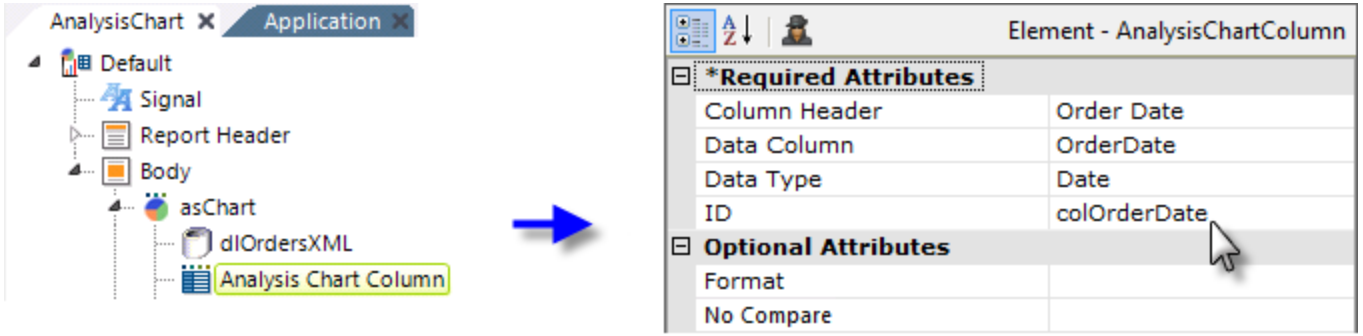
The columns available for use with an Analysis Chart consist of one or more columns from the datalayer.

Analysis Chart Columns have the following attributes:

Attribute	Description
Column Header	(Required) Specifies the text that will appear in the column header.
Data Column	(Required) Specifies the name of the column in the datalayer that will appear in this column.
Data Type	(Required) Specifies the type of data in this column: *Boolean*, *Date*, *DateTime*, *Number*, or *Text*.
ID	(Required) Specifies a unique identifier for this element.
Format	Specifies the formatting characteristics of the data. More information can be found in *Format Data*.
No Aggregates	Set to *True* to prevent this column from being used as an aggregate column for charts. The user will not be able to pick the column from drop-down lists.
No Compare	Set to *True* to prevent this column from being used as a comparison column. The user will not be able to pick the column from the drop-down list. In general, comparison columns should have a limited number of possible values, since each value appears in the chart legend.
No Grouping	Set to *True* to prevent this column from being used as a grouping column when building charts, such as Bar, Pie and Heatmap charts. The user will not be able to pick the column from drop-down lists.

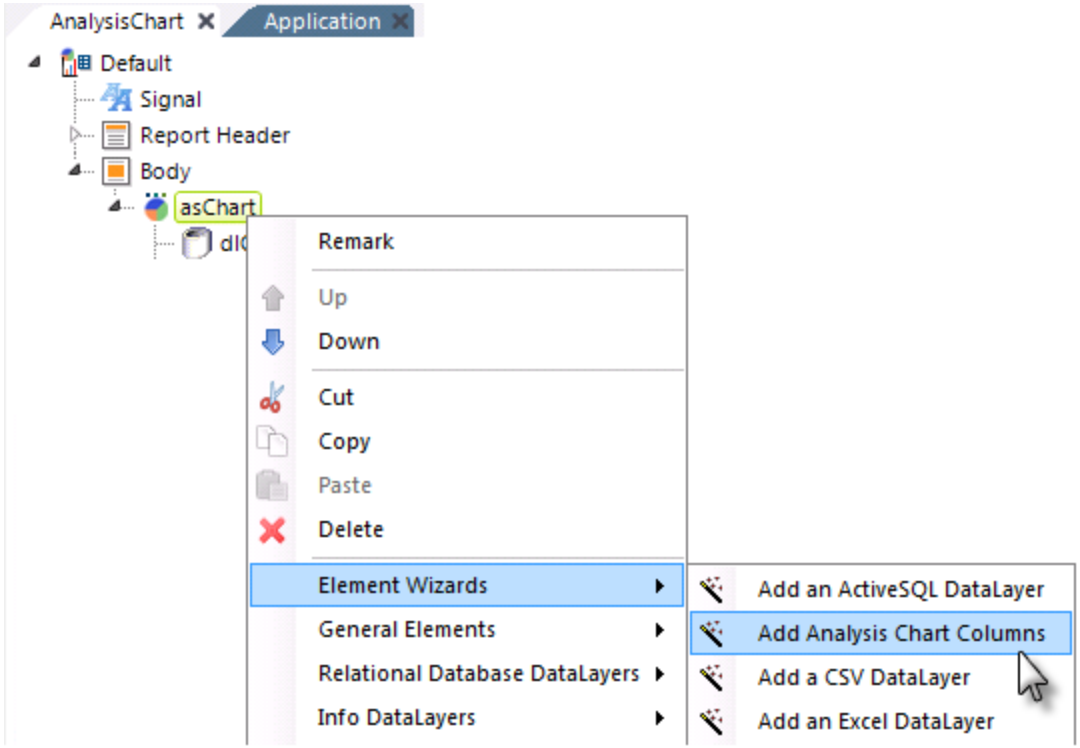
Attribute	Description
Security Right ID	When using Logi Security, specifies one or more Security Right IDs, separated by commas. Users with these Security Right IDs will be able to view the column, users without them will not.

Working with Analysis Chart columns is easy and here's how to add them:

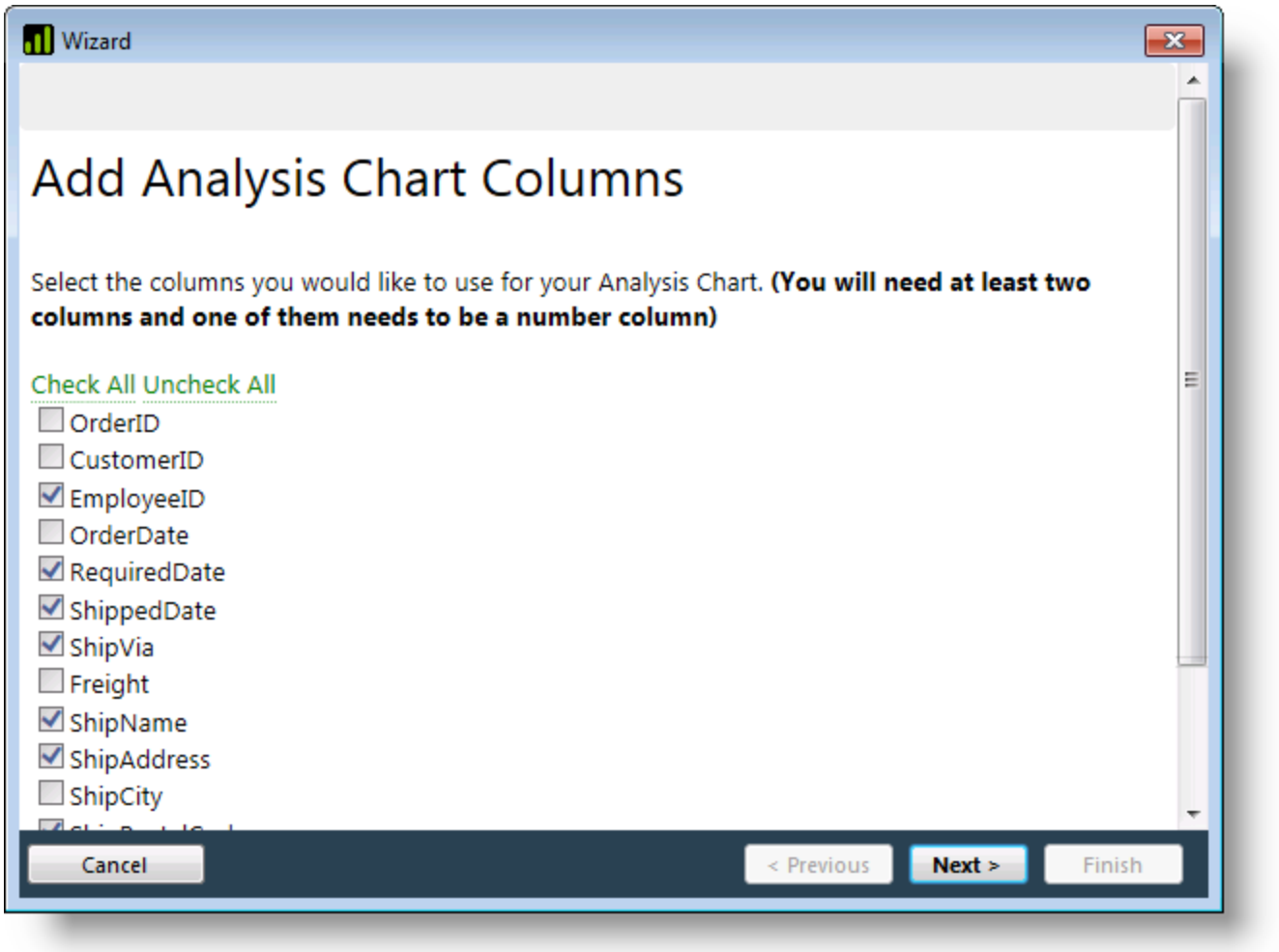


1. In Studio, select the parent **Analysis Chart** element and add the **Analysis Chart Column** element.
2. Set the **Column Header** and **ID** attributes.
3. Set the **Data Column** attribute to the name of a column returned by the datalayer. Tokens are not necessary here; just providing the column name is sufficient.
4. Set the **Data Type** attribute to Text, Number, or Date.

There is no way to make the appearance of an individual Analysis Chart Column conditional.



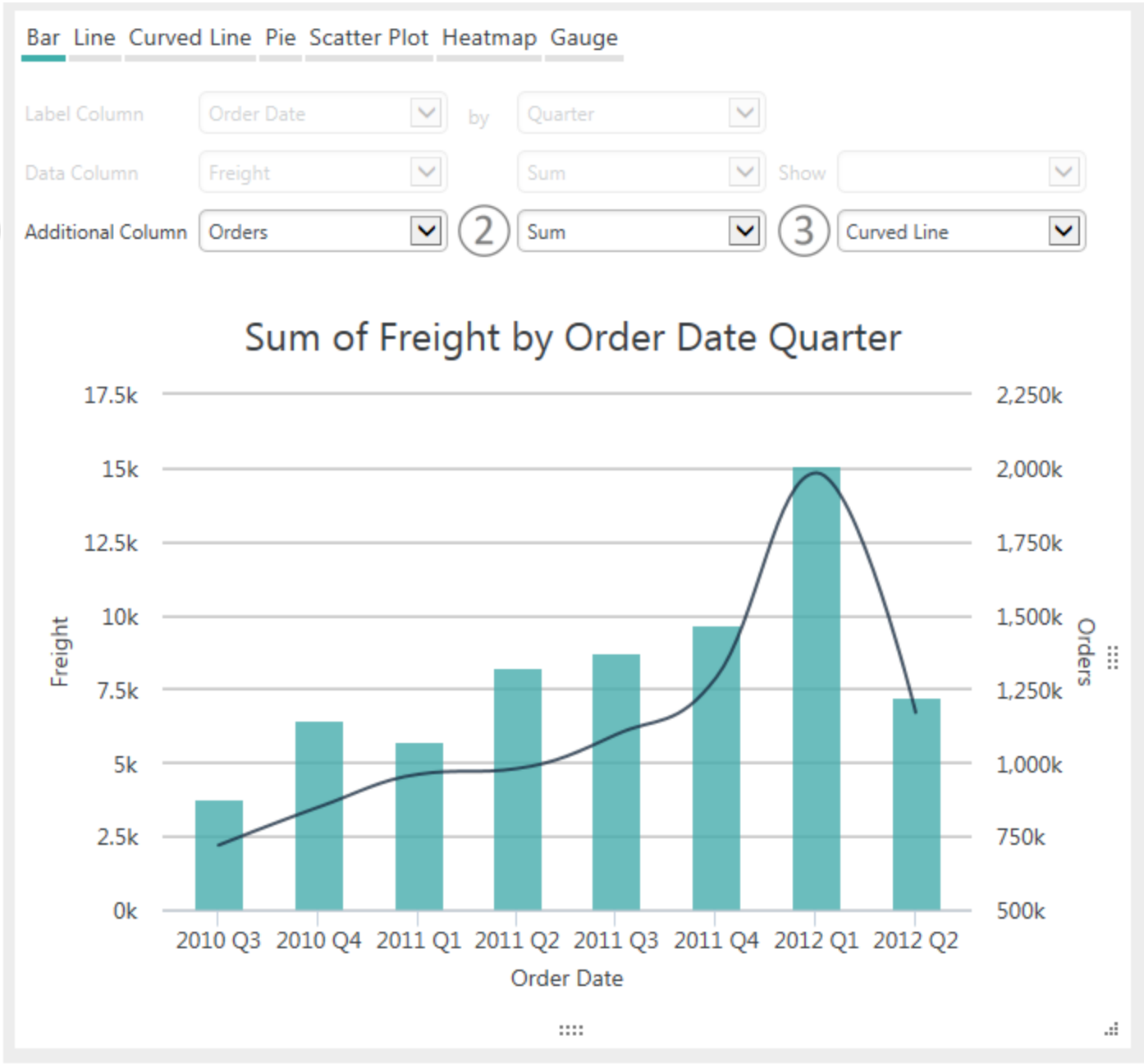
Studio includes a wizard to assist in adding Analysis Chart Columns. Select and right-click your Analysis Chart element, and use the context menus shown above to launch the wizard.



As shown above, the wizard lets you select the columns in the datalayer to use with the chart and to set their data types. The wizard can be launched by selecting the Analysis Chart element and then either clicking it in the main menu's **Wizards tab** or right-clicking and selecting it from the element's **Element Wizards** pop-up menu.

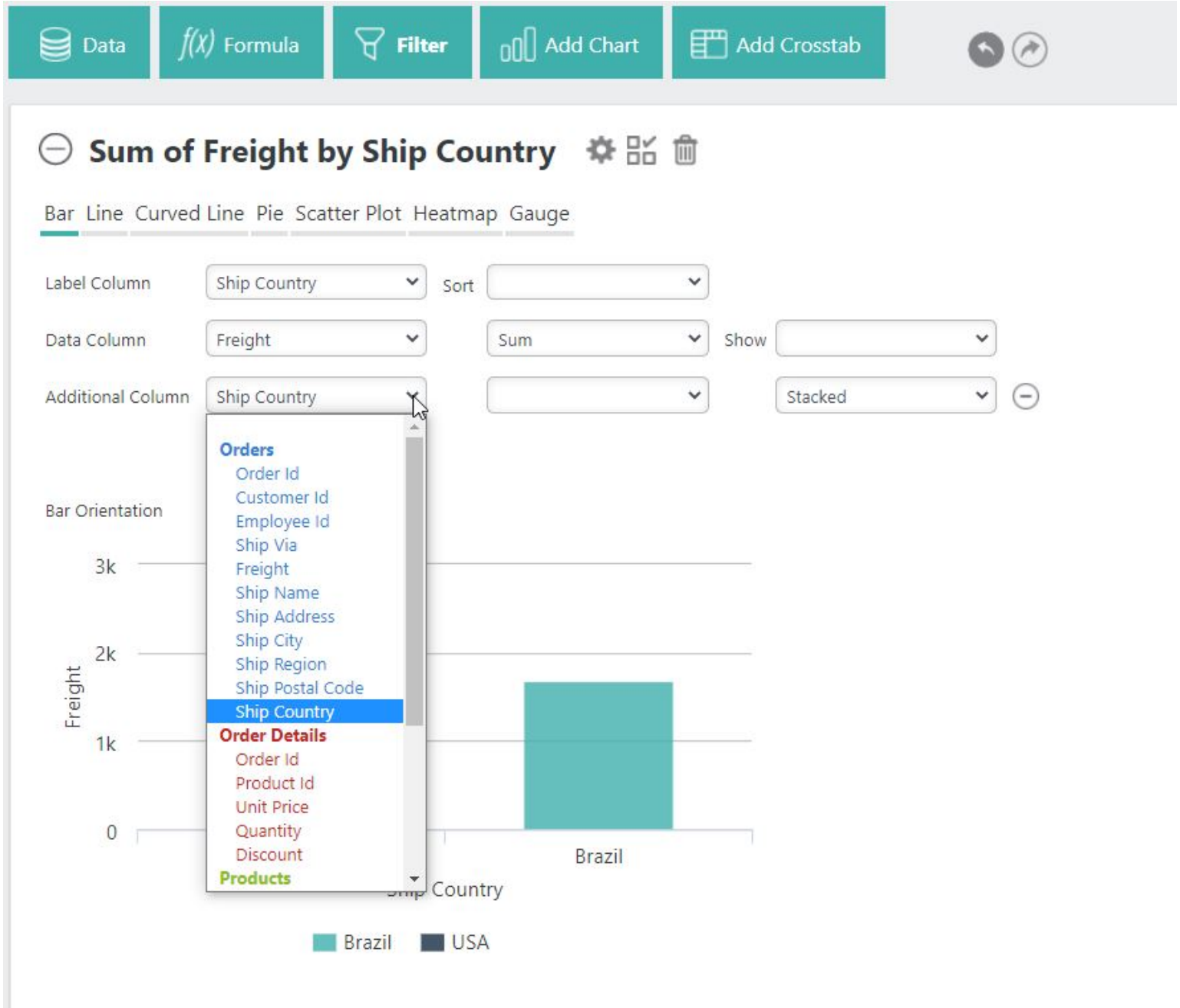
Analysis Chart - Adding Additional Series

It's frequently useful to add additional series to the chart, allowing easier visual comparisons. Controls are available for this:


Bar
Line
Curved Line
Pie
Scatter Plot
Heatmap
Gauge
Label Column
Order Date
by
Quarter
Data Column
Freight
Sum
Show
1
Additional Column
Orders
2
Sum
3
Curved Line
Sum of Freight by Order Date Quarter
17.5k
15k
12.5k
10k
7.5k
5k
2.5k
0k
Freight
2,250k
2,000k
1,750k
1,500k
1,250k
1,000k
750k
500k
Orders
2010 Q3
2010 Q4
2011 Q1
2011 Q2
2011 Q3
2011 Q4
2012 Q1
2012 Q2
Order Date

When the chart type (*Bar*, *Line*, *Curved Line*, *Scatter Plot*) supports it, the (1) Additional Column control will be available for you to use in selecting the Data Column for a second series. The second series (2) Aggregation and (3) Chart Type can be selected to suit your needs. A second axis and label will appear automatically along the side of the chart opposite the original Y-axis, as shown above.

To add an additional column to your chart, select the Additional Column drop-down menu:

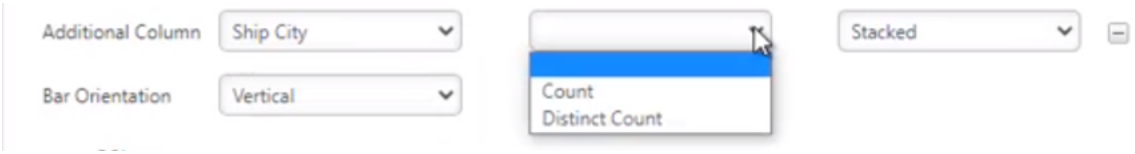

Data
Formula
Filter
Add Chart
Add Crosstab
Sum of Freight by Ship Country
Bar Line Curved Line Pie Scatter Plot Heatmap Gauge
Label Column
Ship Country
Sort
Data Column
Freight
Sum
Show
Additional Column
Ship Country
Stacked
Orders
Order Id
Customer Id
Employee Id
Ship Via
Freight
Ship Name
Ship Address
Ship City
Ship Region
Ship Postal Code
Ship Country
Order Details
Order Id
Product Id
Unit Price
Quantity
Discount
Products
Bar Orientation
3k
2k
1k
0
Freight
Brazil
Country
Brazil
USA

Then, select the column you want to add. Repeat these steps to add multiple additional columns.

Depending on the chart type, other controls will appear for use configuring additional series, including aggregation options and charting types. These controls allow you to choose between Stacked, Stacked Percentage, and Side By Side:



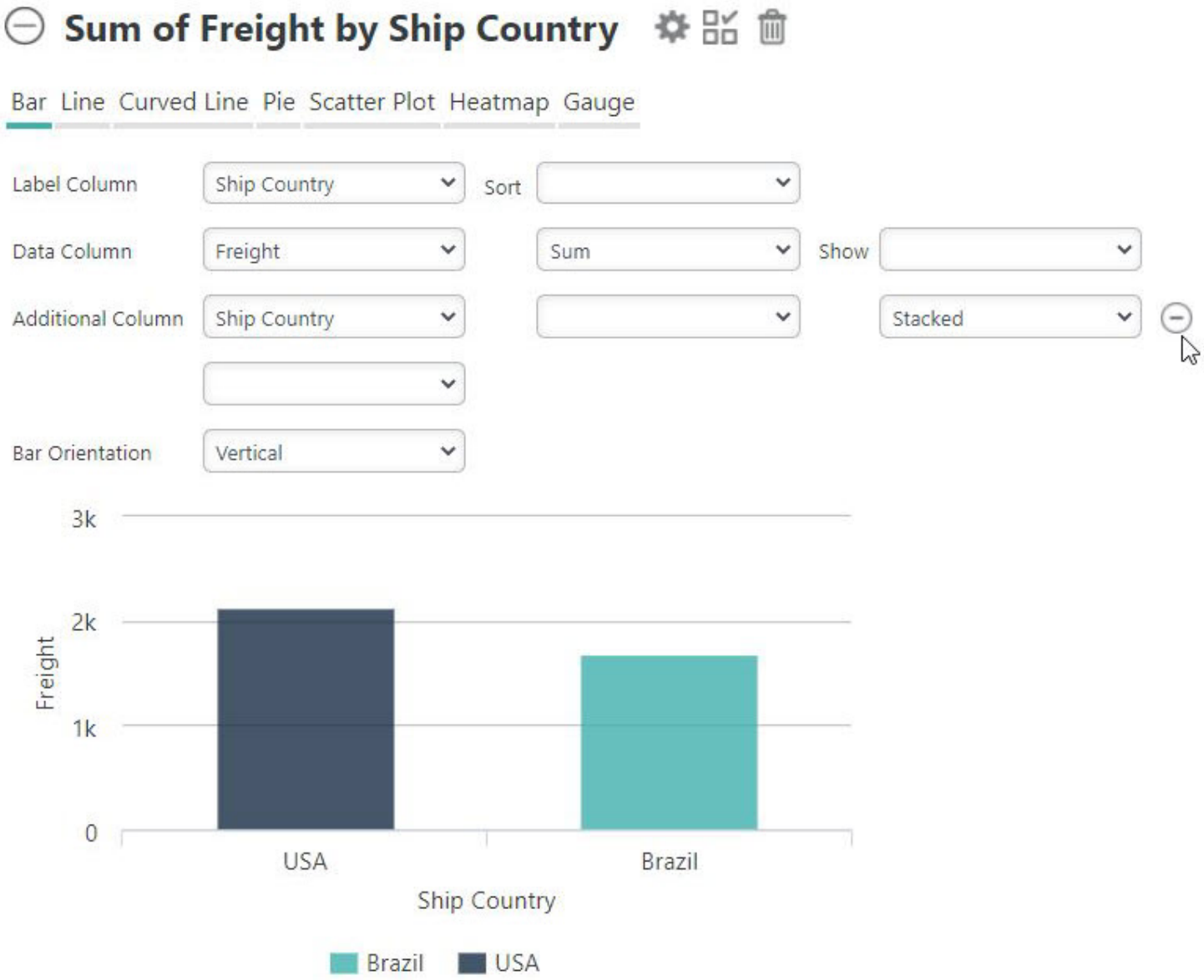
Aggregation options include Count, Sum, or Distinct Count:



Additional options for charts include Bar, Line, Curved Line, and Scatter Plot with Dual or Single Axis graphs:



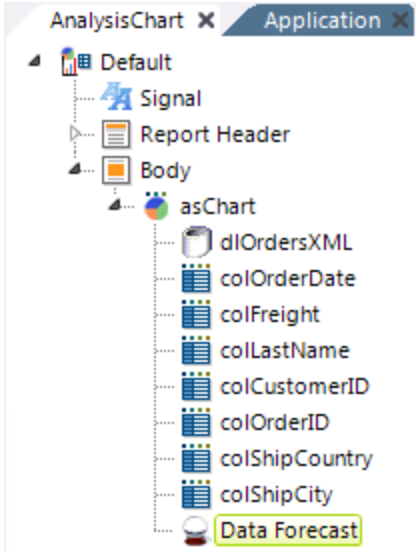
Remove the additional column(s) by selecting the **-** icon:


Sum of Freight by Ship Country
Bar Line Curved Line Pie Scatter Plot Heatmap Gauge
Label Column
Ship Country
Sort
Data Column
Freight
Sum
Show
Additional Column
Ship Country
Stacked
Bar Orientation
Vertical
3k
2k
1k
0
Freight
USA
Brazil
Ship Country
Brazil
USA

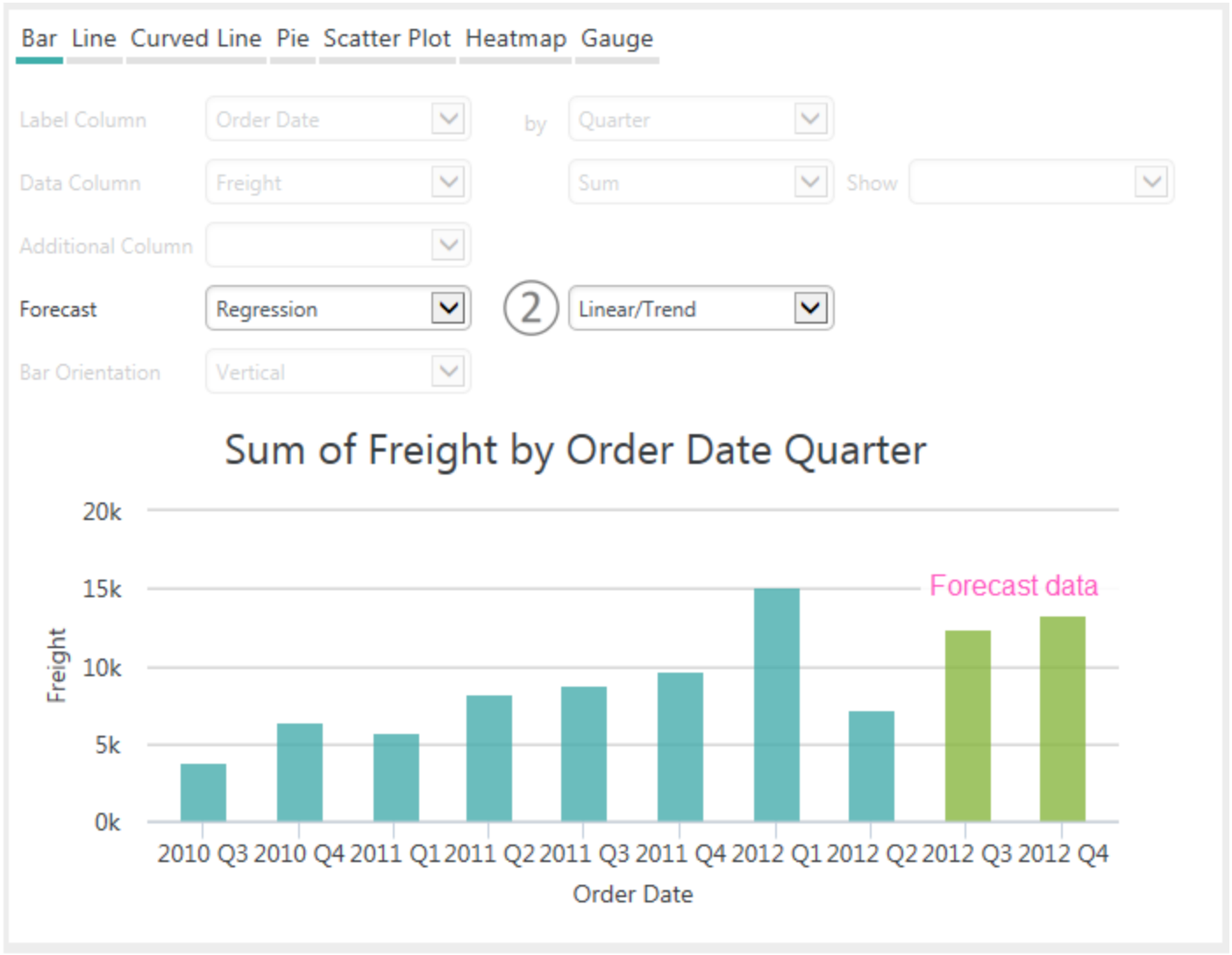
If you only have one additional column, selecting the blank option at the top of the Additional Column drop-down menu will also remove the additional column.

Analysis Chart - Adding Forecasting

Forecasting features are included in the Analysis Chart. These features use a variety of techniques to produce projected values by analyzing existing values. Forecasting is available for Bar, Line, and Curved Line chart types.



Forecasting is enabled in an Analysis Chart by including a **Data Forecast** element beneath it, as shown above. This element has no attributes.


Bar Line Curved Line Pie Scatter Plot Heatmap Gauge
Label Column
Order Date
by
Quarter
Data Column
Freight
Sum
Show
Additional Column
1
Forecast
Regression
2
Linear/Trend
Bar Orientation
Vertical
Sum of Freight by Order Date Quarter
20k
15k
10k
5k
0k
Freight
Forecast data
2010 Q3 2010 Q4 2011 Q1 2011 Q2 2011 Q3 2011 Q4 2012 Q1 2012 Q2 2012 Q3 2012 Q4
Order Date

When the Data Forecast element is present and an appropriate X-axis column is selected, a special selection list (1) of forecast options will become visible in the control panel, as shown above. When an option is selected, the forecast data is displayed in the example chart as an extra set of bars. In other chart types, it might appear as a dotted line.

If the X-axis column is of DateTime type, the data may be analyzed using either **Time Period Decomposition** or **Regression** analysis; columns of Numeric type are analyzed using Regression analysis. An additional option list (2) allows selection of a time span for Time Period Decomposition analysis, or a regression method for Regression analysis. Available regression methods include:

- Linear/Trend
- Autoregressive (using "Burg's" formula)
- Exponential
- Logarithmic
- Polynomial (orders 2 - 5)
- Power

The forecast options also include **Trend Line**, with straight or curved presentation options.

he Studio wizard for creating an Analysis Chart also includes forecasting options. More information about forecasting is available in "Forecast Methodologies" on page 473.

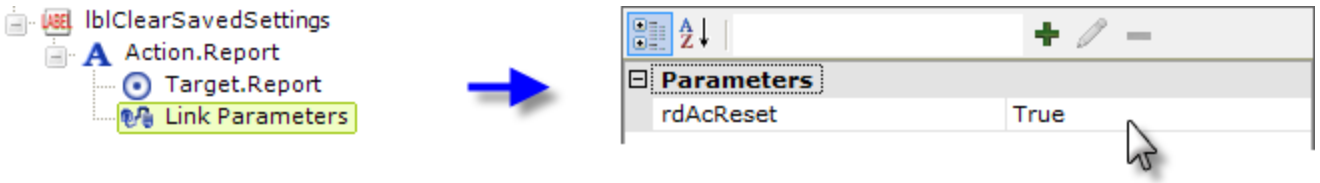
Analysis Chart - Saving Individual User Settings

The Analysis Chart will automatically **save** a user's runtime configuration changes during their session, if the **Save File** attribute is set. The file specified will be created if it does not exist but its containing folder is not automatically created. You must ensure that the account used by the web server to run the application has "Write" permission to the folder.

Save files can also be stored in a database, see *Bookmarks*.

You can also save separate configurations for each user that survive their session. If Logi Security is being used, include the @Function.UserName~ token as part of the Save File name. If not, use @Function.GUID~ to generate a unique file name for the user and save the file name as a cookie in the user's browser.

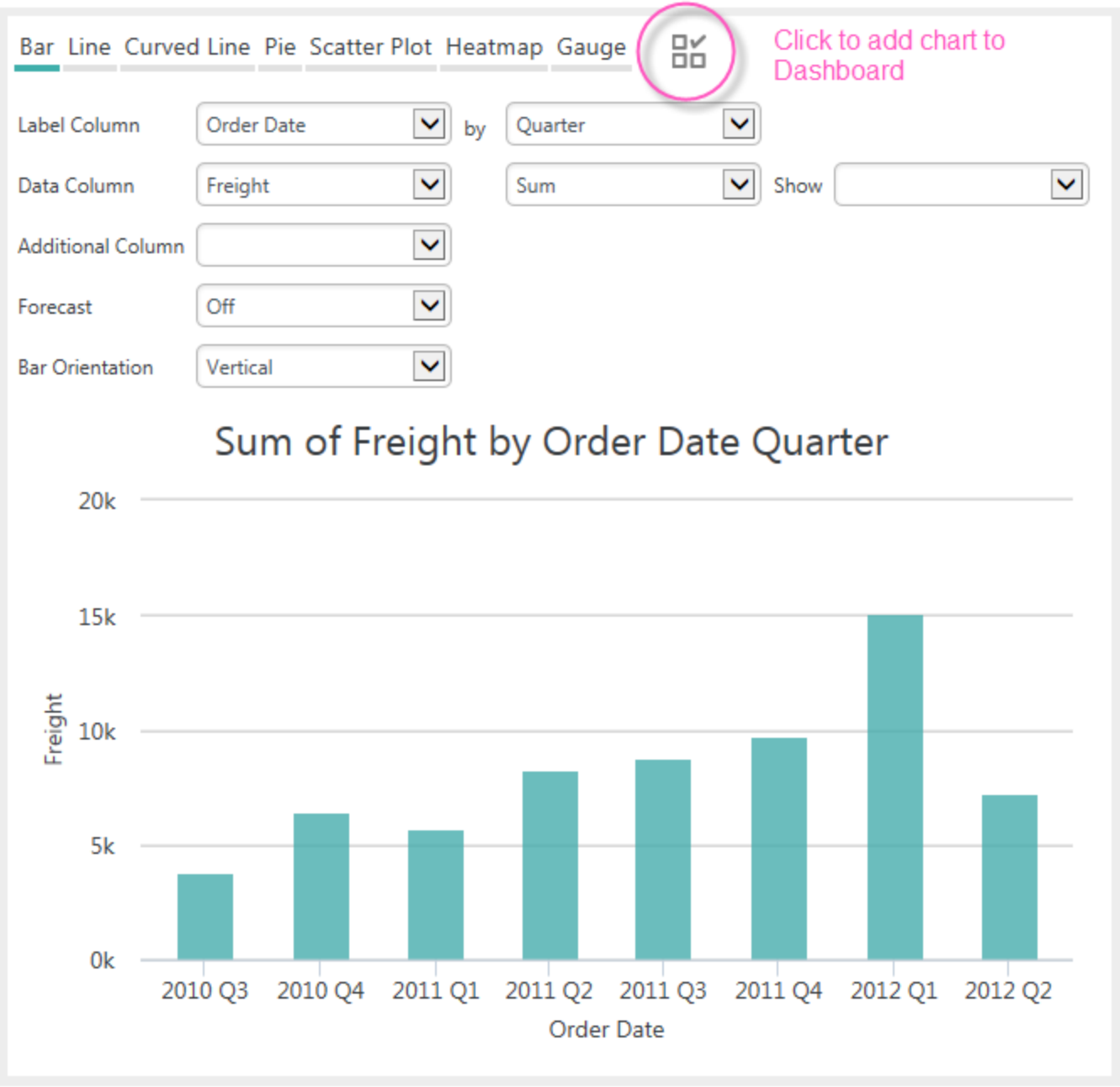
Users' settings can be cleared by calling the Analysis Chart report with the **rdAcReset** parameter set to *True*. To get fresh data from the datasource while maintaining the user's settings, call the report with the **rdAcRefreshData** parameter set to *True*.



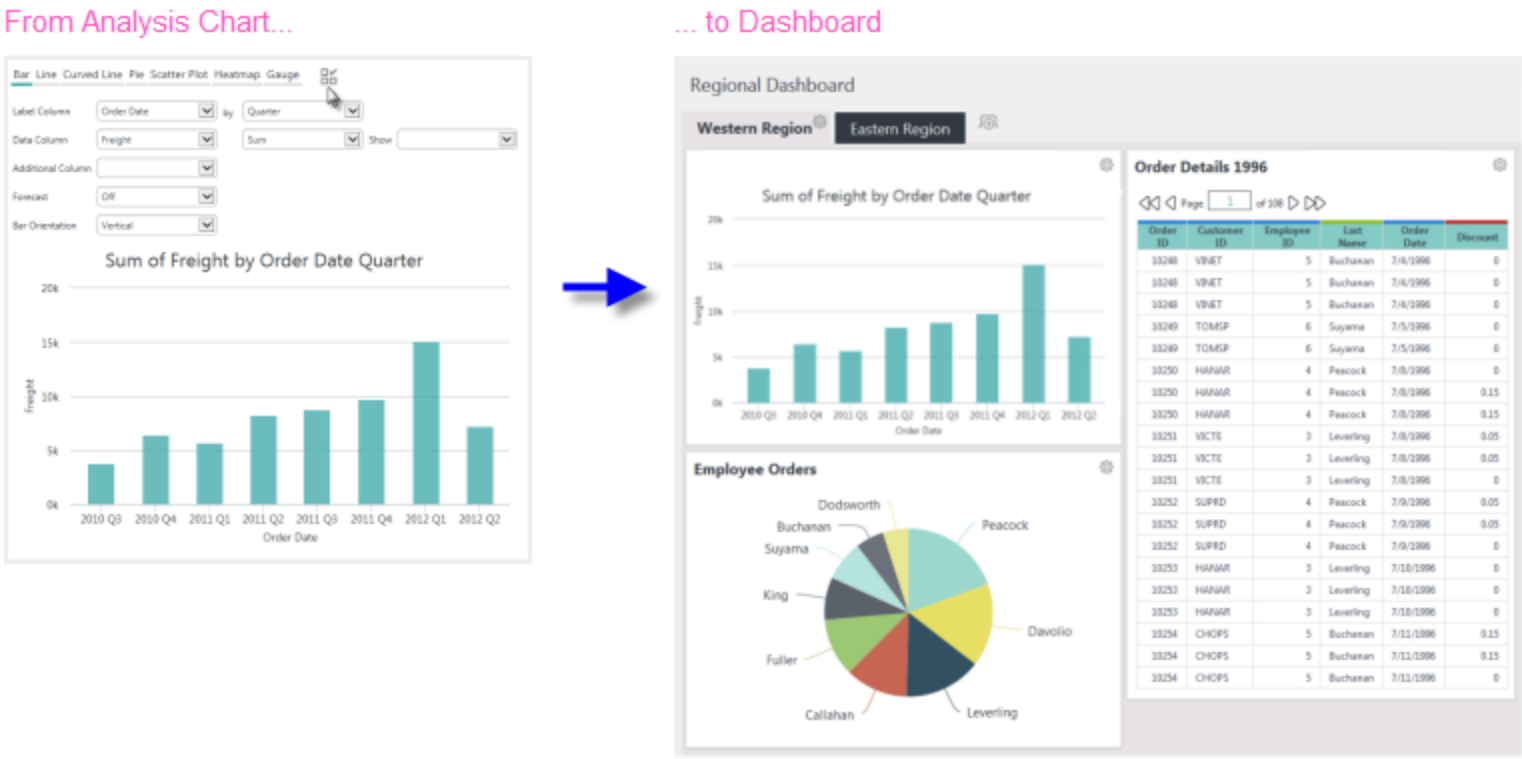
In the example above, a Label has been added to an Analysis Report as a link that clears the user's settings. The Target.Report element is set to *Current Report* and the Link Parameters element is used to pass the rdAcReset parameter.

Analysis Chart - Adding Charts to Dashboards

An exciting level of interaction between the Analysis Chart and the **Dashboard**element is available. Users working with an Analysis Chart in one report, at runtime, can generate a chart and then, with one mouse click, add it as a new panel in an existing Dashboard in another report.


Bar
Line
Curved Line
Pie
Scatter Plot
Heatmap
Gauge
Click to add chart to Dashboard
Label Column
Order Date
by
Quarter
Data Column
Freight
Sum
Show
Additional Column
Forecast
Off
Bar Orientation
Vertical
Sum of Freight by Order Date Quarter
20k
15k
10k
5k
0k
Freight
2010 Q3
2010 Q4
2011 Q1
2011 Q2
2011 Q3
2011 Q4
2012 Q1
2012 Q2
Order Date

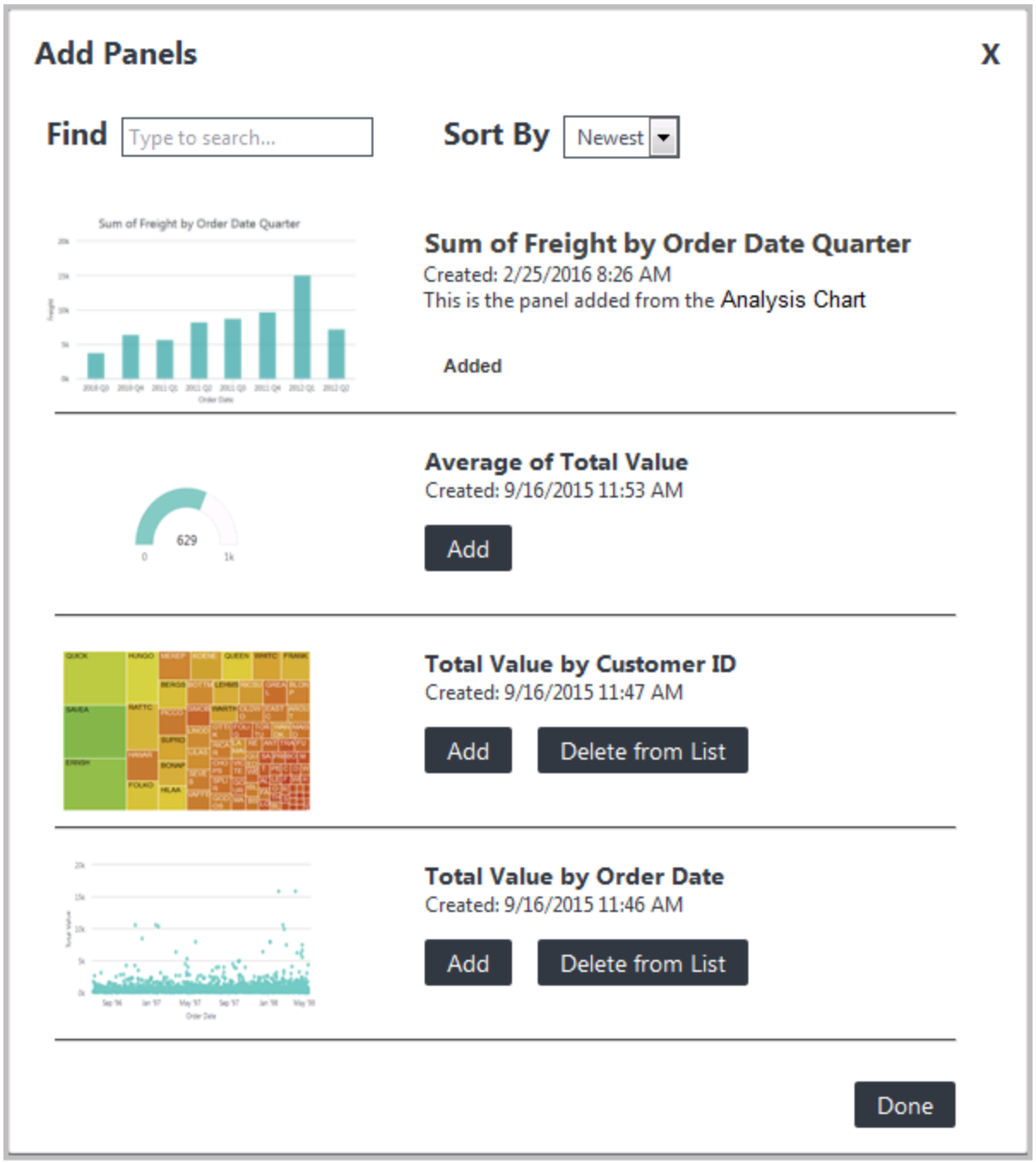
When properly configured, the Analysis Chart will display an **Add To Dashboard** icon, as shown above.



When the icon is clicked, the chart is added as a new panel in the Dashboard, as shown above.

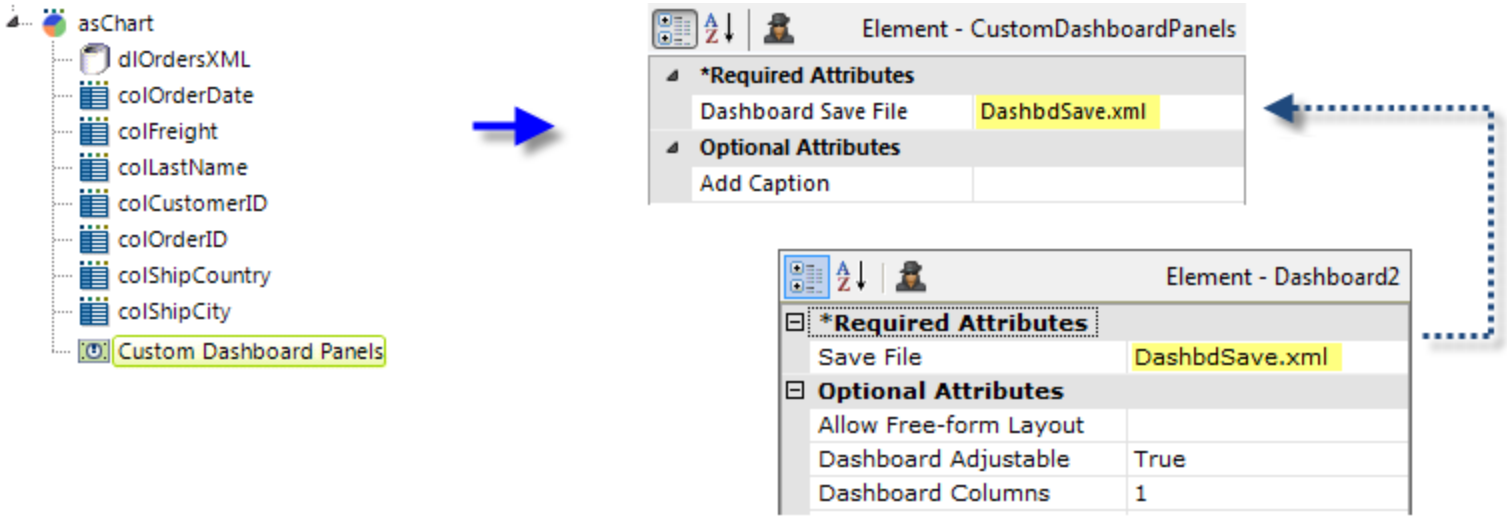
Behind the scenes, when the icon is clicked, the Analysis Chart writes the underlying XML for its chart into the Dashboard's configuration file (its "Save File"). In the Dashboard, the chart is inserted at the top of Column 1; if Dashboard tabs are being used, the insertion occurs in the currently active tab.

Just before panel insertion, the user will be prompted for the **Panel Title** and a description for display on the Configuration Page.


Add Panels
X
Find
Type to search...
Sort By
Newest
Sum of Freight by Order Date Quarter
Sum of Freight by Order Date Quarter
Created: 2/25/2016 8:26 AM
This is the panel added from the Analysis Chart
Added
Average of Total Value
Created: 9/16/2015 11:53 AM
629
0
1k
Add
Total Value by Customer ID
Created: 9/16/2015 11:47 AM
Add
Delete from List
Total Value by Order Date
Created: 9/16/2015 11:46 AM
Add
Delete from List
Done

The new chart panel thereafter appears in the Dashboard Configuration Page, as shown above, just like any other panel, complete with a thumbnail image. The new panel can be removed from the visible Dashboard panels and from the configuration page entirely, using the usual controls.

The user can insert multiple tables and charts into a Dashboard using this technique.



In the report definition, the developer makes this functionality available by adding a **Custom Dashboard Panels** element, shown above, beneath the Analysis Chart element. Its **Dashboard Save File** attribute value should be set to match the target Dashboard's **Save File** attribute value.

The Save File attribute values shown above have been shortened for visual clarity. However, a fully-qualified path and file name, with .xml extension, is required, so a realistic attribute value would be something like:

```
@Function.AppPhysicalPath~\DashFiles\DashbdSave.xml
```

and the account the Logi application runs under must be given **Write**permissions to the "DashFiles" folder. For experimentation purposes, you can use the rdDownload folder, for which appropriate permissions already exist (but its contents are periodically deleted, so don't use it for anything but experimentation).

Dashboard Save Files are often named using a token, such as @Function.UserName~, in order to create personalized Dashboard configurations and, in that case, the Custom Dashboard Panel attribute would use the same token.

The Custom Dashboard Panels element has a new attribute, **Add Caption**, which sets the caption for the button used to save the Data Table or chart. If left blank, the caption is "Add to Dashboard".

For more information about Dashboards, see *Dashboard for End Users*.

Analysis Chart - Customizing Analysis Chart Appearance

Analysis Chart appearance can be changed most easily by applying a theme to the report definition. Most of the screen shots in this document were taken with the *Signal* theme applied.

You can create your own custom theme, based on a standard theme, *Using the Theme Editor* tool.

Changing Appearance Using Style Classes

Analysis Chart appearance can be also customized using **style classes** and Info Studio provides the following standard style sheet for all Analysis Charts:

```
<YourAppFolder>\rdTemplate\rdAnalysisChart\rdAcStyle.css
```

Developers can override classes in this style sheet by *copying* them to their application style sheet and modifying them there.

 Do not make changes in the standard style sheet in the rdTemplate folder.

Changing Appearance Using Template Modifiers

The Analysis Chart element uses a "template file" to define certain element properties that are not otherwise available as attributes to the developer for modification. These include language- and culture-specific **Caption** attributes that you may want to change to match the locale (or you may simply want to change the captions to better suit your application).

The Analysis Chart element's **Template Modifier File** attribute identifies a custom XML file developers can create containing elements that will override the same elements in the template file.

For example, the Analysis Chart template file:

`<yourAppFolder>\rdTemplate\rdAnalysisChart\rdAcTemplate.lgx`

contains several Label elements. One of them has an ID of "lblChartXLabelColumn_rdAcID"; this controls the caption text that appears beside the Label Column input select list. This text can be modified by changing the Caption associated with that Label element. To change the text from its default "Label Column" to "X-axis Data Column", create your own XML file, identify it in the Template Modifier File attribute, and add this code to it:

```
<TemplateModifier>
<SetAttribute ID="lblChartXLabelColumn_rdAcID" Caption="X-axis Data Column" />
</TemplateModifier>
```

You can set the attributes for any number of elements in this file; examine the rdAcTemplate.lgx file to learn the ID and Caption attributes available. The template modifier file can be in any folder accessible to the web application; if a fully-qualified file path is not provided in the Template Modifier File attribute value, then the application expects the file to be in its _SupportFiles folder.

More detailed information about template modifier files can be found in Template Modifier Files.

The Analysis Filter

The Analysis Filter provides a mechanism that allows users to filter data at runtime and immediately updates visualizations dependent on that data.

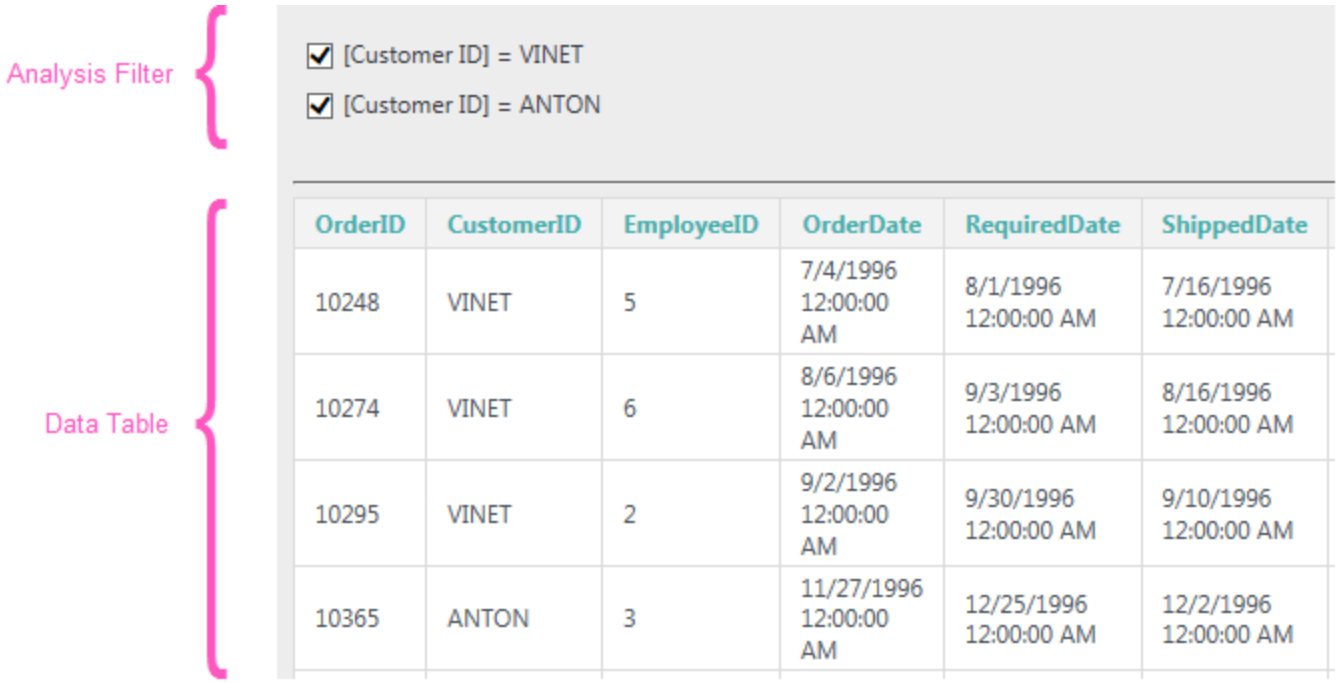
The following topics introduce developers to the **Analysis Filter** super-element:

- Using the Analysis Filter Wizard
- Analysis Filter Attributes
- Analysis Filter Column Attributes
- Using the Analysis Filter Elements
- Saving Individual User Settings
- Filtering with Query String Parameters
- Using Analysis Filters in Dashboards
- Customizing Analysis Filter Appearance

About the Analysis Filter

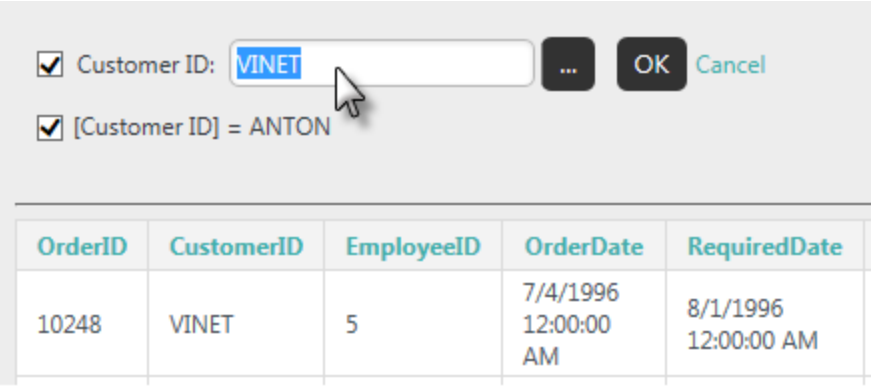
The **Analysis Filter** allows users to apply filtering, at runtime, to the datalayers that support their visualizations. Changes made to the filtering immediately causes the visualizations to be updated.

As a super-element, the Analysis Filter has an interface that users can interact with at runtime. The interface can be configured and restricted depending on the use-case requirements.

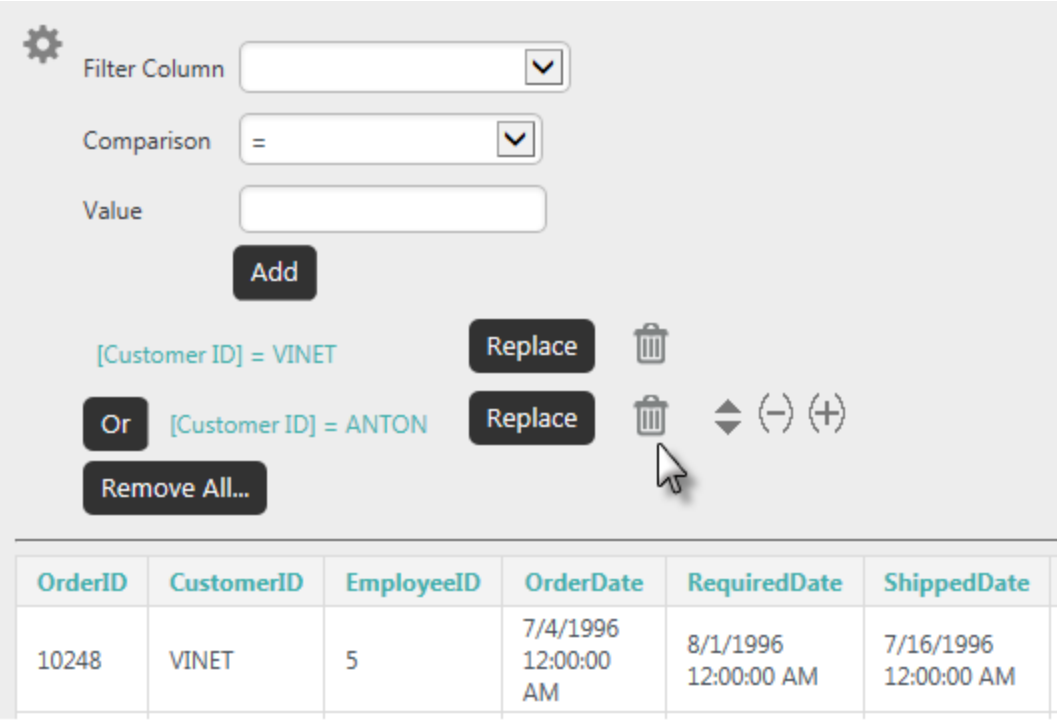


OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate
10248	VINET	5	7/4/1996 12:00:00 AM	8/1/1996 12:00:00 AM	7/16/1996 12:00:00 AM
10274	VINET	6	8/6/1996 12:00:00 AM	9/3/1996 12:00:00 AM	8/16/1996 12:00:00 AM
10295	VINET	2	9/2/1996 12:00:00 AM	9/30/1996 12:00:00 AM	9/10/1996 12:00:00 AM
10365	ANTON	3	11/27/1996 12:00:00 AM	12/25/1996 12:00:00 AM	12/2/1996 12:00:00 AM

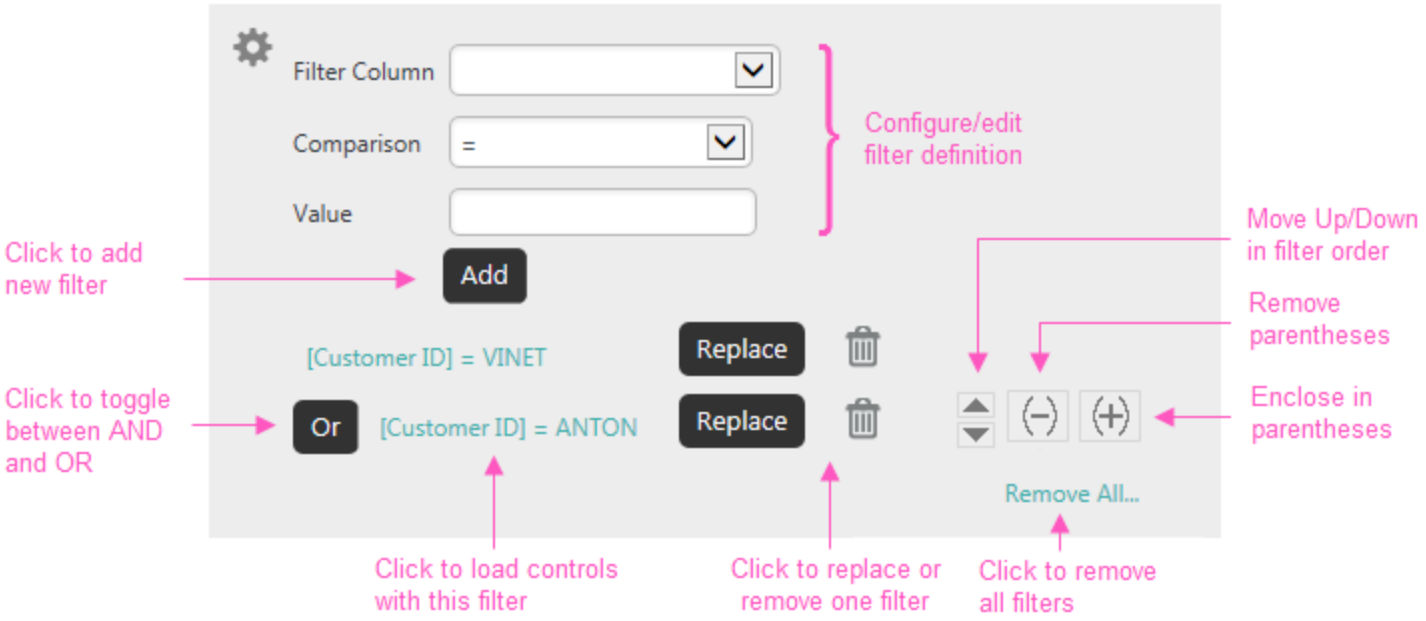
The example above shows the Analysis Filter in use with a Data Table. If the user checks or unchecks one of the filters, the Data Table will be refreshed accordingly (the refresh can include the entire page or just selected elements). The Analysis Filter has two developer-controlled viewing modes: Simple and Design.



In **Simple** view, shown above, users can click an existing filter description to display input controls and change the filtering value.

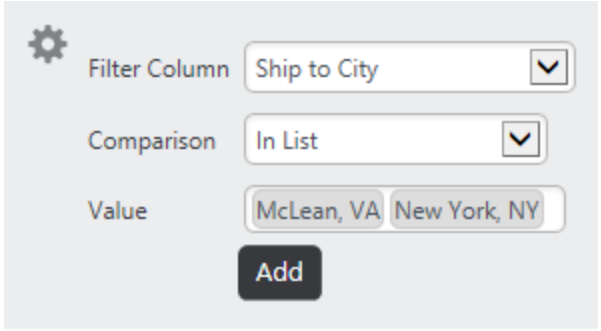


In **Design** view, a "gear" icon is displayed; clicking it opens a control panel, shown above, that can be used to manage filters.

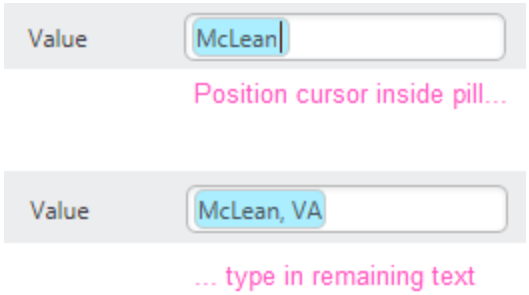


The controls provide a variety of tools for creating, editing, grouping, and removing filters, as shown above.

When using "In List" or "Not In List" comparisons in filters, multiple values can be entered, separated by commas. However, the values themselves may *include* commas, causing incorrect comparisons.

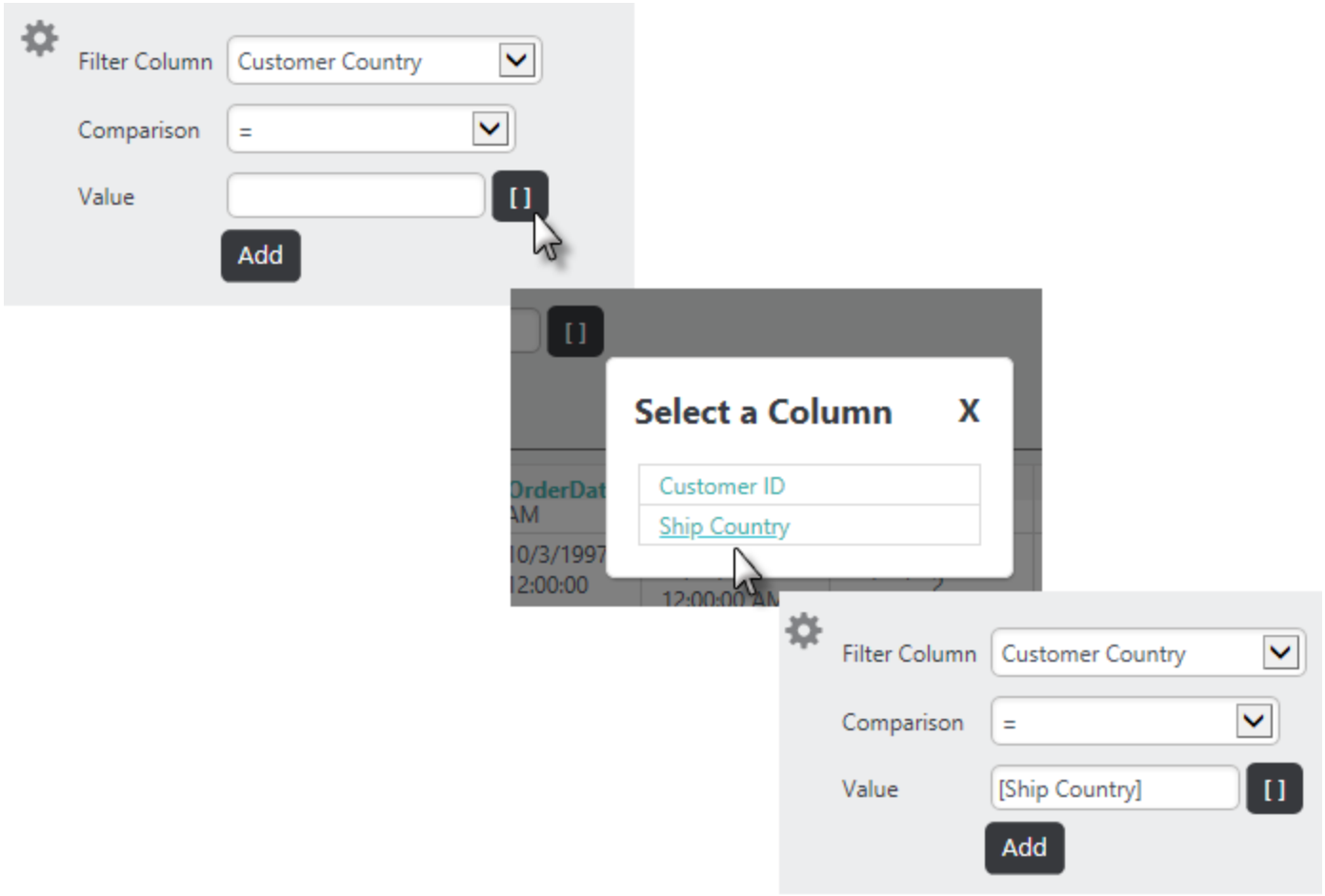


To address this, values for these types of filters are now represented in the controls by visual "pills" that enclose the complete value, as shown above. The filter uses the complete string within the pill during its comparison operation.



Pills with included commas are created by entering the first part of the value and pressing comma. This creates a blue pill as shown above. Then place your cursor *inside* the pill and type the rest of the value. Press tab to exit the pill and repeat the process for the next value, if desired.

The Analysis Filter is usually used to compare column data and a value you enter, but you can also directly compare the values in two data columns:



To do this, first select a Filter Column, as shown above, and a supported Comparison operator: *=, <, <=, >, Not =, In List, Not In List, Starts With, Contains, Not Starts With, Not Contains, Like, Not Like*.

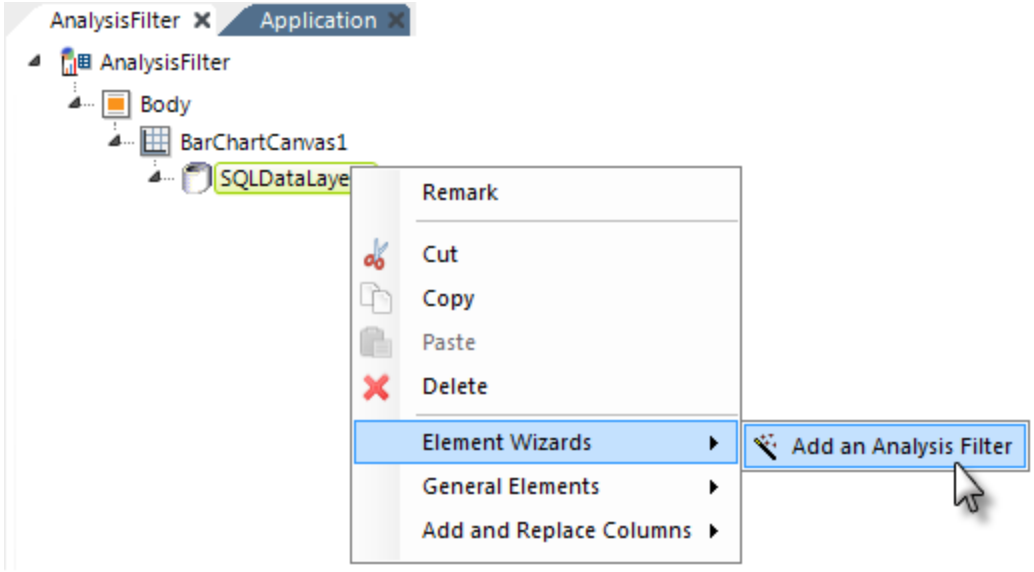
Then select the second column from the list displayed when you click the **[ ]** button, or enter the column name within square brackets.

Retaining User Settings

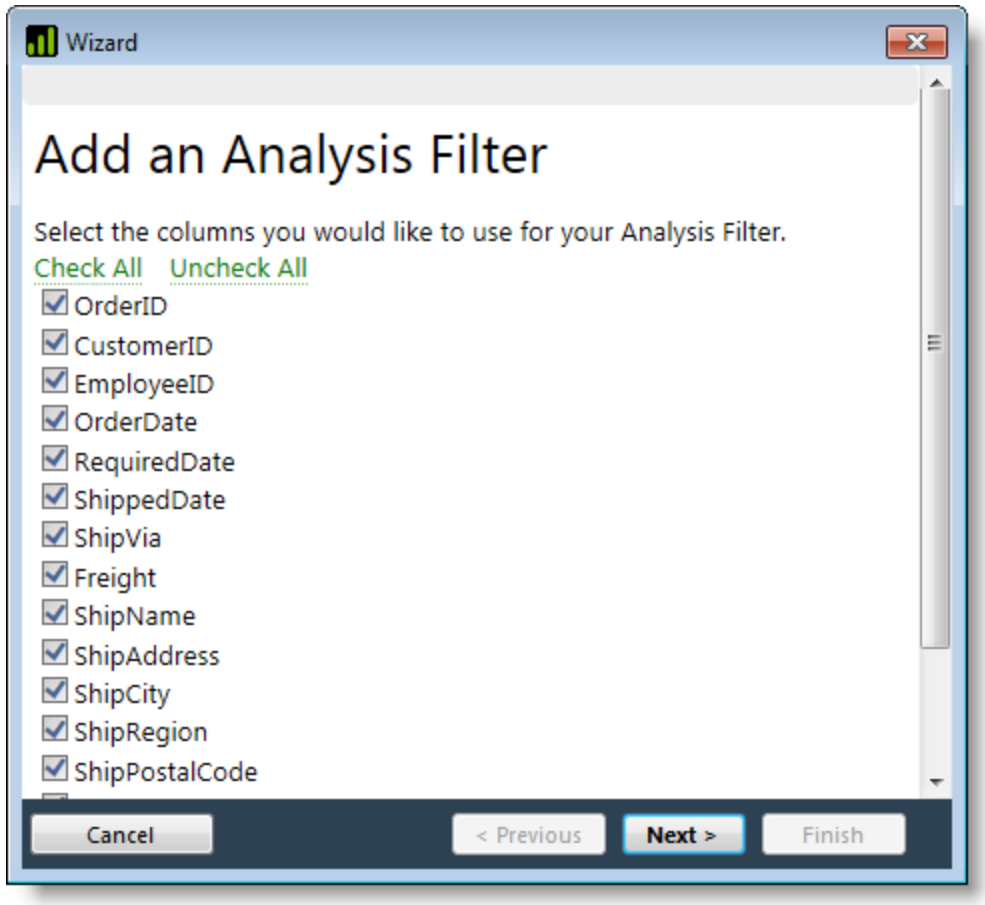
Filters users create or modify can be saved between sessions and made available to them during their next session (this is discussed in more detail below).

Analysis Filter - Using the Analysis Filter Wizard

Logi Studio includes a wizard that can assist you in creating an Analysis Filter. The wizard assumes that you've already added an appropriate Connection element in the _Settings definition and configured it, and have added a DataLayer element that uses it.



As shown above, the wizard can be started by selecting and then right-clicking most datalayer elements, and using the context menus to select "Add an Analysis Filter". The wizard will open:



Select the columns that you want to be able to use for filtering the data. Columns that are *not* selected here will *not* appear as filtering choices in the Analysis Filter interface. Click **Next**, and then **Finish**.

The wizard will insert the necessary Analysis Filter family elements.

The wizard may also insert other elements, if they're necessary and not already in place. For example, if you run the wizard by selecting a datalayer in a Dashboard panel and there is no Panel Parameters element already in use, the wizard will insert one and then insert the Analysis Filter elements under it.

Analysis Filter - Attributes

The Analysis Filter element has the following attributes:

Attribute	Description
ID	(Required) Specifies a unique identifier for this element.
Case Sensitive	Specifies whether comparisons will be case-sensitive. For elements using DataLayer.ActiveSQL, this attribute value may be set to *DataSourceCollation*, causing case-sensitivity to be defined by the database column's "collation", which may be case-sensitive or not. This option can provide better performance for case-insensitive filters. For an Analysis Filter used with DataLayer.Active SQL and Sql Compare Filter elements, the default is *DataSourceCollation*. The default is *True* for all other elements.
Disable Design View	Specifies whether to display the filter in Design view mode. This allows the user full control of filter *creation* and updates. Normally, a report run without an Analysis Filter Save File specified initially appears in Design view. When redisplayed, the report's Analysis Filter appears in Simple view. Disabling Design view causes the Analysis Filter to appear only in the Simple view. Default Value: *False.*
Disable Simple View	Specifies whether to display the filter in Simple view mode. This allows the user to do easy updates of existing filters. Normally, a report run without an Analysis Filter Save File specified initially appears in Design view. When redisplayed, the report's Analysis Filter appears in Simple view.

Attribute	Description
	Disabling Simple view causes the Analysis Filter to appear only in Design view. Default Value: *False.*
Filter Caption Element ID	Specifies the ID a Label element that will display a one-line description of the filter(s) in use, allowing filtering criteria to be shown anywhere in the report page. To use it, add a Label element to the report and give it a unique ID, leave its Caption attribute blank, and specify its ID in this attribute. The Label's Caption will be filled at runtime with the filter's description/caption. For example: [Customer ID] = VINET
Filter List Location	Specifies whether, when using the Design view, the list of defined filters appears beneath the input controls or beside them to the right. This allows management of the space the filter uses. Default Value: *Bottom* (beneath the controls)
No Filters Simple Caption	Specifies text to be displayed in Simple view when no filters have been defined. Default value: *(No filters)*
Refresh Element IDs	Specifies, in a comma-separated list, the IDs of the elements to be refreshed within the page, using AJAX, when a filter is checked or unchecked or a filter definition or value is changed. If left blank, then the entire page will be refreshed when a filter value changes.
Request Forwarding	Set to *True* to pass all request parameters received by this page to the next page. Values set using a Default Request Values element are not forwarded. Default value: *False*.  Request parameters cannot be forwarded if there is an Input element with the same name on the page,

Attribute	Description
	because this would result in an attempt to pass two values with the same name.
Save File	Specifies the fully-qualified file path and name of an XML file where user runtime filter changes will be saved. Must include the .xml file extension and can use an @Function token. Example: `@Function.AppPhysicalPath~\Filters\AFSaveFile.xml` Example: `@Function.AppPhysicalPath~\UserFilters\@Function.UserName~.xml` Save files can also be stored in a database, see Storing Bookmark, Gallery, and Save Files in a Database.
Save File Starter	Specifies the fully-qualified file path and name of an XML file to be used as the initial Save File in cases when the Save File does not exist yet. For example, when the Save File is different for each user, the Save File Starter file can include a set of default filters. Create a Save File Starter file by setting a Save File value, then defining filters to create the desired default set, then moving and/or renaming the Save File into the Save File Starter location. Example: `@Function.AppPhysicalPath~\Filters\InitialFilters.xml`
Security Right ID	If specified, access to this element is controlled via Logi security. Specify the ID of a Right defined in the application's settings/security section. Only users that Right will be able to see the element. Multiple Right IDs may be entered in a comma-separated list. In that case, the element will be visible to users with any one of the Rights specified.
Template Modifier File	Specifies the full name of a template modifier file to be used to alter the Analysis Filter element. See "Analysis Filter - Using the Analysis Filter Elements" on page 184 for more information about template modifiers. The

Attribute	Description
	template modifier file can be in any folder accessible to the web application; if a fully-qualified file path is not provided in this attribute value, then the application expects it to be in your project's _SupportFiles folder.

The columns that can be selected in the Design view control panel when creating filters are defined by **Analysis Filter Column** elements, one per column, that are added as children of the Analysis Filter element.

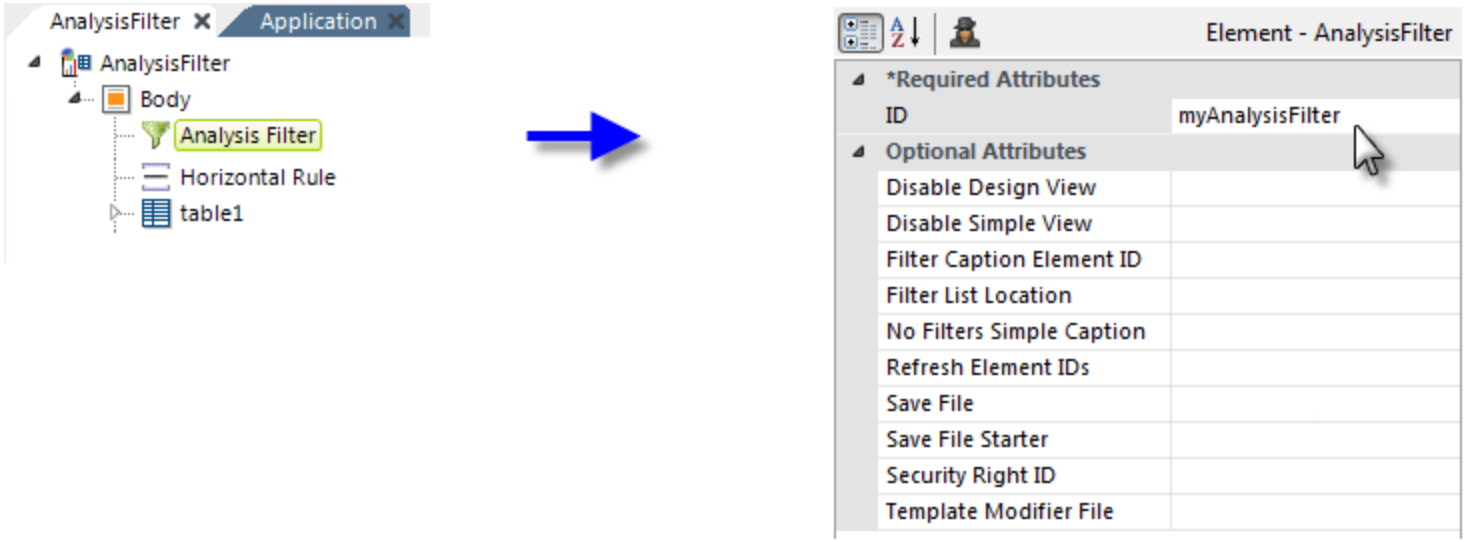
Analysis Filter - Column Attributes

The Analysis Filter Column element has the following attributes:

Attribute	Description
Column Header	(Required) Specifies the text describing the column that will appear in the list of columns that can be selected when defining a filter in Design view. This text also appears in the filter description, between square brackets, in Simple view. Allows you to specify a "user-friendly" name for the column.
Data Column	(Required) Specifies the name of a column in the datalayer.
Data Type	(Required) Specifies the data type of the column specified in the previous attribute.
ID	(Required) Specifies a unique identifier for this element.
Format	Specifies a format for values or percentages displayed. More information can be found in *Format Data*.
Popup Values for Filter	Specifies text to be displayed in Simple view when no filters have been defined. Default value: *(No filters)*
Security Right ID	If specified, access to this element is controlled via Logi security. Specify the ID of a Right defined in the application's settings/security section. Only users that Right will be able to see the element. Multiple Right IDs may be entered in a comma-separated list. In that case, the element will be visible to users with any one of the Rights specified.

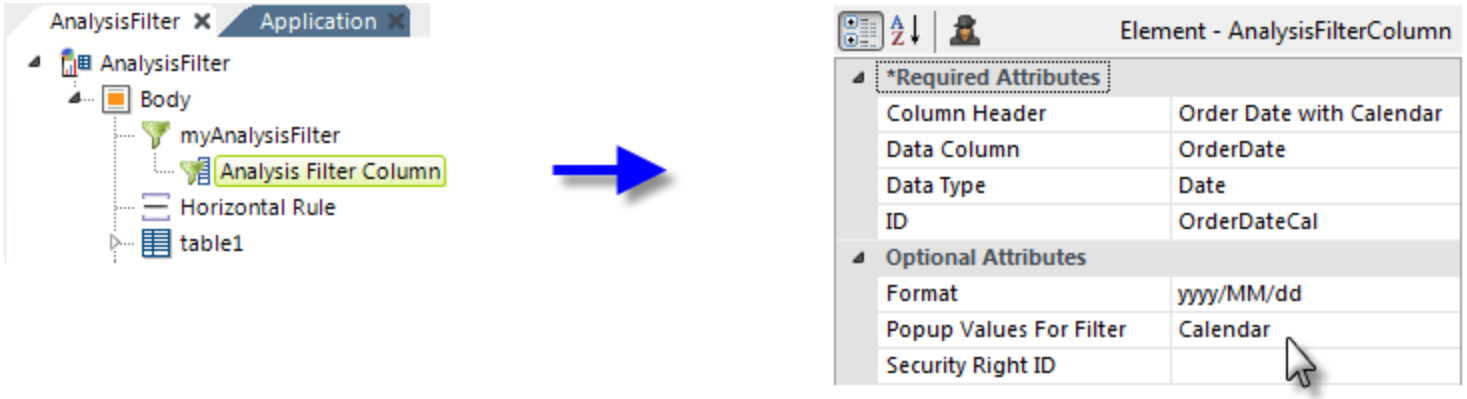
Analysis Filter - Using the Analysis Filter Elements

The Analysis Filter works by adding an automatically-generated Condition Filter element beneath existing datalayer elements (Sql Condition Filter for DataLayer.ActiveSQL). To implement it, you need to add an **Analysis Filter** element (and its child elements) and the **Analysis Filter Insert** element. Here's an example:

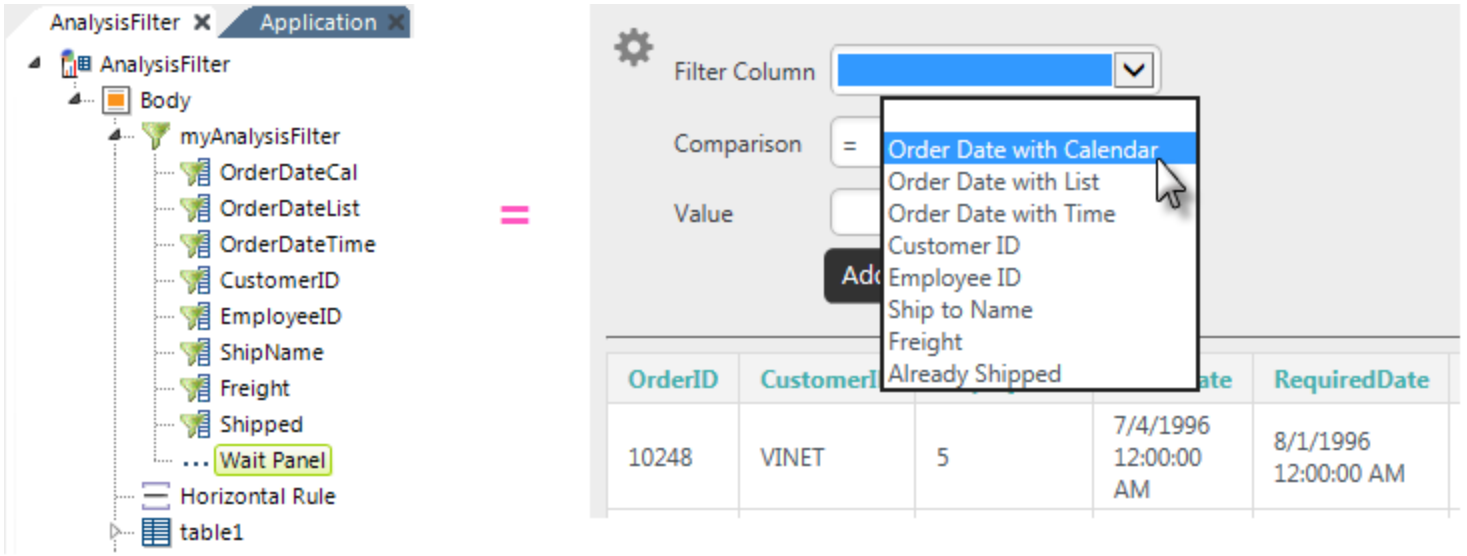


The report definition shown above includes a Data Table, with datalayer and columns. An Analysis Filter element has been added and configured with an element ID. This configuration will cause the whole page to be refreshed when the filter changes.

If, instead, we wanted to refresh only the Data Table and not the whole page, we could enter its element ID in the **Refresh Element IDs** attribute.

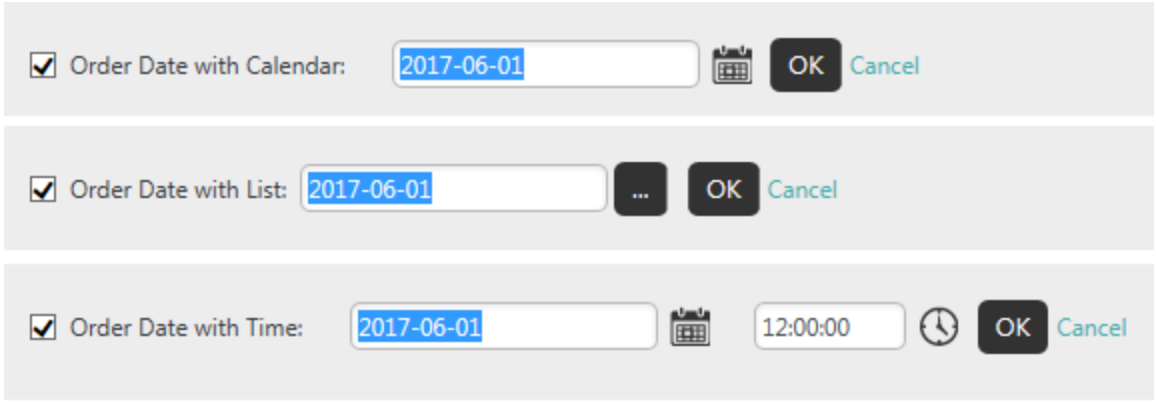


Next, we'll add **Analysis Filter Column** elements beneath the Analysis Filter element. Each of these adds a column into the list of columns that can be used when defining a filter in Design view. It's important to accurately specify the data type as this affects which user controls are displayed later.

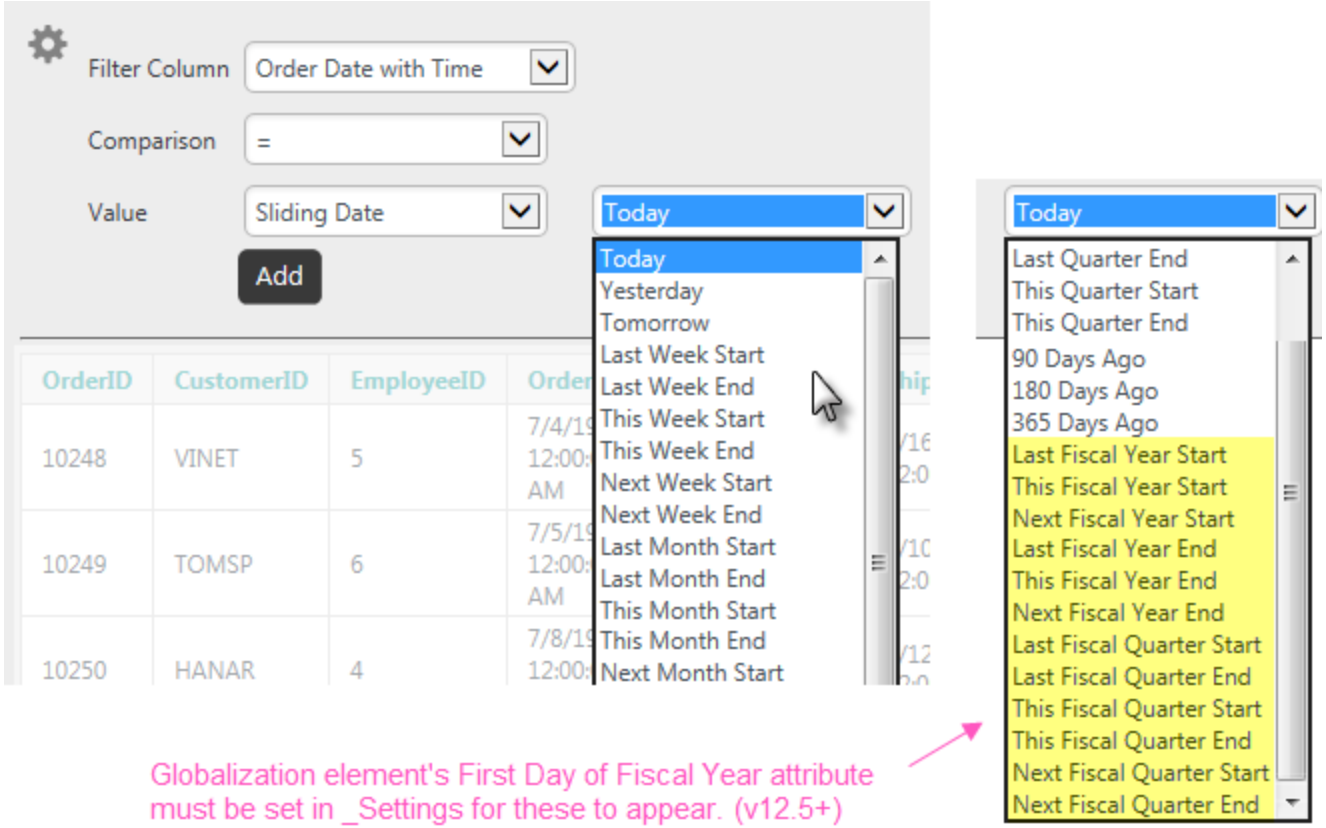


Additional Analysis Filter Column elements have now been added and, as shown above, you can see how they produce entries in the drop-down list of filter columns in Design view. In addition, an optional Wait Panel element has been added, in case the filtering operation takes some time to complete.

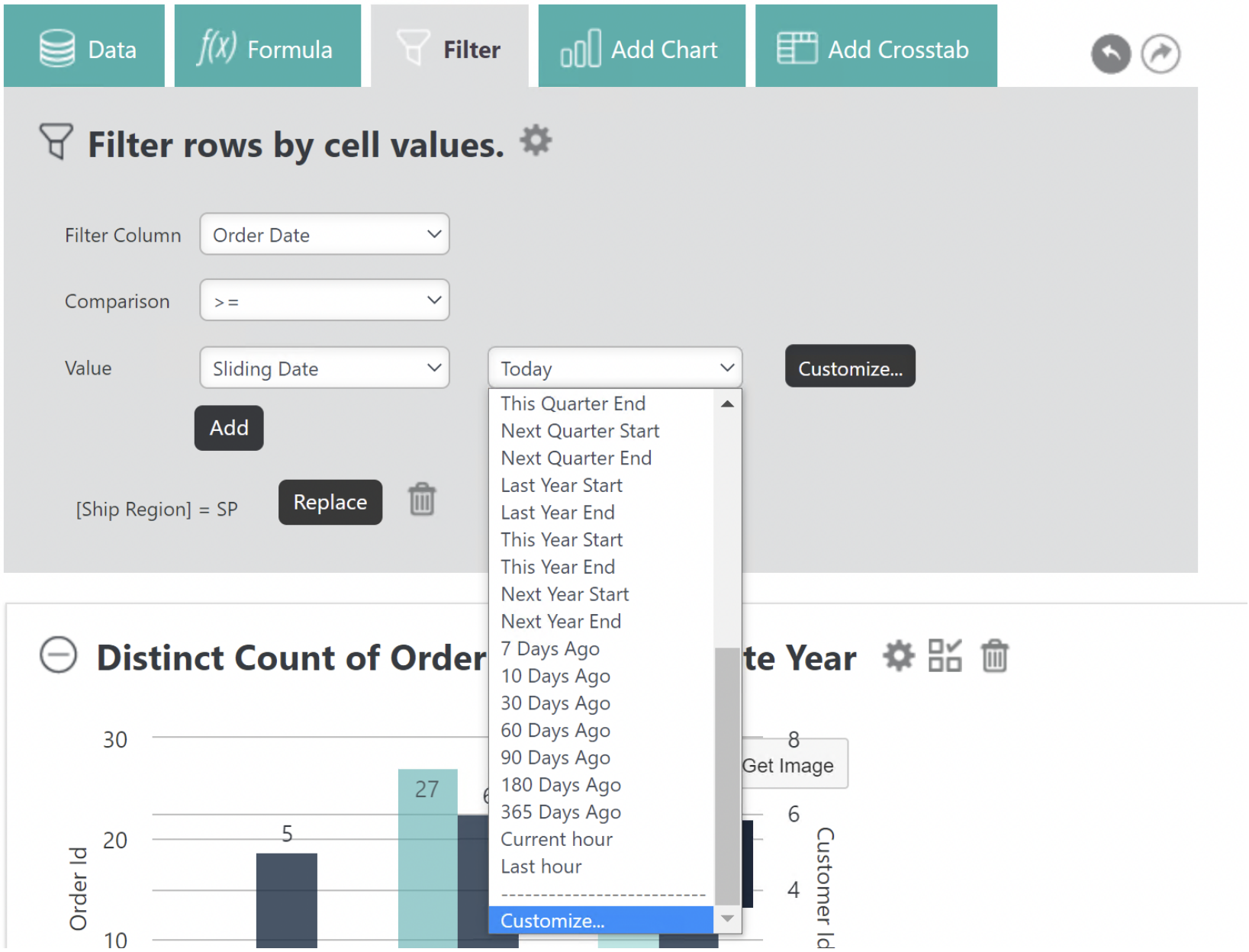
Date- and DateTime-type columns can be configured, using the **Popup Values for Filter** attribute, to offer users different filter value selection methods, in addition to direct entry: selection from a list, from a calendar, and from a time picker. Different input controls are displayed for each method:



For illustration purposes, in the example above, we've created separate filters for each of these methods - a real implementation would probably not include this. The Order Date column can only be represented by one Analysis Filter Column element so to produce the variety of selection methods in the example, we added columns to the datalayer using Calculated Column elements, configured using @Data.OrderDate~ tokens, and then added Analysis Filter Columns for these calculated columns.

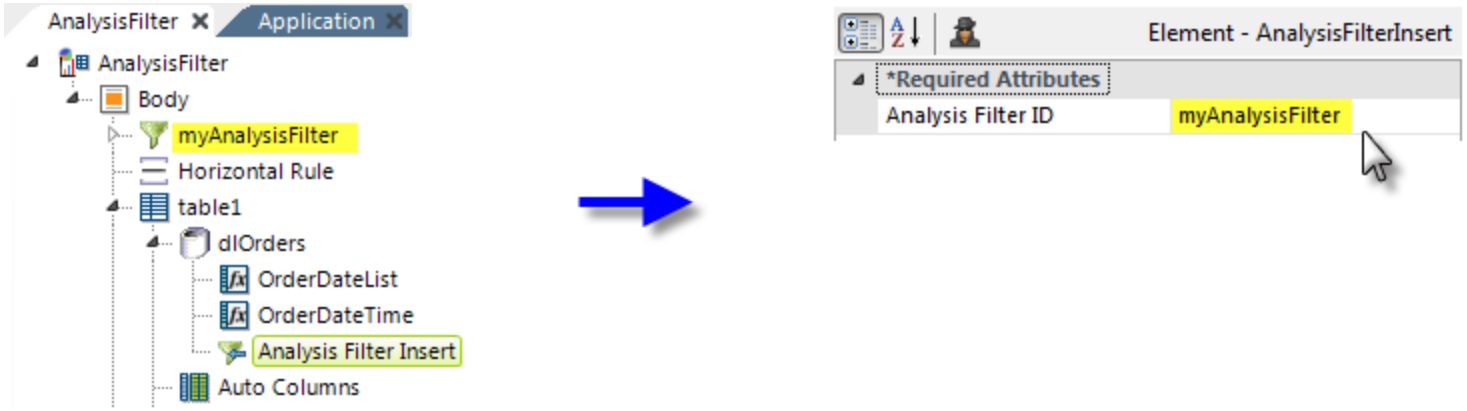


Globalization element's First Day of Fiscal Year attribute must be set in _Settings for these to appear. (v12.5+)


Data
Formula
Filter
Add Chart
Add Crosstab
Filter rows by cell values.
Filter Column
Order Date
Comparison
>=
Value
Sliding Date
Today
Customize...
Add
[Ship Region] = SP
Replace
This Quarter End
Next Quarter Start
Next Quarter End
Last Year Start
Last Year End
This Year Start
This Year End
Next Year Start
Next Year End
7 Days Ago
10 Days Ago
30 Days Ago
60 Days Ago
90 Days Ago
180 Days Ago
365 Days Ago
Current hour
Last hour
Customize...
Distinct Count of Order
te Year
Get Image
30
20
10
Order Id
5
27
8
6
4
Customer Id

When working with Date- and DateTime-type data, you can select a comparison against a specific date or a "Sliding Date" time window. A drop-down list of options appears, with date options as well as the ability to customize the date and time, as shown above.

If the Globalization element's **First Day of Fiscal Year** attribute has been set in the _Settings definition, the drop-down list will include a set of Fiscal options, highlighted above.



Finally, we've added an **Analysis Filter Insert** element, as shown above, beneath our table's datalayer. Its sole attribute is configured with the ID of the Analysis Filter element. This is the element that connects the datalayer and filter.

We now have a fully-functional filtering mechanism for our Data Table.

Passing Parameters

You may need to pass parameters when the Analysis Filter updates the page or visualization. The element's **Request Forwarding** attribute allows you to pass *all* the request parameters received by the page (but doesn't include any values from

Default Request Parameters). You can also be more selective about what's passed by using a **Link Parameters** element as a child of the Analysis Filter.

More information about passing parameters can be found in *Pass Information*.

Analysis Filter - Saving Individual User Settings

The Analysis Filter will automatically save a user's runtime filter changes during their session, if the **Save File** attribute is set. The file specified will be created if it does not exist but its containing folder is not automatically created. You must ensure that the account used by the web server to run the application has "Write" permission to the folder (usually, creating the folder under the application's root folder will handle this).

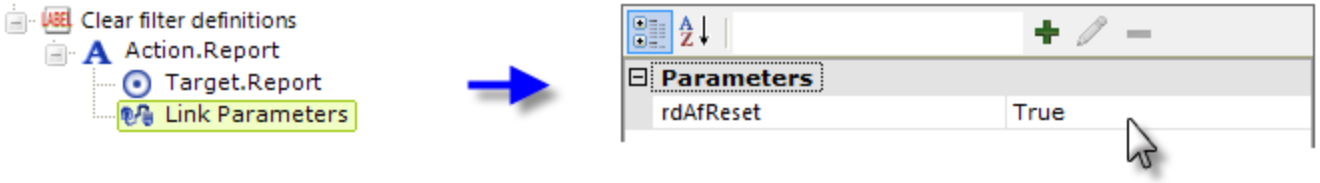
You can also save separate configurations for each user that survive their sessions. If Logi Security is being used, include the @Function.UserName~ token as part of the Save File name. If not, you can use @Function.GUID~ to generate a unique file name for the user and then save the file name as a cookie in the user's browser.

Analysis Filter - Filtering with Query String Parameters

Special reserved query string parameters can be used to manipulate the Analysis Filter.

Remove all Filter Definitions

All filter definitions can be removed by calling the report with the **rdAfReset** parameter set to *True*.



In the example above, a Label has been added to a report as a link that clears all filters. The Target.Report element is set to *Current Report* and the Link Parameters element is used to pass the rdAfReset parameter.

Force a Page Refresh with Current Filter Values

You can force a complete page refresh by calling the report with the **rdAfRefresh** parameter set to *True*.

Define a New Filter

You can define and apply a new filter by sending a full set of query string parameters. The parameters are:

rdAfCommand="FilterSet"
rdAfFilterColumn_<*AnalysisFilterID*>="<*ColumnName*>"

rdAfFilterOperator_<*AnalysisFilterID*>="<*Operator*>"
rdAfFilterValue_<*AnalysisFilterID*>="<*FilterValue*>"
rdAfMode_<*AnalysisFilterID*>="<*ViewMode*>"
rdAnalysisFilterID="<*AnalysisFilterID*>"

where:

- <*AnalysisFilterID*> is the ID of an existing Analysis Filter element in the report being called
- <*ColumnName*> is the name of a column that has an Analysis Filter Column element configured for it
- <*Operator*> is one of the operators available in Design view
- <*FilterValue*> is the data value on which to filter the column
- <*ViewMode*> specifies the filter display mode to be used, *Simple* or *Design*

If one or more filters already exist for the specified column then when these parameters are applied, the *value* of the first existing filter will change. An additional filter will *not* be created.

Here's an example for comparison purposes, using the XML source for these parameters defined in a Link Parameters element. They're based on the filter examples used in "Analysis Filter - Using the Analysis Filter Elements" on page 184:

```
<LinkParams
 rdAfCommand="FilterSet"
 rdAfFilterColumnID_<AnalysisFilterID>="CustomerID"
 rdAfFilterOperator_myAnalysisFilter="="
 rdAfFilterValue_myAnalysisFilter="VINET"
 rdAfMode_myAnalysisFilter="Simple"
 rdAnalysisFilterID="myAnalysisFilter"
/>
```

To make the code dynamic, tokens can be used to provide values for the parameters:

```
rdAfFilterValue_myAnalysisFilter="@Request.FilterOnThis~"
```

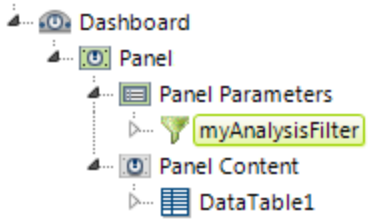
Analysis Filter - Using Analysis Filters in Dashboards

Analysis Filters can be used to filter data for visualizations in Dashboards, in individual panels and globally.

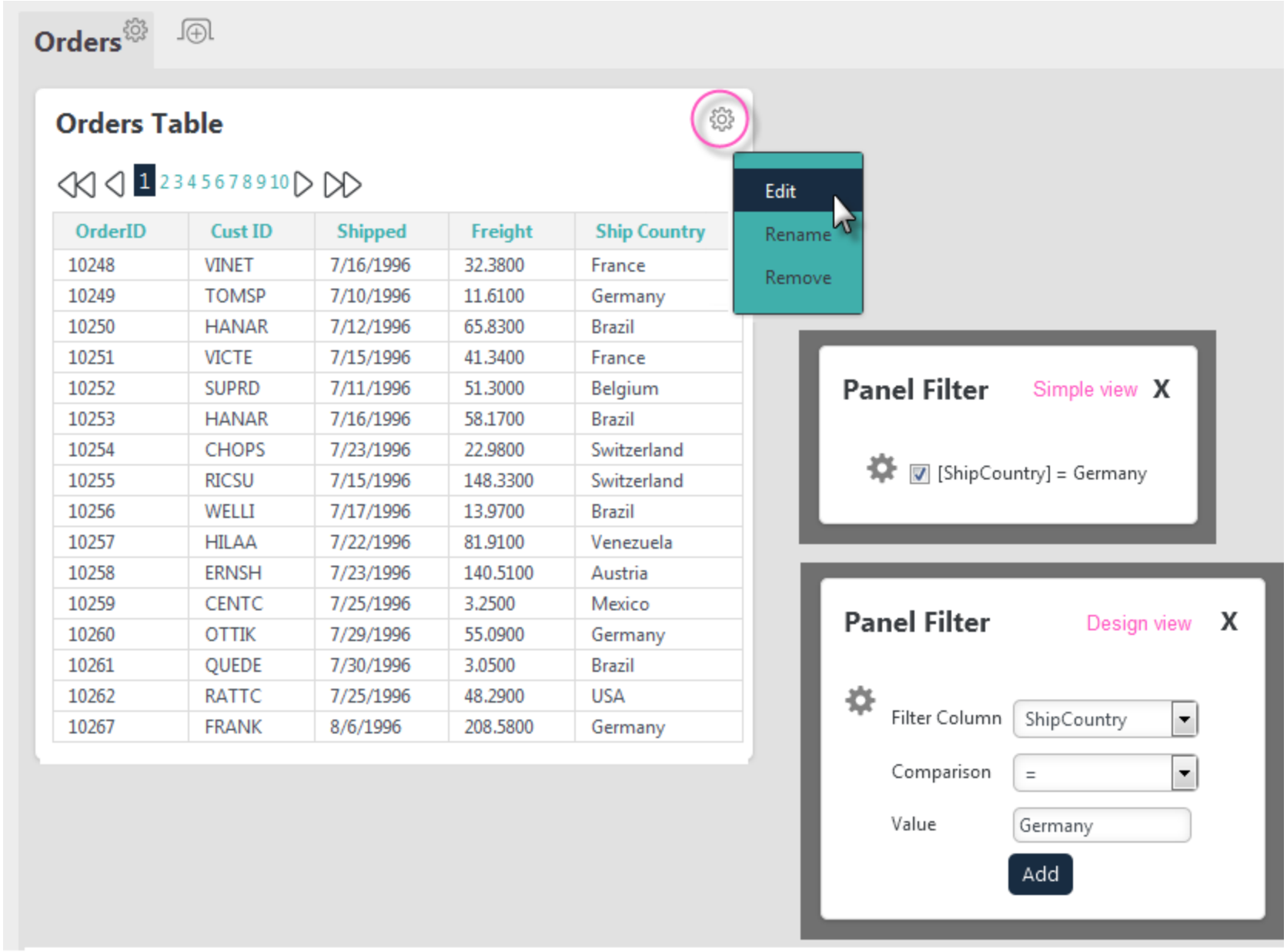
Panels derived from an Analysis Grid having an Active Query Builder child element can have the Analysis Filter *automatically generated*. This enables the user to filter data for panels created from a metadata-driven Analysis Grid, without the need to add the Analysis Filter elements described below. This is controlled by the Dashboard element's **Auto Panel Filters** attribute.

Filtering Individual Panels

As you might expect, the data used in a Dashboard panel can be filtered with an Analysis Filter.



As shown above, implementing Analysis Filters in a panel is simple. In the Dashboard panel, add a **Panel Parameters** element with a child Analysis Filter element. Configure the Analysis Filter, and its child elements. Don't forget to add an **Analysis Filter Insert** element beneath your datalayer.

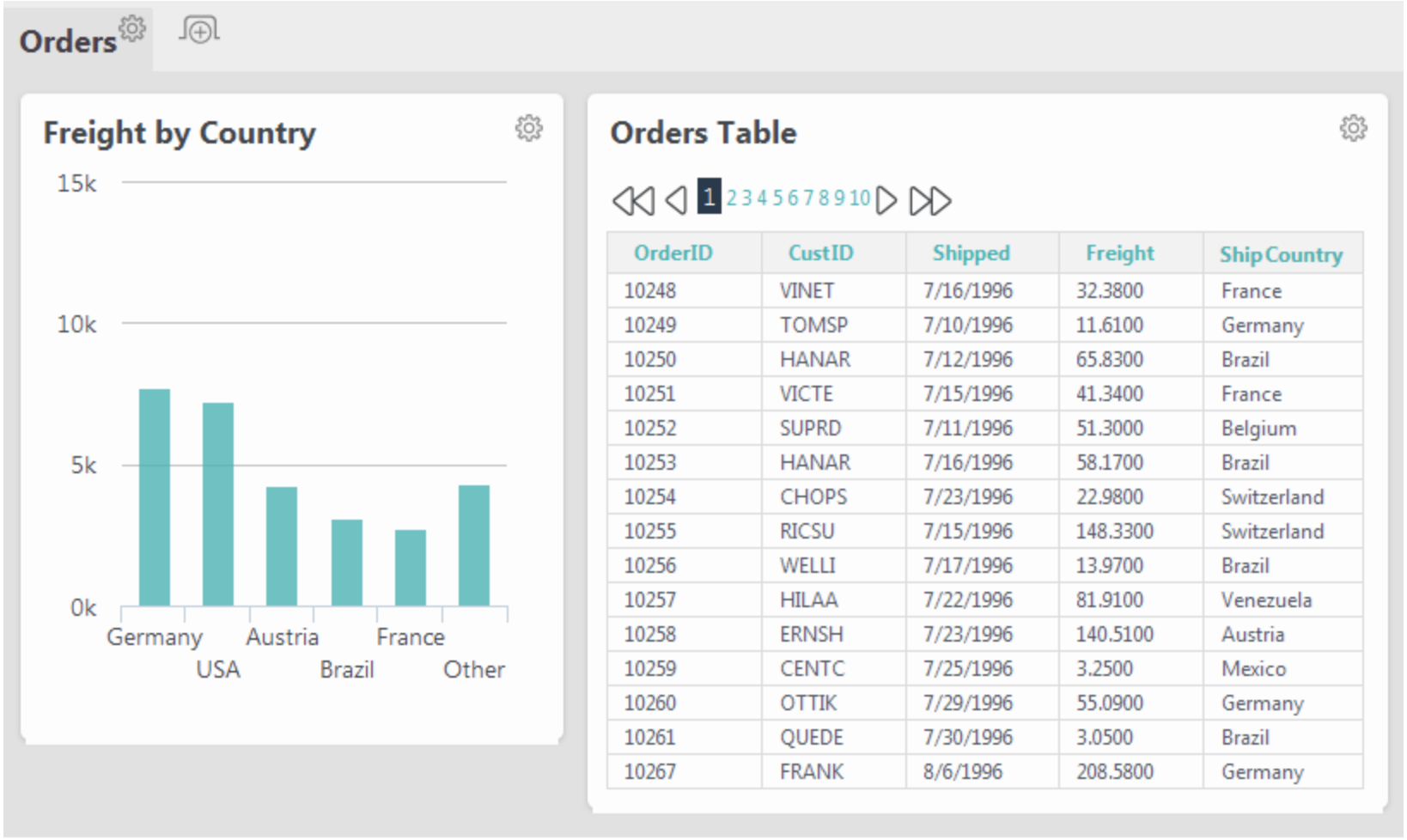

Orders
Orders Table
1 2 3 4 5 6 7 8 9 10
OrderID | Cust ID | Shipped | Freight | Ship Country
10248 | VINET | 7/16/1996 | 32.3800 | France
10249 | TOMSP | 7/10/1996 | 11.6100 | Germany
10250 | HANAR | 7/12/1996 | 65.8300 | Brazil
10251 | VICTE | 7/15/1996 | 41.3400 | France
10252 | SUPRD | 7/11/1996 | 51.3000 | Belgium
10253 | HANAR | 7/16/1996 | 58.1700 | Brazil
10254 | CHOPS | 7/23/1996 | 22.9800 | Switzerland
10255 | RICSU | 7/15/1996 | 148.3300 | Switzerland
10256 | WELLI | 7/17/1996 | 13.9700 | Brazil
10257 | HILAA | 7/22/1996 | 81.9100 | Venezuela
10258 | ERNSH | 7/23/1996 | 140.5100 | Austria
10259 | CENTC | 7/25/1996 | 3.2500 | Mexico
10260 | OTTIK | 7/29/1996 | 55.0900 | Germany
10261 | QUEDE | 7/30/1996 | 3.0500 | Brazil
10262 | RATTC | 7/25/1996 | 48.2900 | USA
10267 | FRANK | 8/6/1996 | 208.5800 | Germany
Edit
Rename
Remove
Panel Filter
Simple view
X
[ShipCountry] = Germany
Panel Filter
Design view
X
Filter Column
ShipCountry
Comparison
=
Value
Germany
Add

Once the Analysis Filter has been added, clicking the panel's "gear" icon will display a menu that has an "Edit" option. This option will open the Panel Filter panel, as shown above. The controls displayed in it will depend on the view mode, Simple or Design, configured in the Analysis Filter attributes and have been discussed earlier in this topic.

Adding or removing filters will immediately filter the data in the Dashboard panel accordingly.

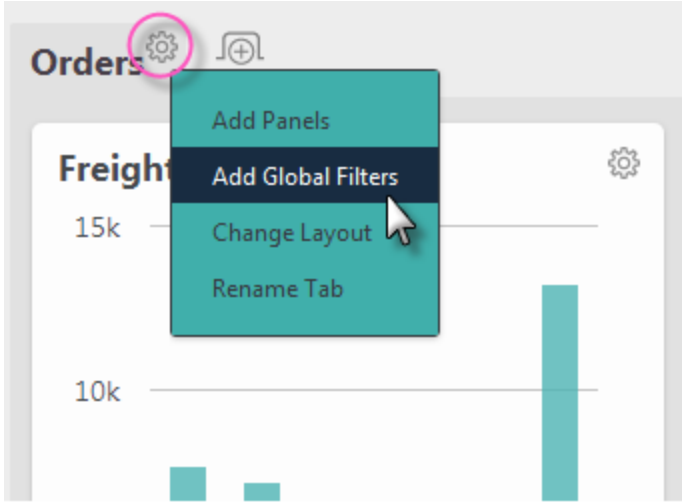
Adding a Global Filter

You can also add a "global" filter, one that affects the data used in *multiple* Dashboard panels at once. This type of filter functionality is automatically available when there are multiple Dashboard panels containing Analysis Filter elements, or multiple panels created from a metadata-driven Analysis Grid, and is controlled by the Dashboard element's **Auto Global Filter** attribute.

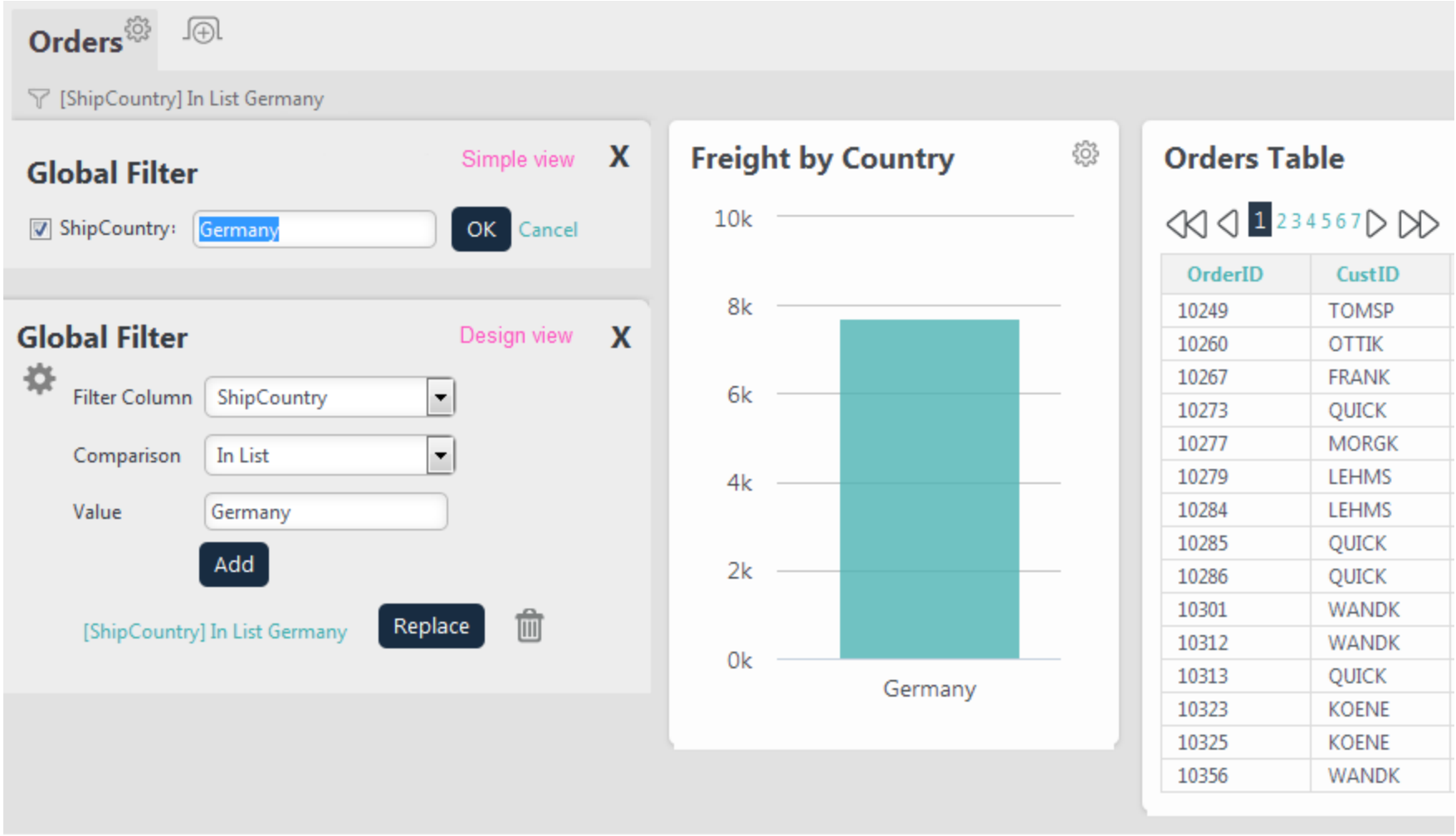


OrderID	CustID	Shipped	Freight	ShipCountry
10248	VINET	7/16/1996	32.3800	France
10249	TOMSP	7/10/1996	11.6100	Germany
10250	HANAR	7/12/1996	65.8300	Brazil
10251	VICTE	7/15/1996	41.3400	France
10252	SUPRD	7/11/1996	51.3000	Belgium
10253	HANAR	7/16/1996	58.1700	Brazil
10254	CHOPS	7/23/1996	22.9800	Switzerland
10255	RICSU	7/15/1996	148.3300	Switzerland
10256	WELLI	7/17/1996	13.9700	Brazil
10257	HILAA	7/22/1996	81.9100	Venezuela
10258	ERNSH	7/23/1996	140.5100	Austria
10259	CENTC	7/25/1996	3.2500	Mexico
10260	OTTIK	7/29/1996	55.0900	Germany
10261	QUEDE	7/30/1996	3.0500	Brazil
10267	FRANK	8/6/1996	208.5800	Germany

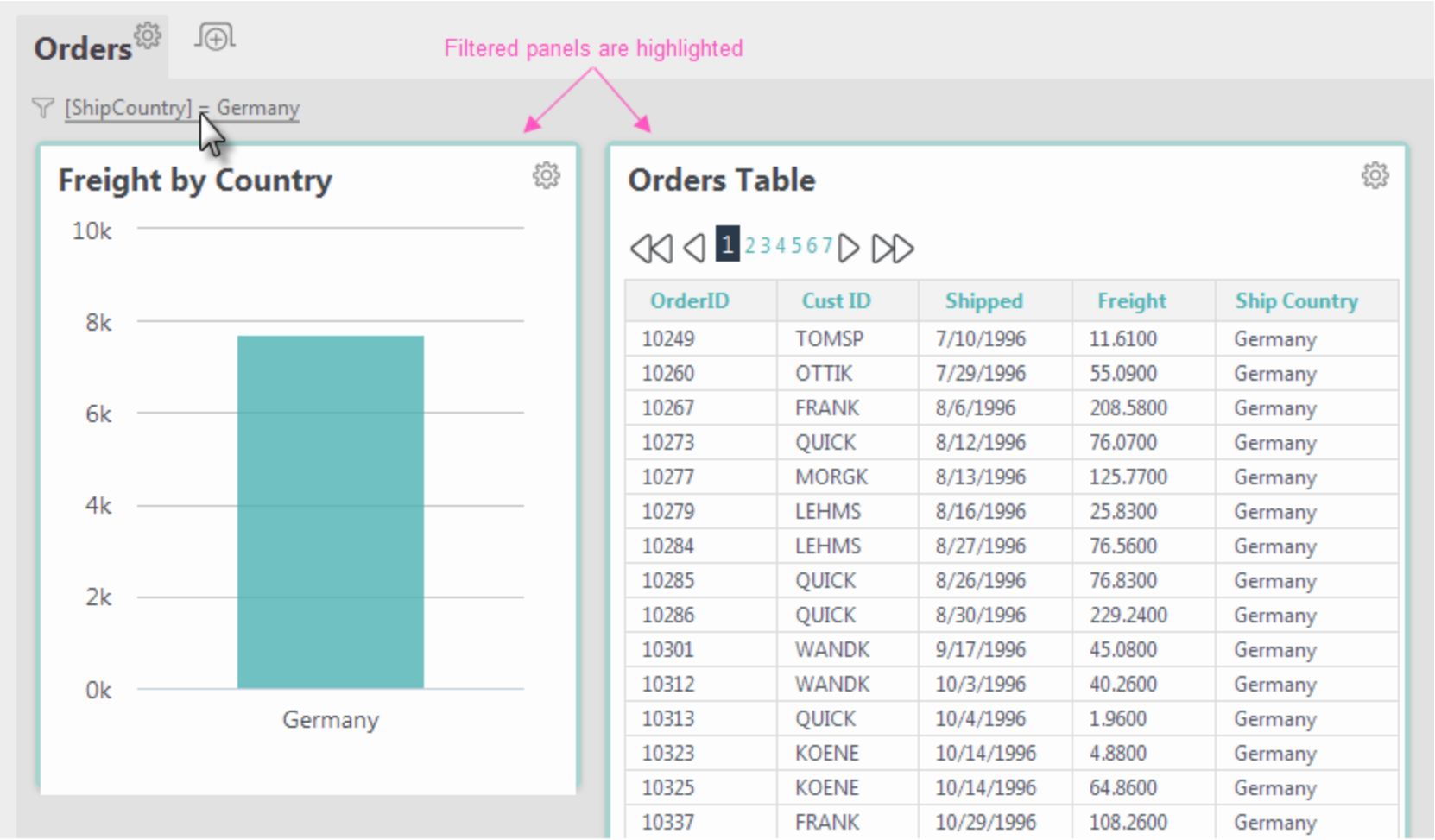
The example Dashboard shown above has two panels, each of which has an Analysis Filter configured for it. Both visualizations use data from the ShipCountry column in their datalayers.



We can create a global filter by clicking the gear icon for the Dashboard tab, as shown above, and selecting the *Add Global Filters* pop-up menu item.



The Global Filter panel will be displayed, as shown above. The controls displayed in it will depend on the configured view mode and have been discussed earlier in this topic. Adding a global filter will immediately update the data in both Dashboard panels.



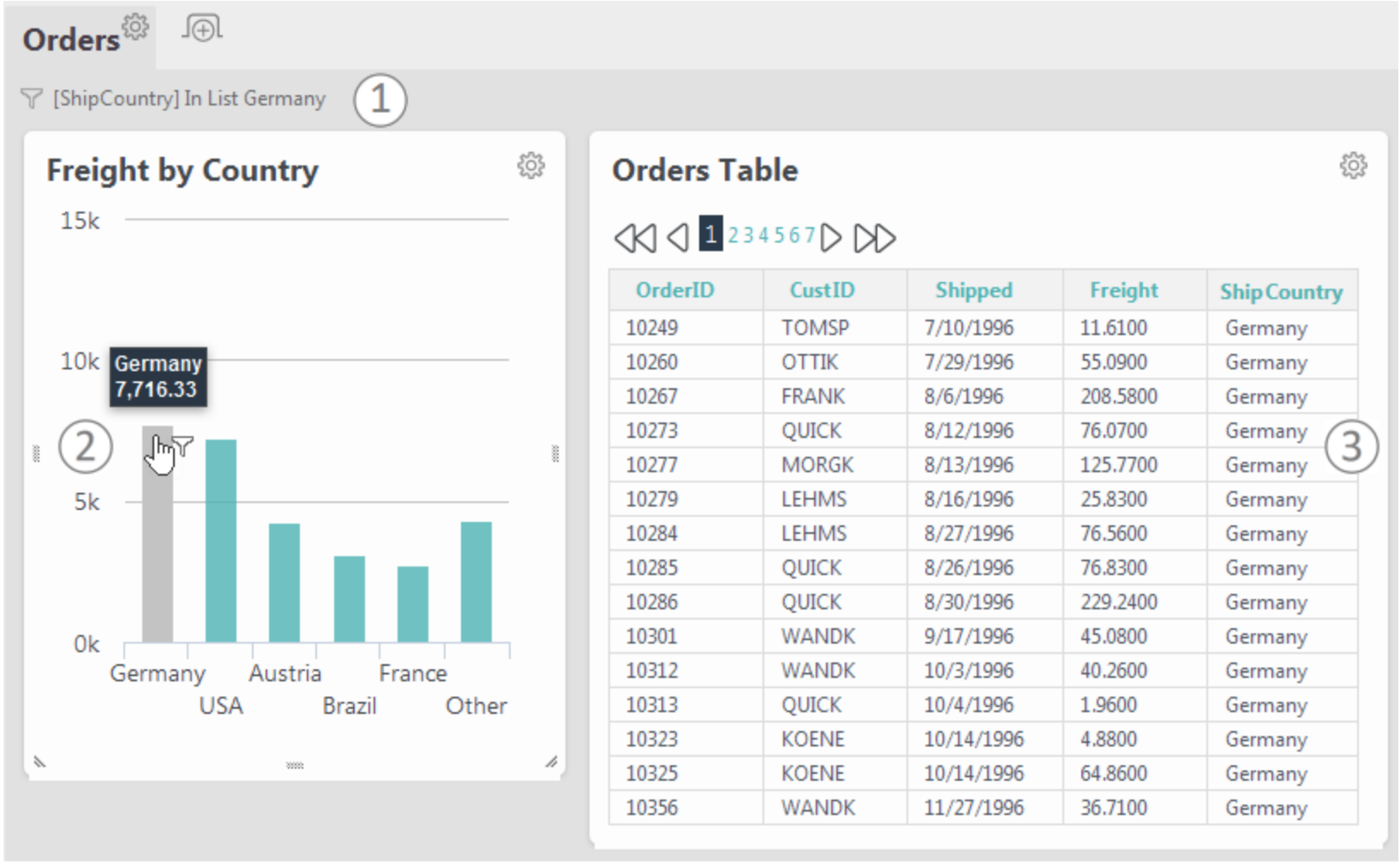
A global filter description will appear at the top of the tab and can be used as a link to re-open the Global Filter panel.

In addition, when you hover your mouse cursor over this description, the panels affected by the global filter will be highlighted, as shown above.

If you uncheck *all* of the defined filters in Simple view and close the panel, no descriptions are shown but the Filter icon will remain and you can click it to re-open the panel.

Global Filtering using Chart Data Points

You can also create a global filter in a Dashboard by selecting data points in a chart. This is behavior similar to that seen when you use Chart Canvas chart and Input Selection elements together. Let's see it in action:

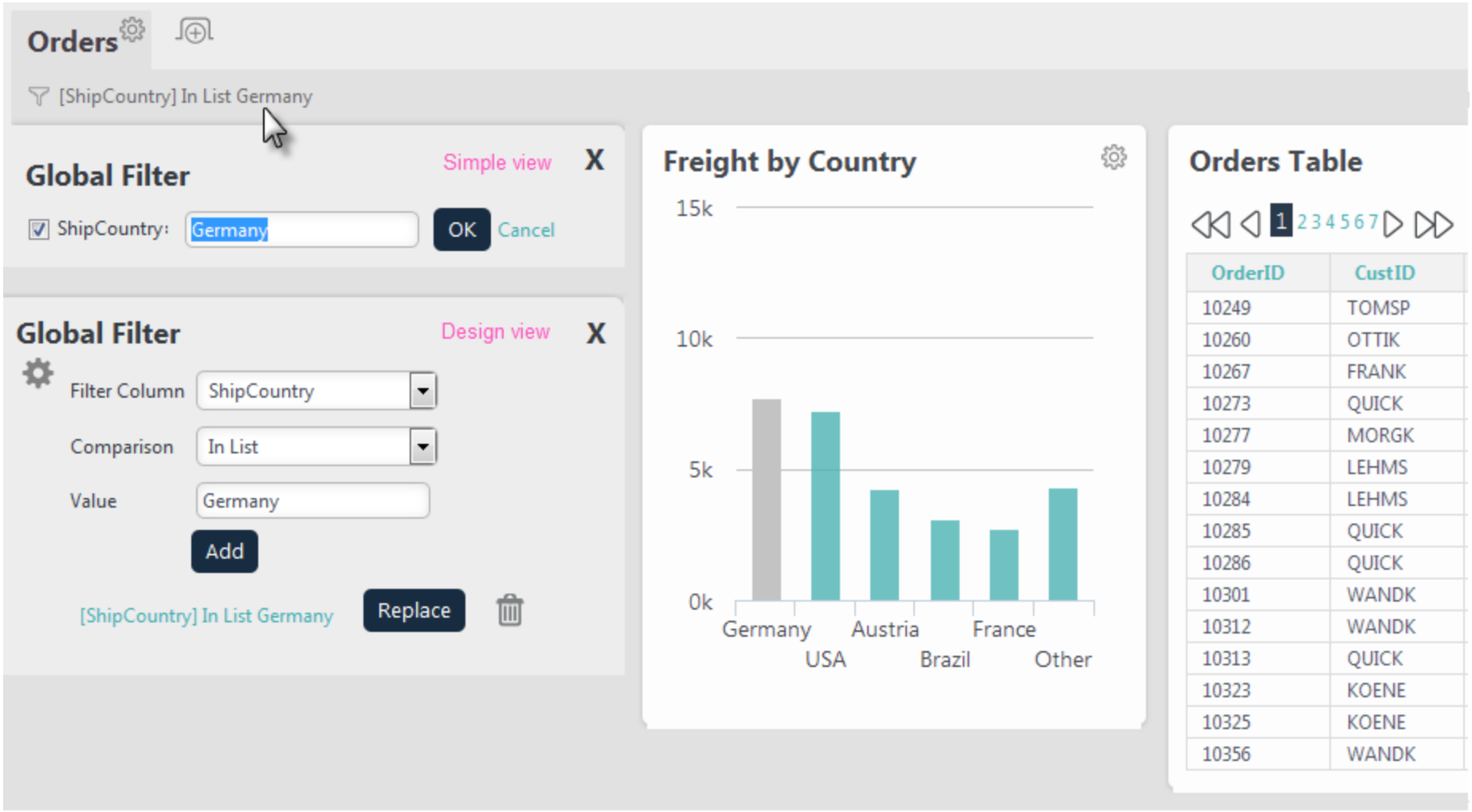


Once again, we have an example with two Dashboard panels, one with a chart and one with a table, shown above. When a bar in the chart is clicked, a global filter is created automatically and three other things happen:

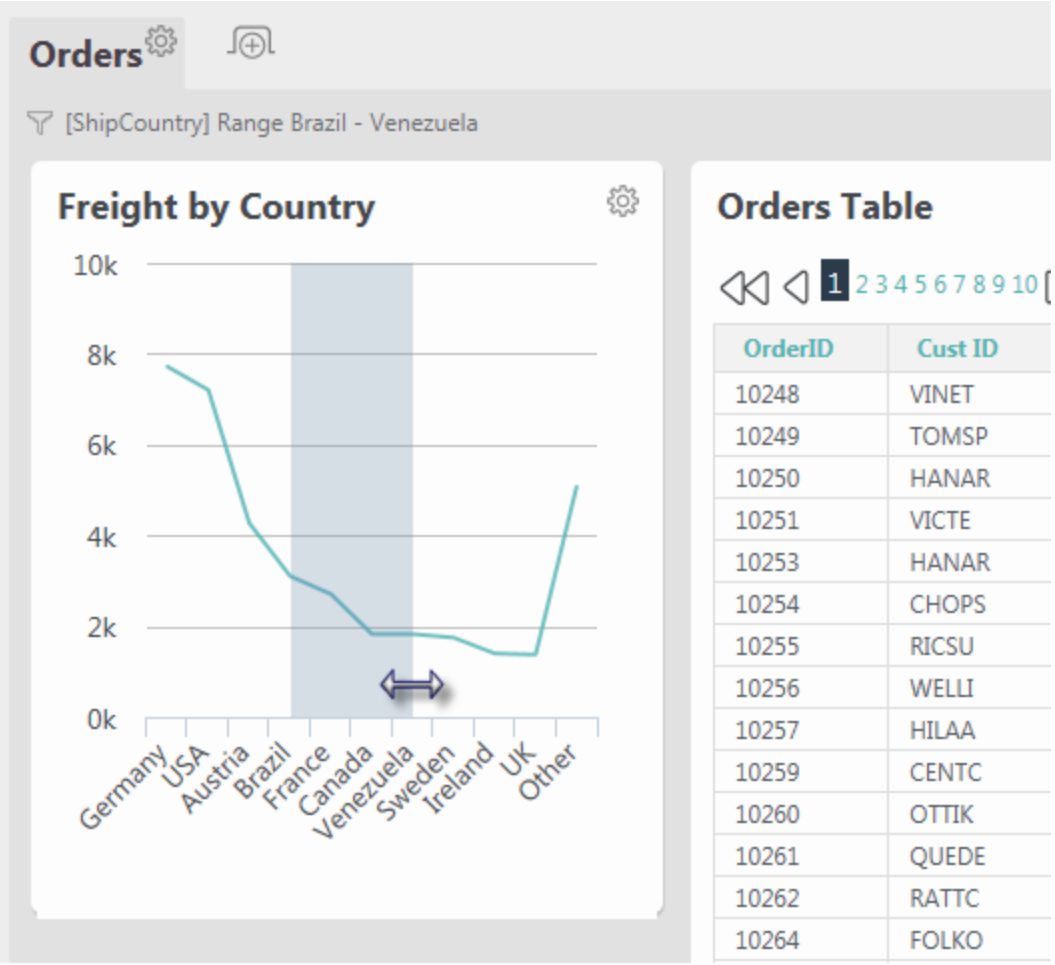
1. A Filter Description appears at the top of the Dashboard tab. This can be clicked to edit the global filter.
2. The color of the selected chart bar changes to indicate it's included in a filter.
3. The data for the table in the second panel is filtered.

This is different from the previous global filtering example because the chart that was clicked is *not* filtered.

Clicking the same chart bar a second time will cancel the filter related to that data. Clicking another bar will add additional criteria to the global filter.



Clicking the Filter Description at the top of the Dashboard tab will cause the automatically-generated Global Filter panel to be displayed. As before, the controls displayed in it will depend on the view mode, Simple or Design, configured in the Analysis Filter attributes.



For chart types that support it, you can also create a global filter by dragging the cursor to select a range of values, as shown in the Line chart above. This is similar to using Chart Canvas chart and Input Selection Range elements.

More information about Dashboards can be found in *Dashboard for End Users*.

Analysis Filter - Customizing the Analysis Filter Appearance

Analysis Filter appearance can be changed most easily by applying a theme to the report definition. Most of the screen shots in this document were taken with the *Signal* theme applied. You can also create your own custom theme, based on a standard theme, using the Theme Editor tool. For more information, see *Working with Themes*.

Changing Appearance Using Style Classes

Analysis Filter appearance can be also customized using **style classes** and the standard classes for it are taken from the Analysis Grid style sheet:

```
<yourAppFolder>\rdTemplate\rdAnalysisGrid\rdAg10Style.css
```

Developers can override classes in this style sheet by *copying* them to their application style sheet and modifying them there.

 Do not make changes in the standard style sheet in the rdTemplate folder.

Changing Appearance Using Template Modifiers

The Analysis Filter element uses a "template file" to define certain element properties that are not otherwise available as attributes to the developer for modification. These include language- and culture-specific **Caption** attributes that you may want to change to match the locale (or you may simply want to change the captions to better suit your application).

The Analysis Filter element's **Template Modifier File** attribute identifies a custom XML file developers can create containing elements that will override the same elements in the template file.

For example, the Analysis Filter template file:

`<yourAppFolder>\rdTemplate\rdAnalysisFilter\rdAfTemplate.lgx`

contains several Label elements. One of them has an ID of "lblFilterColumn_rdAfID"; this controls the caption text that appears beside the Filter Column drop-down list in Design view. This text can be modified by changing the Caption associated with that Label element. To change the text from its default "Filter Column" to "Column to filter by", create your own XML file, identify it in the Template Modifier File attribute, and add this code to it:

```
<TemplateModifier>
<SetAttribute ID="lblFilterColumn_rdAfID" Caption="Column to filter by" />
</TemplateModifier>
```

You can set the attributes for any number of elements in this file; examine the rdAfTemplate.lgx file to learn the ID and Caption attributes available. The template modifier file can be in any folder accessible to the web application; if a fully-qualified file path is not provided in the Template Modifier File attribute value, then the application expects the file to be in its _SupportFiles folder.

More detailed information about template modifier files can be found in *Template Modifier Files*.

The Calculated Column

The **Calculated Column** element adds a new column to a datalayer containing a value that's the result of a formula. The formula can include data from other columns, tokens, and literals, and is evaluated for every row in the datalayer.

The following topics discuss the Calculated Column element:

- Calculated Column Attributes
- Calculated Column Usage Examples
- NaN Errors

About the Calculated Column

The Calculated Column is similar to other elements designed to *extend* the data available in a datalayer. Like the **Aggregate Column** element, the Calculated Column is added as a child element beneath a datalayer and adds a new column to the datalayer. This column value is *derived data* which is accessible using an @Data token.

The new column's value results from evaluating a formula, for each row in the datalayer. This calculated value is populated in the new column in every row of the datalayer and can be used in further calculations, Summary Rows, Link Params, etc.

Similar calculations could be performed within SQL queries and *may* offer better performance when done by the database server. However, the Calculated Column element allows calculations to be performed on data from other sources too, such as web services and data files, that do not themselves offer SQL-like queries.

Calculated Column Attributes

The attributes for the Calculated Column element are described below:

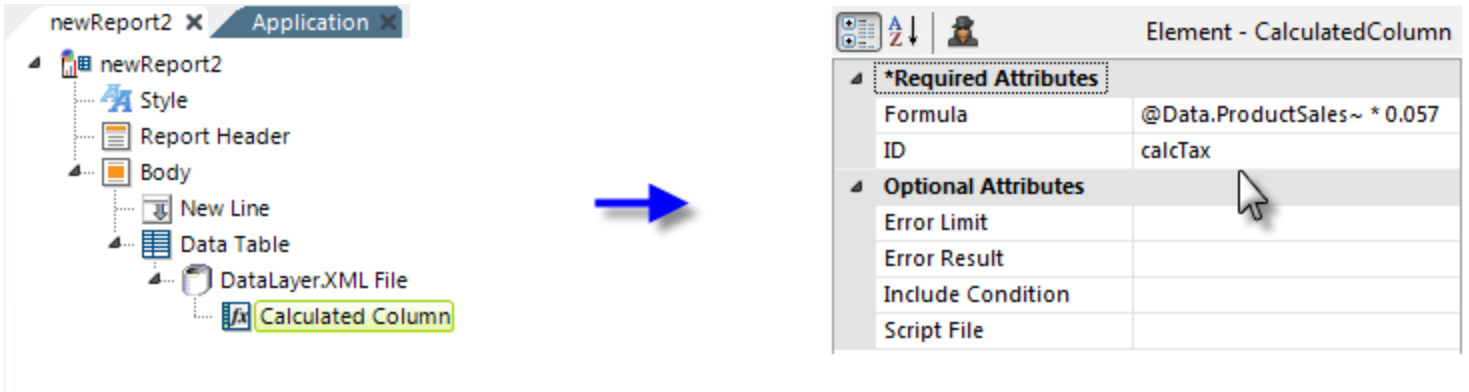
Attribute	Description
Formula	(Required) Specifies a JavaScript expression that returns a string, numeric, or boolean value. Tokens, including data from other datalayer columns, literals, intrinsic functions, and functions in script files may be used here. *Do not* start the expression with an equals sign (=). Usage examples are provided below. Select the scripting type using the General element's Scripting Language attribute, in the _Settings definition (JavaScript if the default and recommended language; VBScript is available for legacy application compatibility only).
ID	(Required) Specifies a unique ID for the Calculated Column element, and becomes the name of the column added to the datalayer.
Error Limit	Specifies the maximum number of times (datalayer rows) evaluating an expression will be allowed to produce an error before the value specified in the Error Result attribute will be used instead.
Error Result	Specifies the value to be returned when the Error Limit is reached during evaluation. Default: *blank*
Include Condition	Specifies whether or not this element will be processed when the report runs. If left blank or if it contains an expression that evaluates to *True*, the element is processed and the new column is added to the datalayer. If it evaluates to *False*, the element is ignored and does not affect the datalayer.
Script File	Specifies the name of an optional script file containing functions or routines called in the Formula expression. If the script file is in the _SupportFiles folder, no path is necessary. Otherwise, specify a fully-qualified file system

Attribute	Description
	path and file name (tokens such as @Function.AppPhysicalPath~ may be used).

Calculated Column Usage Examples

Here are some examples of the Calculated Column element in use.

1. The following shows a simple implementation - calculating sales tax:



In the definition shown above, a Data Table and a datalayer element have been added. A **Calculated Column** element has been added beneath the datalayer and its attributes have been configured so that it will multiply the values in the ProductSales column by a constant to produce the sales tax amount.

CategoryID	ProductName	ProductSales
1	Chai	$4,887.00
2	Aniseed Syrup	$1,724.00
3	Chocolade	$1,282.01
4	Camembert Pierrot	$20,505.40
5	Filo Mix	$2,124.15
6	Alice Mutton	$17,604.60
7	Longlife Tofu	$1,000.50
8	Boston Crab Meat	$9,814.73

Original Data

CategoryID	ProductName	ProductSales	calcTax
1	Chai	$4,887.00	$278.56
2	Aniseed Syrup	$1,724.00	$98.27
3	Chocolade	$1,282.01	$73.07
4	Camembert Pierrot	$20,505.40	$1,168.81
5	Filo Mix	$2,124.15	$121.08
6	Alice Mutton	$17,604.60	$1,003.46
7	Longlife Tofu	$1,000.50	$57.03
8	Boston Crab Meat	$9,814.73	$559.44

After using Calculated Column to add calcTax column

The resulting sales tax amount will be placed in a new column, named calcTax, in every row of the datalayer, as shown above. The data will be accessible as the token @Data.calcTax~. Instead of hard-coding the sales tax rate in the formula, it could be made dynamic using a token, like this:

```
@Data.ProductSales~ * @Request.TaxRate~
```

Here's a slightly different example: the following expression uses the intrinsic IIF function to apply different tax rates, depending on other criteria:

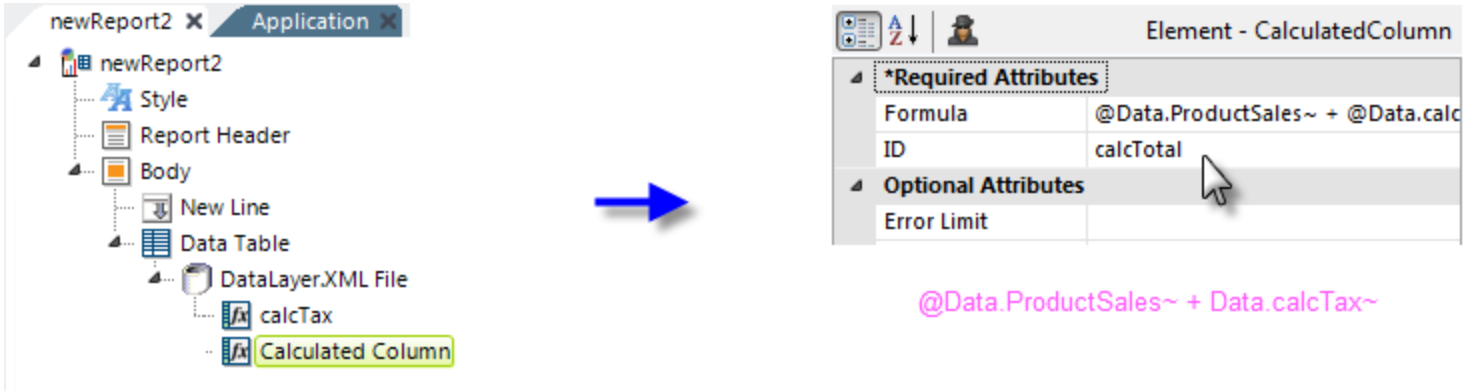
```
IIF(@Data.CategoryID~ = 4, @Data.ProductSales~ * @Request.TaxRate1~, @Data.ProductSales~ *
@Request.TaxRate2~)
```

Of course, your expression doesn't *have* to use data from the datalayer at all. The following example simply creates a new column and puts the elapsed seconds for every row for it:

```
DateDiff("s", CXMLDate("@Request.StartTime~"), CXMLDate("@Function.Time~"))
```

Note the use of the intrinsic CXMLDate function, which is necessary to convert ISO formatted XML DateTime data into a format that works with the DateDiff function (and should be used with DateAdd, DatePart, and other date functions). The possibilities are endless and allow developers a great deal of flexibility in manipulating the datalayer.

2. This example illustrates how the value created by one Calculated Column can be used in another Calculated Column.



In the definition shown above, a second **Calculated Column** has been added beneath the datalayer. As you can see, its Formula attribute uses data from one of the retrieved data columns *and* the first calculated column.

CategoryID	ProductName	ProductSales
1	Chai	$4,887.00
2	Aniseed Syrup	$1,724.00
3	Chocolade	$1,282.01
4	Camembert Pierrot	$20,505.40
5	Filo Mix	$2,124.15
6	Alice Mutton	$17,604.60
7	Longlife Tofu	$1,000.50
8	Boston Crab Meat	$9,814.73

Original Data

CategoryID	ProductName	ProductSales	calcTax	calcTotal
1	Chai	$4,887.00	$278.56	$5,165.56
2	Aniseed Syrup	$1,724.00	$98.27	$1,822.27
3	Chocolade	$1,282.01	$73.07	$1,355.08
4	Camembert Pierrot	$20,505.40	$1,168.81	$21,674.21
5	Filo Mix	$2,124.15	$121.08	$2,245.23
6	Alice Mutton	$17,604.60	$1,003.46	$18,608.06
7	Longlife Tofu	$1,000.50	$57.03	$1,057.53
8	Boston Crab Meat	$9,814.73	$559.44	$10,374.17

after using a second Calculated Column
to add the ProductSales and calcTax columns

The resulting total amount will be placed in a new column, named calcTotal, in every row of the datalayer, as shown above. The data will be accessible as the token @Data.calcTotal~.

NaN Errors

If data or other circumstances cause the Calculated Column's expression to attempt a "division-by-zero", the resulting value in the datalayer will be `NaN`, which stands for "Not a Number". This can cause problems or be confusing when the data is visualized.

If this kind of result is possible, then you may want to wrap your expression in an IIF function in order to provide an alternative to the `NaN` result in the data. For example, the expression

```
IIF(@Data.RoomsAvailable~ <> 0, @Data.RoomRevenue~/@Data.RoomsAvailable~, "")
```

tests to see if a division by zero is going to happen and, if so, instead of doing the calculation, supplies an empty value in the datalayer.

The Color Range Column

The **Color Range Column** element, introduced in v12.6, adds new columns, containing color values that are correlated to data, to a datalayer. This allows you to dynamically set the background and text colors in individual Data Table cells.

The following topics introduce the Color Range Column element:

- Color Range Column Attributes
- Color Range Column Usage Examples: Numeric Data
- Color Range Column Usage Examples: Text Data

About the Color Range Column

The **Color Range Column** is similar to other elements designed to *extend* the data available in a datalayer. It's added as a child element beneath a datalayer and adds a two new columns to the datalayer. These column values are derived data, accessible using @Data tokens.

The new columns' values result from testing other column data values against specified ranges, for each row in the datalayer, and correlating them to specified color values. These color values populate the new columns in every row of the datalayer and, via their @Data tokens, can be used to dynamically set background and text color values for each Data Table cell.

When definitions using Color Range Columns are exported to Excel or to PDF, the Data Table cell text and background colors are exported as well.

Each Color Range Column element evaluates one other data column; multiple Color Range Column elements can be used, if needed.

Color Range Column Attributes

The attributes for the Color Range Column element are described below:

Attribute	Description
Data Column	(Required) Specifies the data column in the datalayer to be evaluated.
ID	(Required) Specifies a unique ID for the Color Range Column element. This becomes the name of the first column added to the datalayer, which typically contains the **background color** values.
Color Range Style	Specifies the type of data to be used when evaluating data ranges. Options include *NumericRange* and *TextList*. See the information below about Color Range element attributes to see how this selection affects its behavior. Default value: *NumericRange*
Contrast Color Column ID	Specifies the name of the second column that will be added to the datalayer, which typically provides the **text color** values. This column is *automatically* populated with color values that will contrast well with the background colors determined for the Data Table cells.
Contrast Dark Color	If you prefer to specify a text color to be used when a cell background color is "dark", rather than let the choice be automatic, you can do so here. For example, white text for a black background. Default: *automatic color selection*
Contrast Light Color	If you prefer to specify a text color to be used when a cell background color is "light", rather than let the choice be automatic, you can do so here. For example, brown text for a yellow background. Default: *automatic color selection*

Color Range Element Attributes

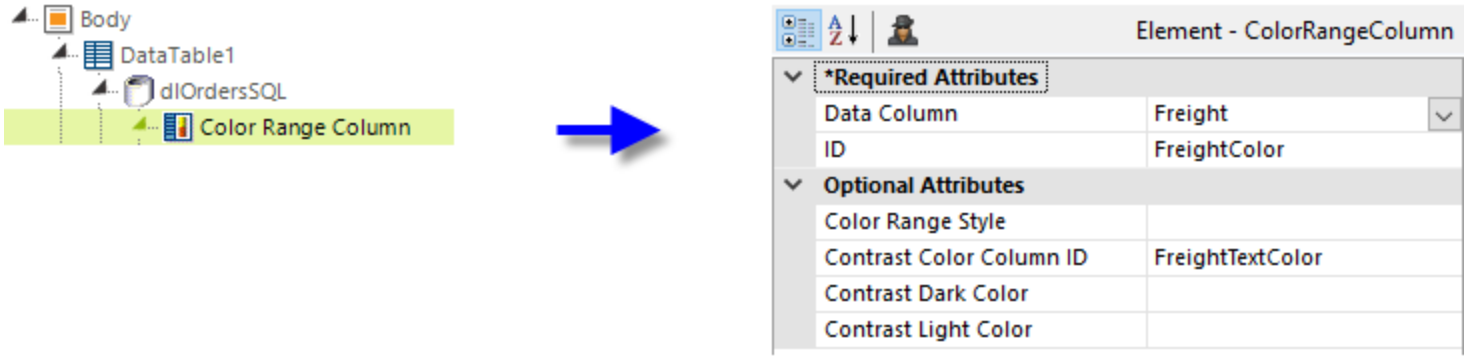
The Color Range Column element uses one or more **Color Range** child elements to specify the data ranges and colors. Multiple Color Range elements can be used to set different colors for different ranges of data.

The attributes of this child element are:

Attribute	Description
Color	Specifies the color that will be used for the cell background color. Enter a color by name, decimal RGB value, or hex RGB value. Prefix hex values with the pound sign, e.g. #112233. *Click the browse button to display theColor Picker tool.*
ID	Specifies a unique ID for the Color Range element.
Range End Value	Specifies the *ending* value of the data range that will be associated with this color. If the Color Range Column element's **Color Range Style** = *NumericRange*, then this value is assumed to be the end of a numeric range that: • Begins at a lower value (if there are *no* other Color Range elements), or • Begins at the previous Range End Value ((if there are other Color Range elements) When adding these elements with a numeric range, configure them so that their Range End values get progressively larger. If the Color Range Column element's **Color Range Style** = *TextList*, then this should be a specific text string value, such as "Profit" or "France". You may use the asterisk wildcard "*" in the string to broaden the range of text values that will be associated with this color.

Color Range Column Usage Examples: Numeric Data

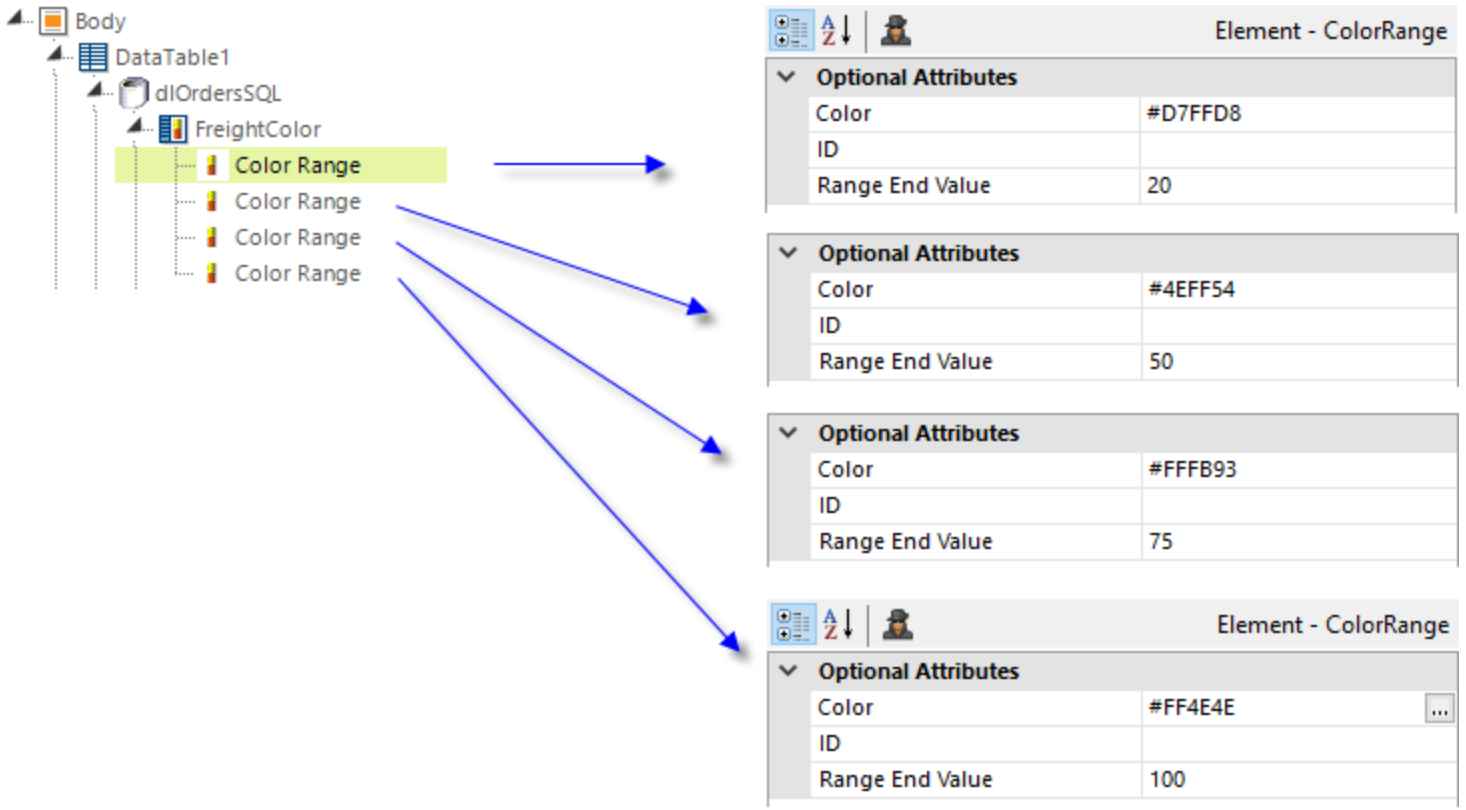
Here's an example of the Color Range Column element in use with numeric data. The technique used here adds two new columns to the data layer, evaluates data ranges and correlates them to a background color value and contrasting text color value, and inserts those color values into the new columns. We then use the @Data tokens for those columns to set the color of the Data Table cells.



In the example shown above, we start with a Data Table element with child DataLayer.SQL element and then add a Color Range Column element. Its attributes are configured as follows:

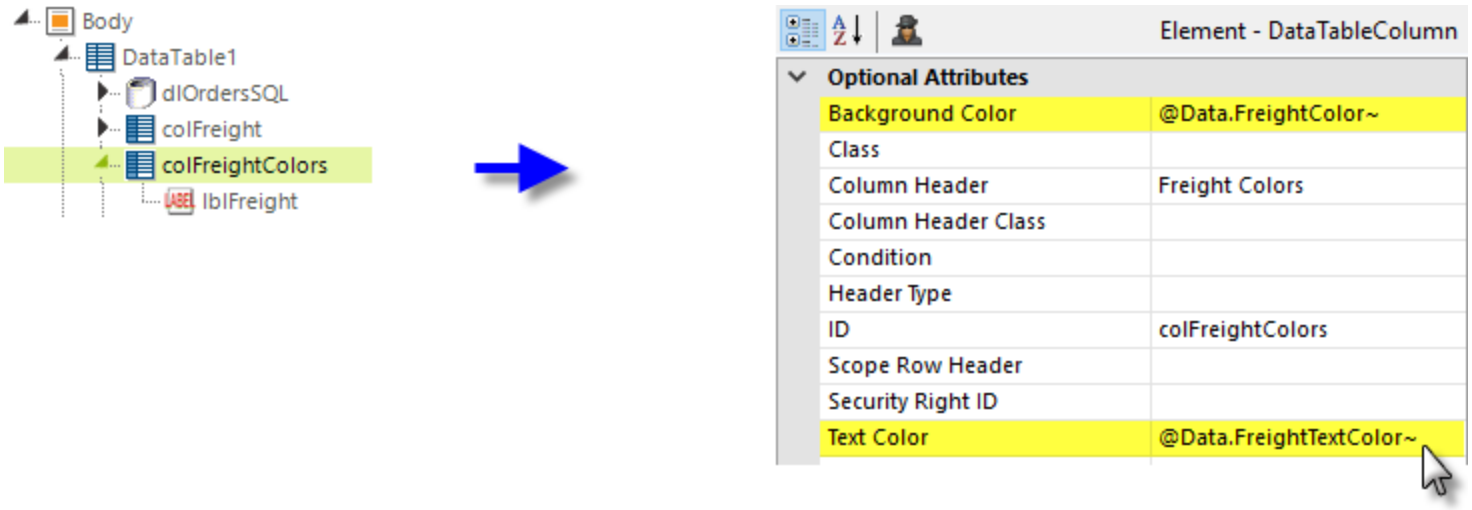
- **Data Column** - The data to be evaluated will be in the *Freight* column.
- **ID** - The new column added to the datalayer containing background color data will be named *FreightColor*.
- **Contrast Color Column ID** - The new column added to the datalayer containing contrasting text color data will be named *FreightTextColor*.

You can use multiple Color Range Column elements beneath a datalayer. Next, we add and configure four Color Range elements:



Color Range elements are evaluated according to their top-to-bottom order in the Element Tree, so the element order matters. Notice how that the Range End Value increases with each successive element. Values up to 20 will have a background color of #D7FFD8, from 21 to 50 the background will be #4EFF54, and so on up to 100. *Values above 100 will have no background color.*

If you do wish to set a background color for all values greater than 100, add a fifth Color Range element (at the "bottom" of the set of Color Range elements) and leave its Range End Value blank.



In order to display the colors, we configure our Data Table Column element, as shown above. The tokens for the data in the two new columns are used to provide values in the **Background Color** and **Text Color** attributes.

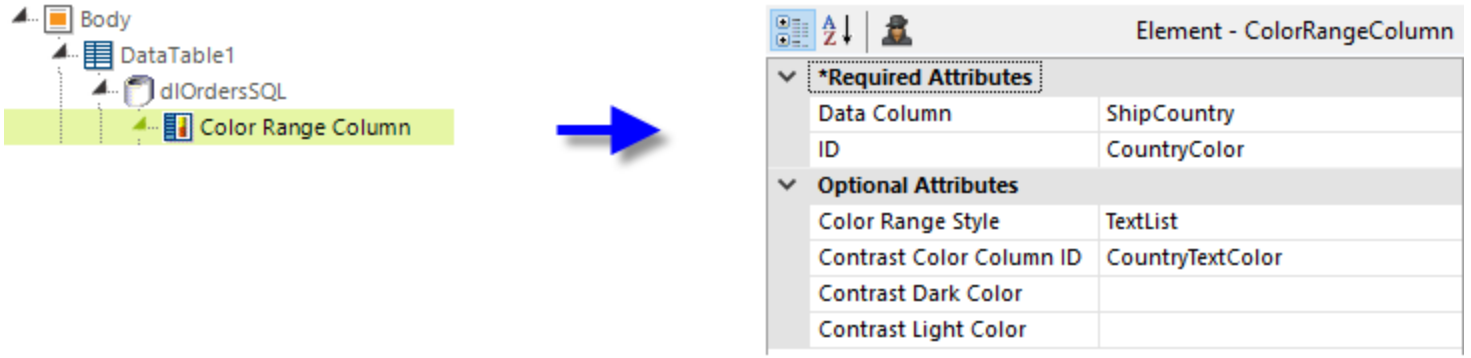
Freight	Freight Colors
32.3800	32.3800
11.6100	11.6100
65.8300	65.8300
41.3400	41.3400
51.3000	51.3000
58.1700	58.1700
22.9800	22.9800
148.3300	148.3300
13.9700	13.9700
81.9100	81.9100
140.5100	140.5100
3.2500	3.2500
55.0900	55.0900
3.0500	3.0500
48.2900	48.2900
146.0600	146.0600
3.6700	3.6700
55.2800	55.2800
25.7300	25.7300
208.5800	208.5800

When we run this example, as shown above, we see the Freight column (which has been included just for reference) and the Freight Colors column. Notice how the contrasting text color changes depending on the background color.

Color Range Column Usage Example: Text Data

Here's an example of the Color Range Column element in use with text data.

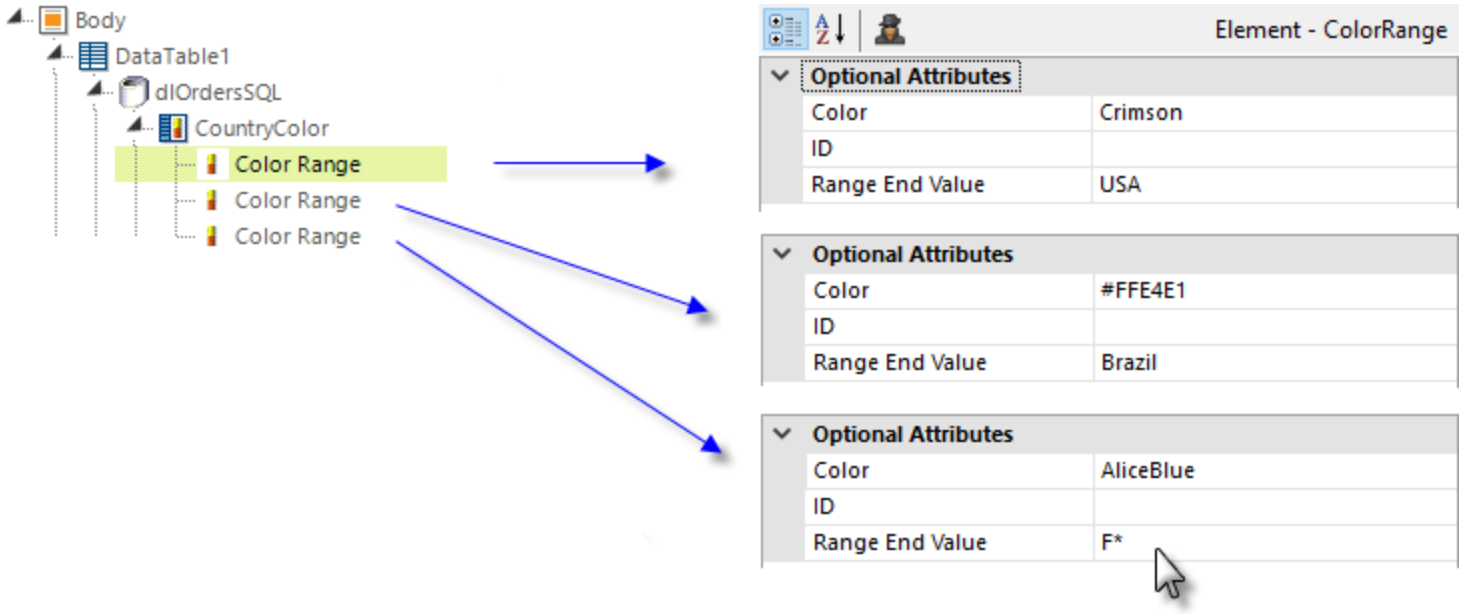
The technique used here adds two new columns to the data layer, evaluates data strings and correlates them to a background color value and contrasting text color value, and inserts those color values into the new columns. We then use the @Data tokens for those columns to set the color of the Data Table cells.



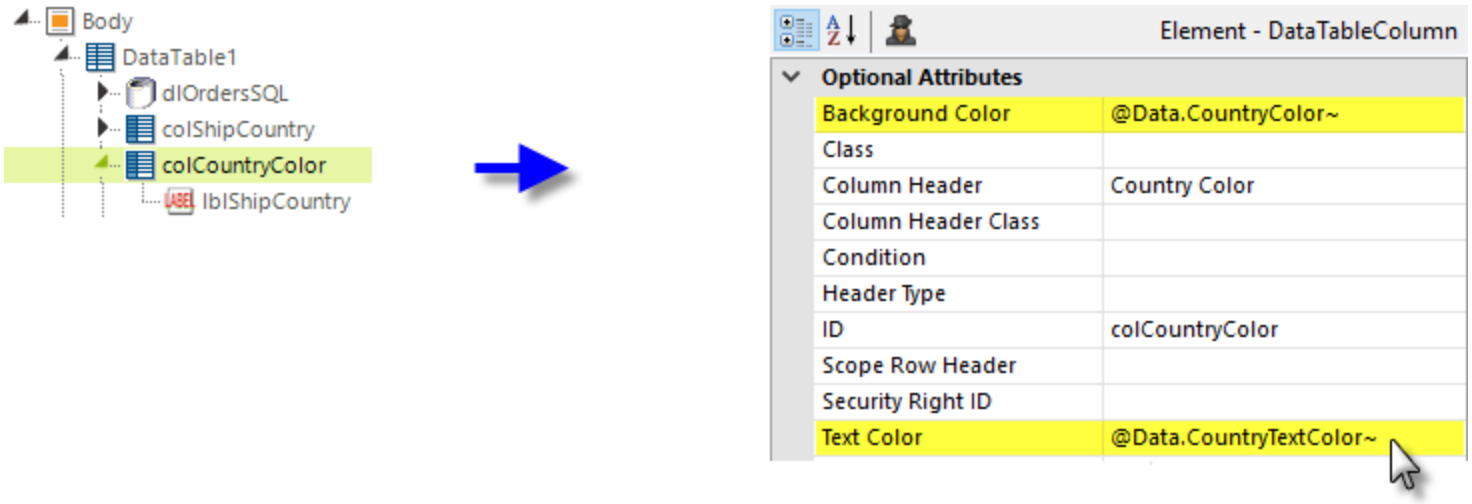
In the example shown above, we start with a Data Table element with child DataLayer.SQL element and then add a Color Range Column element. Its attributes are configured as follows:

- **Data Column** - The data to be evaluated will be in the *ShipCountry* column.
- **ID** - The new column added to the datalayer containing background color data will be named *CountryColor*.
- **Color Range Style** - The data being evaluated consists of specific text strings, so this is set to *TextList*.
- **Contrast Color Column ID** - The new column added to the datalayer containing contrasting text color data will be named *CountryTextColor*.

You can use multiple Color Range Column elements beneath a datalayer. Next, we add and configure three Color Range elements:



Color Range elements are evaluated according to their top-to-bottom order in the Element Tree, so the element order matters. Notice the use of the asterisk wildcard (*) in the bottom Range End Value attribute. *Values that do not match USA, Brazil, or F* will have no background color.* If you do wish to set a background color for all *other* text values, add a fourth Color Range element (at the "bottom" of the set of Color Range elements) and set its Range End Value to *.



In order to display the colors, we configure our Data Table Column element, as shown above. The tokens for the data in the two new columns are used to provide values in the **Background Color** and **Text Color** attributes.

ShipCountry	Country Color
France	France
Germany	Germany
Brazil	Brazil
France	France
Belgium	Belgium
Brazil	Brazil
Switzerland	Switzerland
Switzerland	Switzerland
Brazil	Brazil
Venezuela	Venezuela
Austria	Austria
Mexico	Mexico
Germany	Germany
Brazil	Brazil
USA	USA
Austria	Austria
Sweden	Sweden
France	France
Finland	Finland
Germany	Germany

When we run this example, as shown above, we see the ShipCountry column (which has been included just for reference) and the Country Color column. Notice how the contrasting text color changes depending on the background color.

The Compare Filter

The **Compare Filter** element is used with datalayer elements to filter out data rows. It's analogous to using a *WHERE* clause in a SQL query but, unlike a query, it's applied to the data *after* it's been retrieved into the datalayer.

The following topics discuss the Compare Filter element:

- Compare Filter Attributes
- Applying the Compare Filter
- Compare Filter Usage Examples
- Using Multiple and Dynamic Compare Filters
- Use with a Condition Filter

About the Compare Filter

The Compare Filter element is available for use with all datalayer elements and is particularly useful in conjunction with those lacking query capabilities, such as DataLayer.XML and DataLayer.Web Service.

In operation, the element basically compares a specified data column value against a specified literal or tokenized value and removes all rows from the datalayer where the comparison returns *False*. A variety of comparisons, such as *Equal*, *Greater Than*, *Less Than*, etc., are supported.

This element is quite similar to element, however, it frequently provides *better performance* because it's implemented as a native part of the Logi Data Engine and so does not need to execute script, as the Condition Filter does.

Compare Filter Attributes

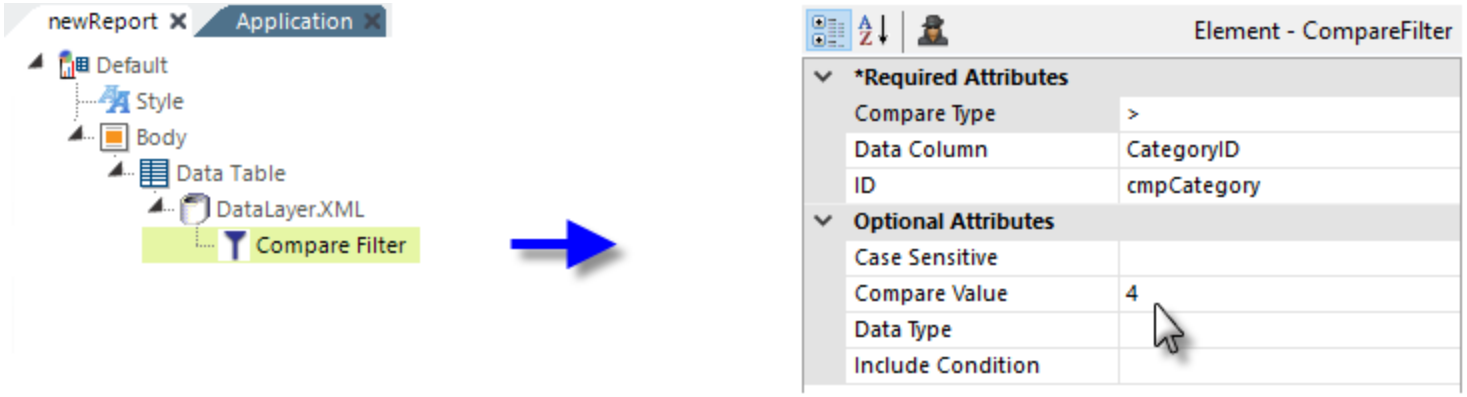
The **CompareFilter** element has the following attributes:

Attribute	Description
Compare Type	(Required) Specifies the comparison operator to be used. See the table of operators below.
Data Column	(Required) Specifies the name of the data column whose data will be compared against the Compare Value attribute value.
ID	(Required) Specifies an unique element ID.
Case Sens-itive	Specifies whether or not the comparisons of text values will be case-sensitive. Default value: *False* When used with DataLayer.ActiveSQL, this attribute may be set to *DataSourceCollation*. Case-sensitivity will then be defined by the database column's "collation" setting, which may or may not be case-sensitive. This option can provide better performance for case-insensitive filters.
Compare Value	Specifies the value that data will be compared against. Tokens may be used, but not @Data tokens for values from the parent datalayer. Leave this value blank to compare against *null*.
Data Type	Specifies the type of data being compared. Options include *Boolean*, *Date*, *DateTime*, *Number*, and *Text* and can be used to ensure unambiguous comparisons.
Include	Specifies whether or not this element will be processed when the report runs. If left blank or if it contains an

Attribute	Description
Condition	expression that evaluates to *True*, the element is processed and the new column is added to the datalayer. If it evaluates to *False*, the element is ignored and does not affect the datalayer.

Applying the Compare Filter

The following example illustrates how the **Compare Filter** is used:



1. As shown above, a **Compare Filter** element is added as a child to a datalayer element.
2. Its **Compare Type** attribute is set to define the comparison operator.
3. Its **Data Column** attribute value is set to the column in that datalayer whose value will be compared.
4. Finally, the **Compare Value** attribute is set to the value the data column values will be compared against.

Using the example above, the sequence of events is that the datalayer reads in all the data from the XML data file, then the Compare Filter **removes** all rows that don't match the criteria (CategoryID column value is greater than 4). All that remains in the datalayer for use in the report are rows whose CategoryID values are greater than 4.

To compare against Null values, leave the **Compare Value** attribute *blank*.

Comparison Operator Options

The **Compare Type** attribute value provides a number of operators in a pull-down list.

The provided operators include:

Operator	Description
=, <, >, <=, >=, < >	Basic standard comparison operations. String types evaluated are culture-sensitive.
Between	The Compare Value attribute must include two comma- or pipe character-separated values. Datalayer rows with values in the specified Data Column that are equal to, or within the range, are retained. See examples below for usage.
Contains	Datalayer rows with values in the specified Data Column that contain the Compare Value string value are retained.
ContainsAll	Compares two comma-delimited arrays, one in the specified Data Column, the other in the Compare Value attribute. For the row to be retained, all values from both arrays must be the same, in any order.
ContainsAllInOrder	Compares two comma-delimited arrays, one in the specified Data Column, the other in the Compare Value attribute. For the row to be retained, all values from both arrays must be the same, and in the same order.
ContainsAny	Compares two comma-delimited arrays, one in the specified Data Column, the other in the Compare Value attribute. For the row to be retained, just one values from either array must be the same.
ContainsNone	Compares two comma-delimited arrays, one in the specified Data Column, the other in the Compare Value attribute. For the row to be retained, none of the values from the Compare

Operator	Description
	Value attribute can be present in the specified Data Column.
EndsWith	For the row to be retained, the value in the specified Data Column mustend with the Compare Value.
FuzzyMatch	Performs a "fuzzy" string match. For the row to be retained, the value in the specified Data Column must "closely" match that in the Compare Value.
InDay	For the row to be retained, the date value in the specified Data Column must be within the same day as the Compare Value.
InList	For the row to be retained, the value in the specified Data Column *must* be one of the values specified in a comma-separated list in the Compare Value.
InWeek	For the row to be retained, the date value in the specified Data Column must be within the same week as the Compare Value.
InMonth	For the row to be retained, the date value in the specified Data Column must be within the same month as the Compare Value.
InQuarter	For the row to be retained, the date value in the specified Data Column must be within the same quarter as the Compare Value.
InYear	For the row to be retained, the date value in the specified Data Column must be within the same year as the Compare Value.

Operator	Description
IsDateTime	For the row to be retained, the specified Data Column must contain a valid date/time value.
IsDecimal	For the row to be retained, the specified Data Column must contain a valid decimal value.
IsEven	For the row to be retained, the specified Data Column must contain an even number.
IsInteger	For the row to be retained, the specified Data Column must contain a valid integer value.
IsNotDateTime	For the row to be retained, the specified Data Column must *not* contain a valid date/time value.
IsNotDecimal	For the row to be retained, the specified Data Column must *not* contain a valid decimal value.
IsNotInteger	For the row to be retained, the specified Data Column must *not* contain a valid integer value.
IsOdd	For the row to be retained, the specified Data Column must contain an odd number.
LengthEquals	For the row to be retained, the length of the specified Data Column value must equal the Compare Value.
LengthGreaterThan	For the row to be retained, the length of the specified Data Column value must be greater than the Compare Value.

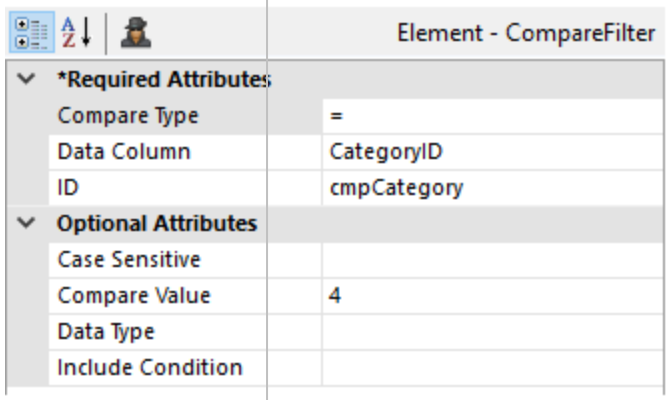
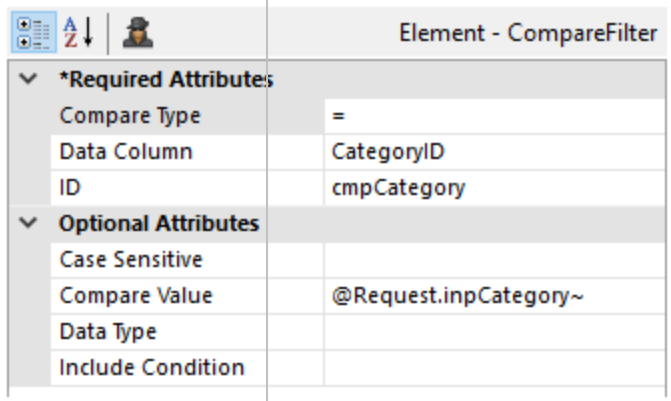
Operator	Description
LengthGreaterThanOrEqualTo	For the row to be retained, the length of the specified Data Column value must be equal to or greater than the Compare Value.
LengthLessThan	For the row to be retained, the length of the specified Data Column value must be less than the Compare Value.
LengthLessThanOrEqualTo	For the row to be retained, the length of the specified Data Column value must be less than or equal to the Compare Value.
LengthNotEqualTo	For the row to be retained, the length of the specified Data Column value must *not* equal the Compare Value.
Like	For the row to be retained, the Data Column Value must match the Compare Value.
NotBetween	The Compare Value attribute must include two comma-separated values. Datalayer rows with values in the specified Data Column that are *outside* the range, are retained.
NotContains	Datalayer rows with values in the specified Data Column that *do not* contain the Compare Value string value are retained.
NotEndsWith	For the row to be retained, the value in the specified Data Column must *not* end with the Compare Value.
NotFuzzyMatch	Performs a "fuzzy" string match. For the row to be retained, the value in the specified Data Column must *not* "closely" match that in the Compare Value.

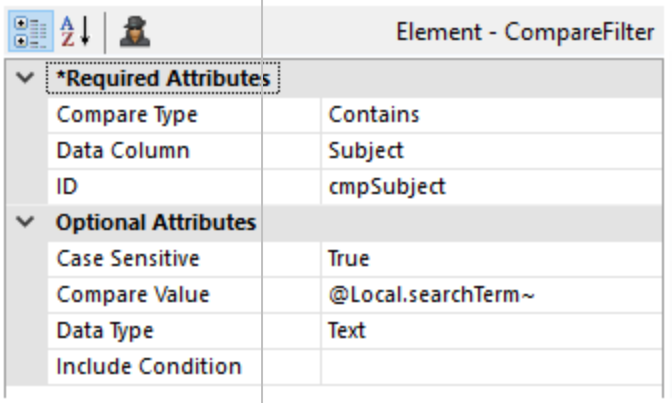
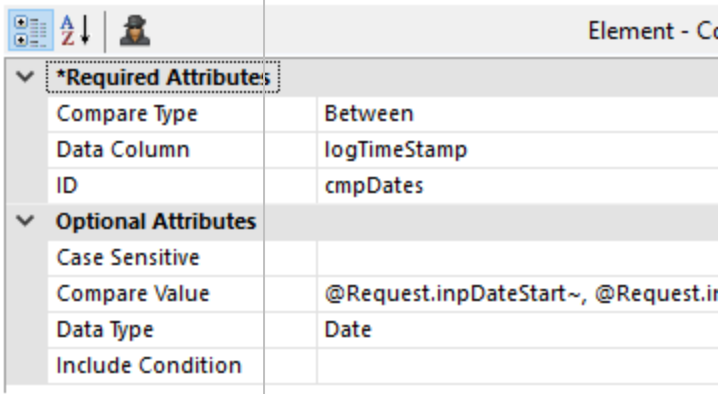
Operator	Description
NotInList	For the row to be retained, the value in the specified Data Column must *not* be one of the values specified in a comma-separated list in the Compare Value.
NotLike	For the row to be retained, the Data Column Value must *not* match the Compare Value.
NotRegEx	For the row to be retained, the value in the specified Data Column must *not* match the Regular Expression in the Compare Value.
NotStartsWith	For the row to be retained, the value in the specified Data Column must *not* start with the Compare Value.
RegEx	For the row to be retained, the value in the specified Data Column must match the Regular Expression in the Compare Value.
StartsWith	For the row to be retained, the value in the specified Data Column must start with the Compare Value.

You may also type in your own operators, such as *!=* (not equal).

Compare Filter Usage Example

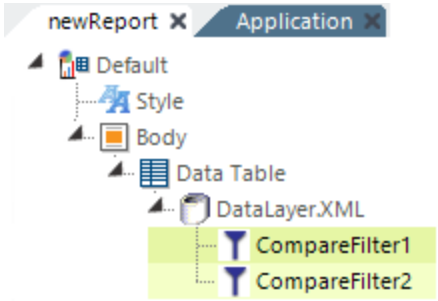
Here are some common usage examples:

Usage/Scenario	Configuration
Filtering on a Number	
Filtering on a Token Value @Data tokens for values in the parent datalayer are not usable for comparisons.	

Usage/Scenario	Configuration
Filtering on Text	
Filtering on Date Range	

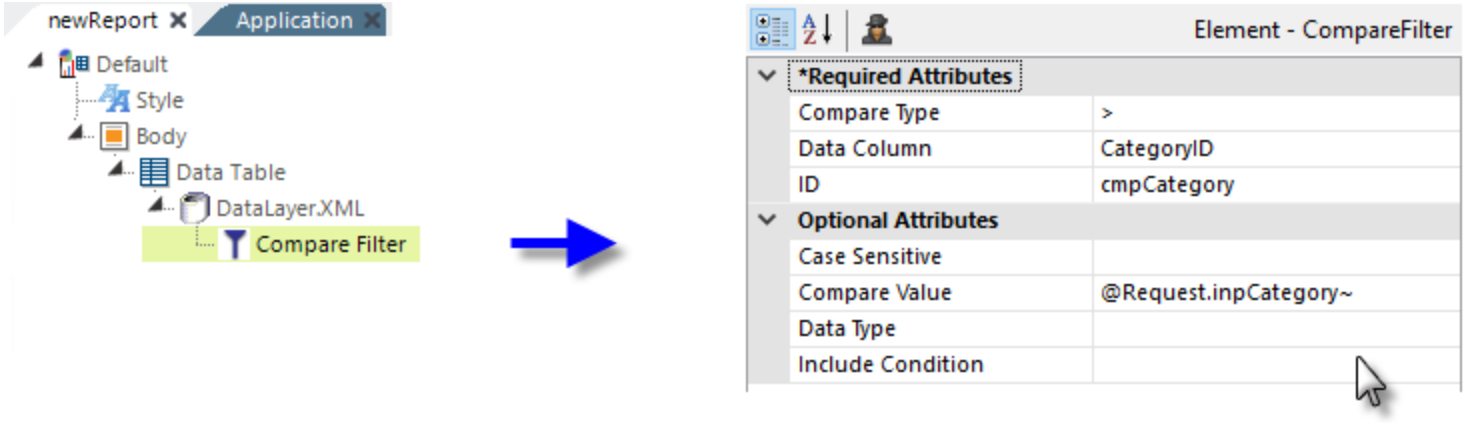
Using Multiple and Dynamic Compare Filters

Developers can use two or more *consecutive* Compare Filter elements:



In this case, the comparisons will be evaluated *sequentially*, based on their arrangement from top to bottom in the element tree. So, after the first Compare Filter is applied, only records that meet its criteria will remain in the datalayer to be evaluated for the second Compare Filter.

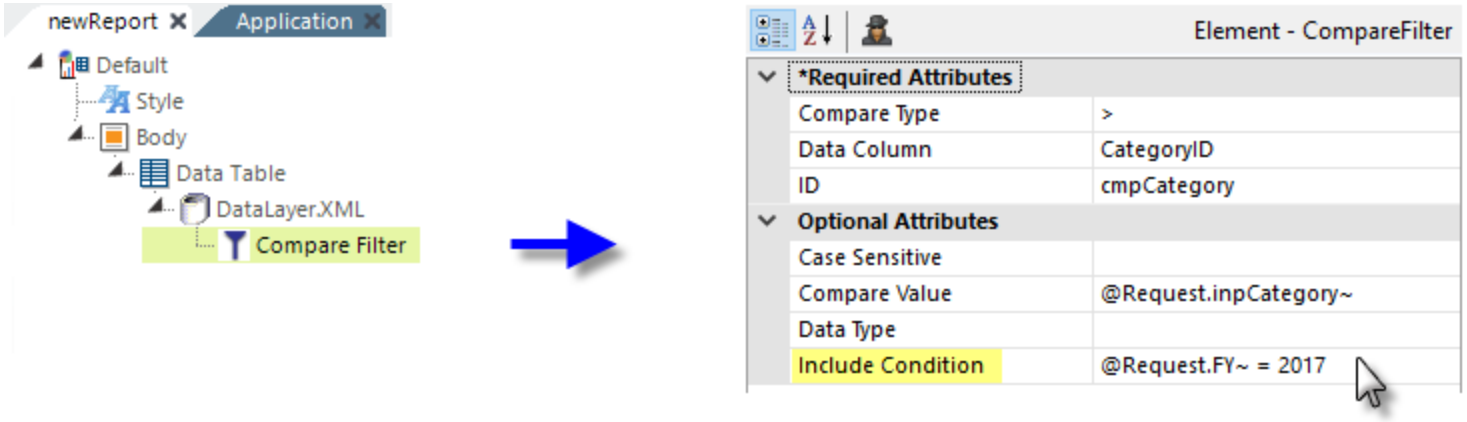
Developers can make Compare Filter elements dynamic by using tokens:



As mentioned earlier and shown above, the Compare Value attribute can contain tokens such as @Request, so that comparison values are dynamic at runtime.

Sometimes it is necessary to ensure that tokens have default values when running a report. When using @Request tokens, the **Default Request Params** element can be used to create default values for each **@Request** token.

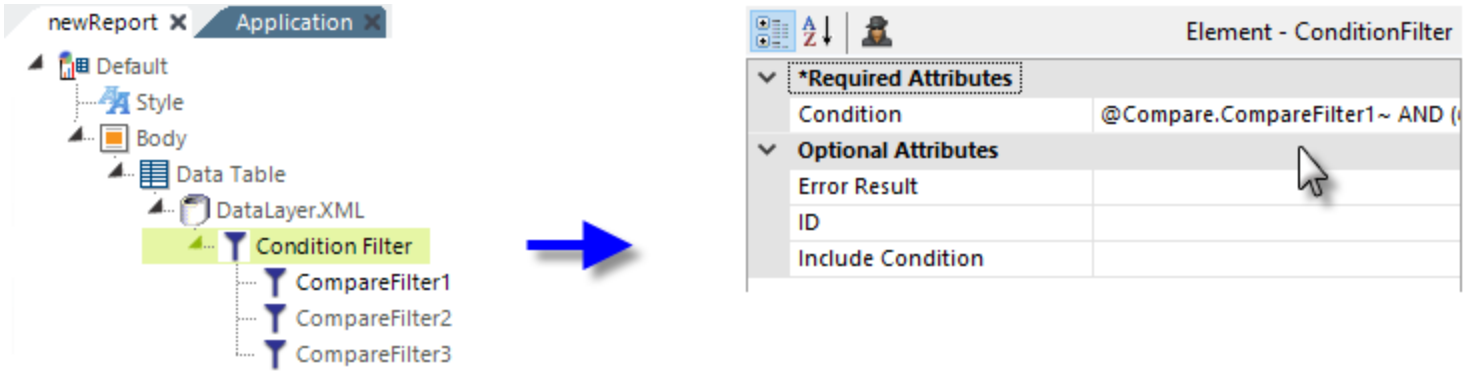
The Compare Filter element also has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows developers to dynamically determine if the datalayer will be filtered or not.

Use with a Condition Filter

Compare Filters can also be used with Condition Filters to create complex expressions that include ANDs and ORs and levels of parentheses. For more information, see "The Condition Filter" on page 248.



For example, consider the element arrangement shown above. For each row of the datalayer, the three **Compare Filters** will be evaluated first. Each of them will produce a *True* or *False* value as a result of their comparison operations. Those values will be assigned to @Compare tokens which include the element ID, for example: `@Compare.CompareFilter1~`. The scope of these tokens is limited to the parent Condition Filter.

After all of the Compare Filters are evaluated, the **Condition Filter** will be evaluated and its **Condition** attribute value can include expressions that use the @Compare tokens. For example, the Condition attribute value above might be:

```
@Compare.CompareFilter1~ AND (@Compare.CompareFilter2~ OR @Compare.CompareFilter3~)
```

And, of course, you can also use other types of tokens and literal values in the Condition attribute to get the expression you need, such as:

```
@Data.UnitPrice~ <= 100 AND (@Compare.CompareFilter1~ OR @Compare.CompareFilter2~)
```

You may be able achieve the same results with complicated scripting for the Condition attribute value, however, in most cases use of multiple Compare Filters as shown in this example will produce better performance than scripting.

The Contain Filter

The **Contain Filter** element is used with datalayer elements to filter out data rows. It's analogous to using a *WHERE*x *CONTAINS* *y* clause in a SQL query but, unlike a query, it's applied to the data *after* it's been retrieved into the datalayer.

The following topics discuss the Contain Filter element:

- Applying the Contain Filter
- Using Multiple and Dynamic Contain Filters

About the Contain Filter

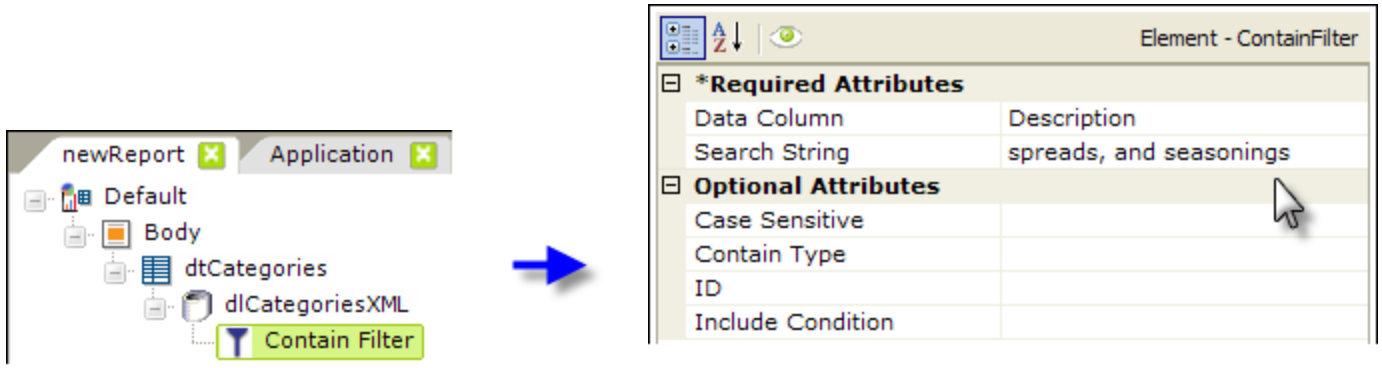
The Contain Filter element is available for use with all datalayer elements and is particularly useful in conjunction with those lacking query capabilities,such as DataLayer.XML and DataLayer.Web Service.

In operation, the element basically searches a specified data column valuefor a specified text string and removes all rows from the datalayer where the string is not found in the column.

This element is similar to element, however, it frequently provides *better performance* because it's implemented as a native part of the Logi Data Engine and so does not need to execute script, asthe Condition Filter does.

Applying the Contain Filter

The following exampleillustrates how the **Contain Filter** is used:



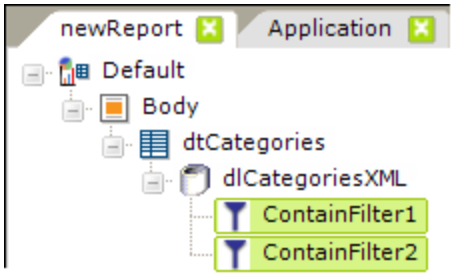
1. As shown above, a **Contain Filter** element is added as a child to a datalayer element.
2. Its **Data Column** attribute value is set to the column in that datalayer whose value will be searched.
3. The **Search String** attribute is set to the text you're searching for in the data column.
4. The optional **Case Sensitive** attribute sets the case-sensitivity of the search. The default value is *True*, for a case-sensitive search. Setting it to *False* produces a case-insensitive search.
5. The optional **Contain Type** attribute is set to define the nature of the search. The default value is *Contains*, which means the data column value must contain the search string. The other values, *StartsWith* and *EndsWith*, let you make the search more selective about where the search string occurs in the column value: at its beginning or at its end.

In the example above, the sequence of events is that the datalayer reads in all the data from the XML data file, then the Contain Filter **removes** all rows that don't match the criteria (Description column value contains "spreads, and seasonings"). All that remains in the datalayer for use in the report are rows whose Description column value contains "spreads, and seasonings".

Using Multiple and Dynamic Contains

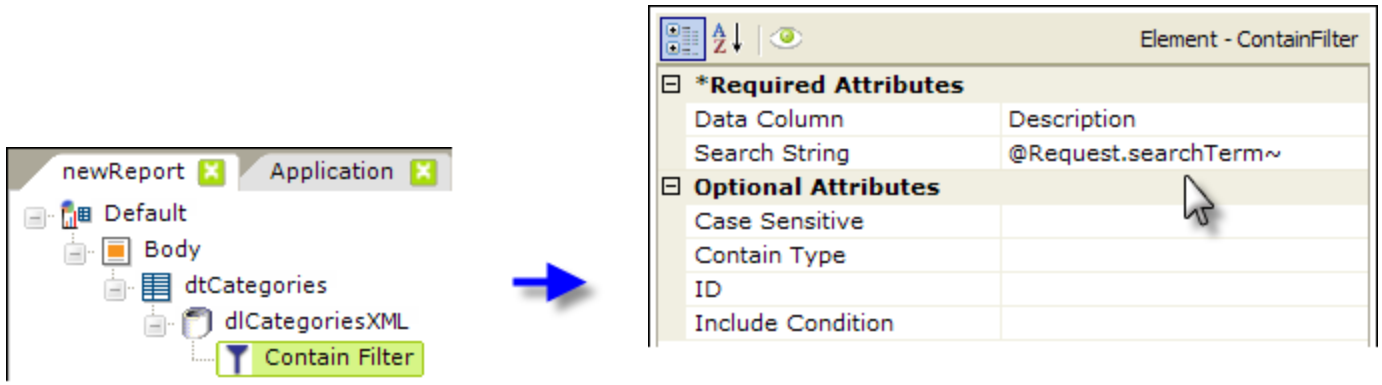
This topic describes how to use multiple and dynamic contains.

1. Developers can use two or more **consecutive** Contain Filter elements:



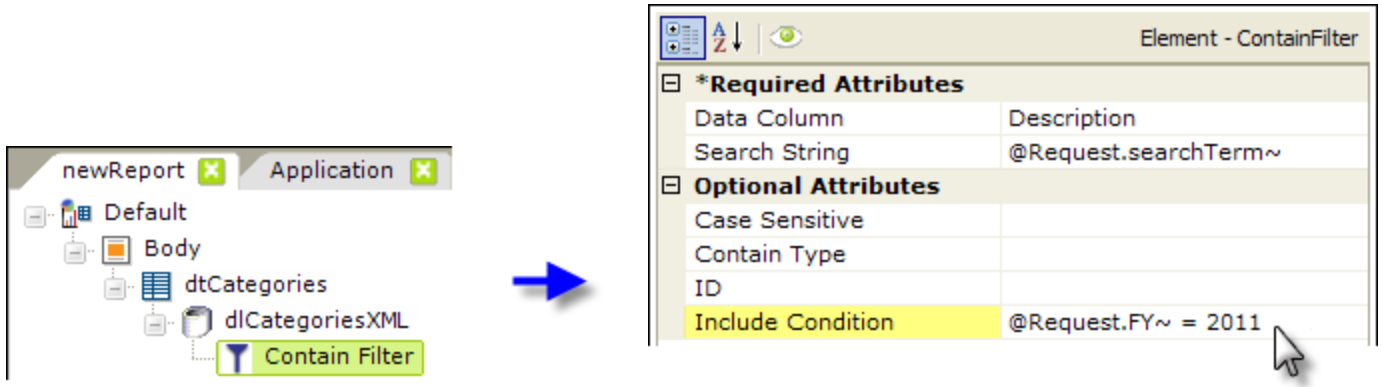
In this case, the searches will be conducted **sequentially**, based on their arrangement from top to bottom in the element tree. So, after the first Contain Filter is applied, only records that meet its criteria will remain in the datalayer to be evaluated bythe second Contain Filter.

2. Developers can make Contain Filter elements **dynamic** by using tokens:



As shown above, the Search String attribute value can contain tokens such as @Request, so that comparison values are dynamic at runtime. Sometimes it is necessary to ensure that tokens have default values when running a report. When using @Request tokens, the **Default Request Params** element can be used to create default values for each **@Request** token.

3. The Contain Filter element also has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows developers to dynamically determine if the datalayer will be filtered or not.

The Condition Filter

The **Condition Filter** element is used with datalayer elements to filter out data rows that don't meet certain criteria.

The following topics discuss the use of this element:

- About the Condition Filter
- Applying the Condition Filter
- Using Multiple and Dynamic Condition Filters
- Use with Compare Filters

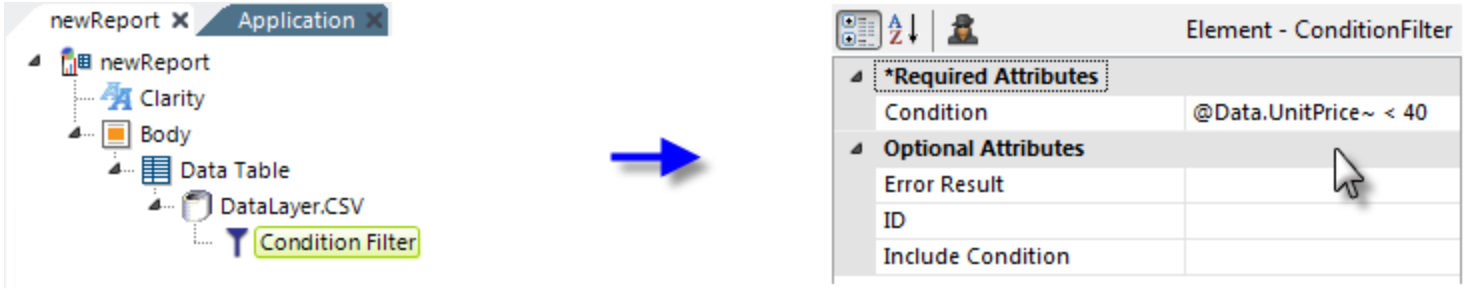
 In many circumstances you will find that "The Compare Filter" on page 228 provides similar functionality and better performance.

About the Condition Filter

The **Condition Filter** element is used with datalayer elements to filter out data rows that don't meet specified criteria. It's analogous to using a *WHERE* clause in a SQL query but, unlike a query, it's applied to data *after* it's been retrieved into the datalayer. The Condition Filter is available to all datalayer elements but is most useful when used with those lacking conditional query capabilities, such as DataLayer.XML and DataLayer.CSV.

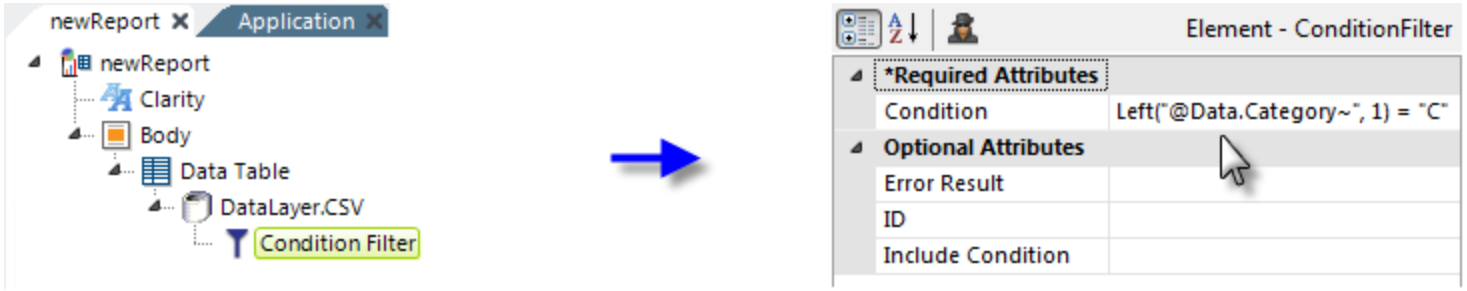
Applying the Condition Filter

The following example illustrates how the **Condition Filter** is used:



1. As shown above, a Condition Filter element is added as a child of a datalayer element.
2. Its **Condition** attribute value is set to a formula that compares the data in the datalayer to some value. Note that no leading equals sign (=) is required for the expression here.

When the example is executed, the datalayer reads in all the data from the CSV data file, then the Condition Filter **removes** all records other than those where the condition formula evaluates to *True;* in this case, where the values in the *UnitPrice* column are greater than or equal to 40. All that remains in the datalayer for use in the report are records whose *UnitPrice* values are less than 40. The Condition Filter element's **Condition** attribute supports built-in functions (see Built in Functions and Operators) and JavaScript functions and can evaluate them to filter data:

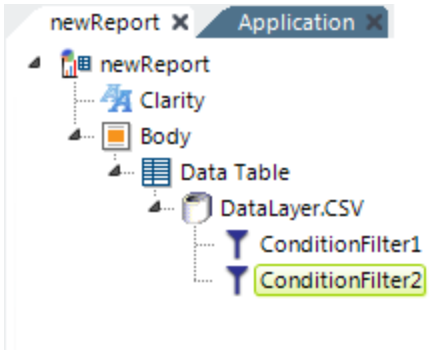


1. As shown above, a Condition Filter element is added as a child of the datalayer element.
2. Its **Condition** attribute value is set to a formula that uses the built-in function *Left()* to compare the data in the datalayer to some value. Note that no leading equals sign (=) is required here.

When executed, the datalayer reads in all the data from the CSV data file, then the Condition Filter evaluates the formula and **removes** all records where the first letter of the *Category* column is not the letter *"C".* All that remains in the datalayer for use in the report are records whose *Category* column values begin with "C". Compound expressions, using AND and OR operators, can also be used in the Condition attribute.

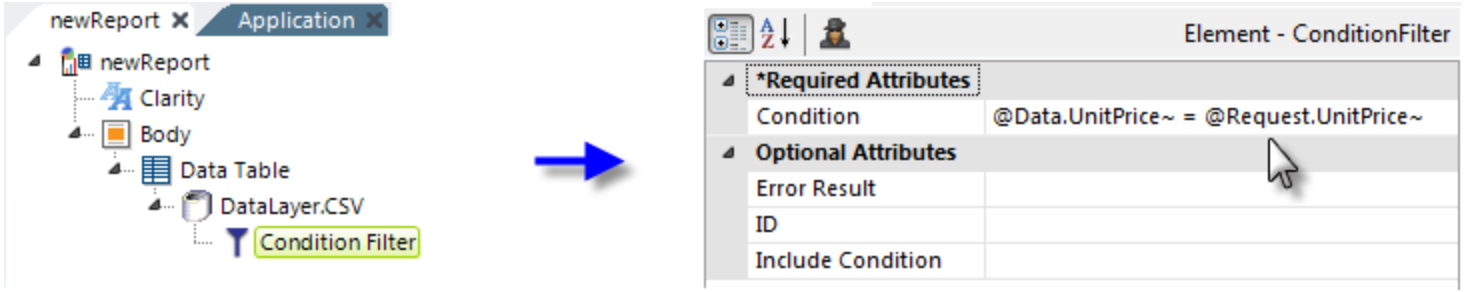
Using Multiple and Dynamic Condition Filters

You can use two or more Condition Filter elements together:



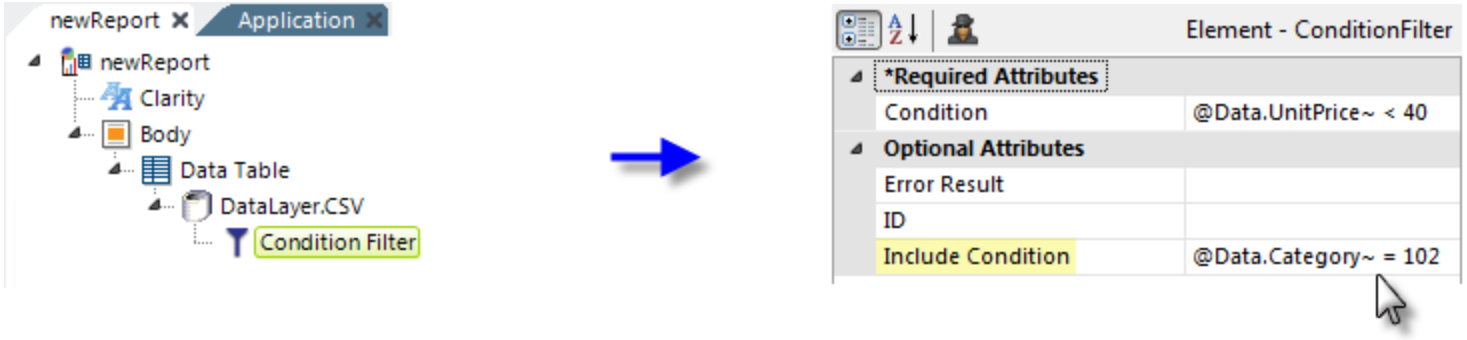
In this case, the conditions will be evaluated *sequentially*, based on their arrangement from top to bottom in the element tree. So, in the example above, after the first condition filter is applied, only records that meet its criteria will remain in the datalayer to be evaluated by the second condition filter.

You can also make Condition Filter elements *dynamic* by using tokens within expressions:



As shown above, condition formulae can contain tokens such as @Request, so that comparison values are dynamic at runtime. Sometimes it is necessary to ensure that tokens have default values when running a report. When using @Request tokens, the **Default Request Params** element can be used to create default values for each Request variable.

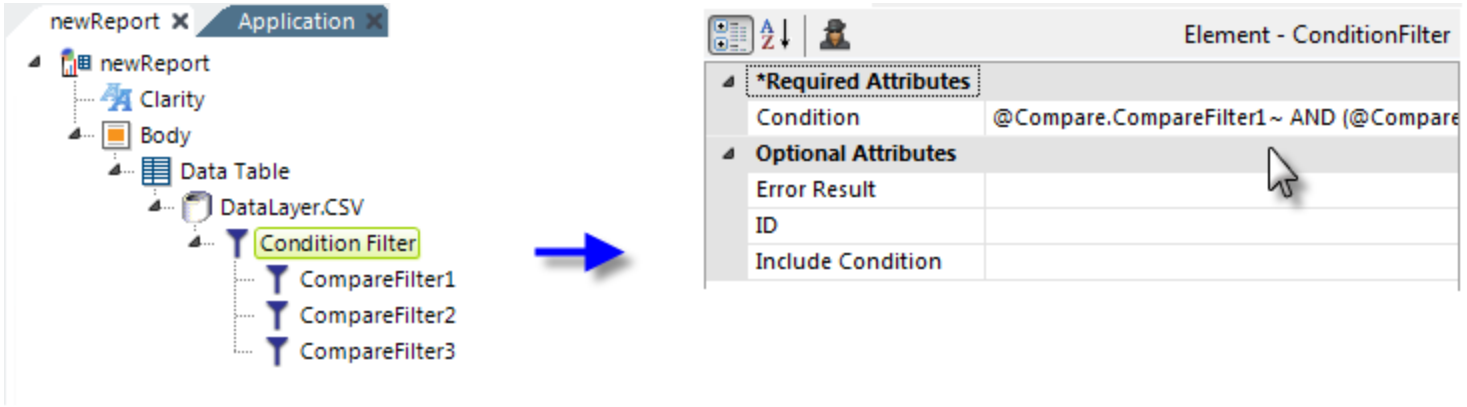
Like other filter elements, the Condition Filter has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows you to dynamically determine if the datalayer will be filtered or not.

Use with Compare Filters

For improved performance, Condition Filters can also be used in combination with "The Compare Filter" on page 228 to create complex expressions that include ANDs and ORs and levels of parentheses.



For example, consider the element arrangement shown above. For each row of the datalayer, the three **Compare Filters** will be evaluated first. Each of them will produce a *True* or *False* value as a result of their comparison operations. Those values will be assigned to @Compare tokens which include the element ID, for example: `@Compare.CompareFilter1~`. The scope of these tokens is limited to the parent Condition Filter. After all of the Compare Filters are evaluated, the **Condition Filter** will be evaluated and its **Condition** attribute value can include expressions that use the @Compare tokens. For example, the Condition attribute value above might be:

```
@Compare.CompareFilter1~ AND (@Compare.CompareFilter2~ OR @Compare.CompareFilter3~)
```

And, of course, you can also use other types of tokens and literal values in the Condition attribute to get the expression you need, such as:

```
@Data.UnitPrice~ <= 100 AND (@Compare.CompareFilter1~ OR @Compare.CompareFilter2~)
```

You may be able achieve the same results with complicated scripting for the Condition attribute value, however, in most cases use of multiple Compare Filters as shown in this example will produce better performance than scripting.

The Crosstab Filter

The **Crosstab Filter** element is used with datalayer elements to "pivot" *columns* of data into *rows* of data. It converts related rows of data into a single row with multiple columns.

The following topics discuss the Crosstab Filter element:

- Applying the Crosstab Filter to a Crosstab Table
- Applying the Crosstab Filter to a Chart
- Using Dynamic Crosstab Filters

About the Crosstab Filter

The **Crosstab Filter** element is available for use with all datalayer elements and is the functional opposite of the UnCrosstab Filter element. In operation, the element basically converts all rows in the datalayer which share a common identifying value into a single row with multiple columns that contain the column values from all the original rows.

Before filter is applied:

CustomerID	CompanyName	ProductName	Quantity
ALFKI	Alfreds Futterkiste	Rössle Sauerkraut	17
ALFKI	Alfreds Futterkiste	Aniseed Syrup	6
ALFKI	Alfreds Futterkiste	Chartreuse verte	21
ALFKI	Alfreds Futterkiste	Spegesild	2
ALFKI	Alfreds Futterkiste	Vegie-spread	20
ALFKI	Alfreds Futterkiste	Flotemysost	20
ALFKI	Alfreds Futterkiste	Lakkalikööri	15

After filter is applied:

Cust ID	Company	Rössle Sauerkraut	Aniseed Syrup	Chartreuse verte	Spegesild	Vegie-spread	Flotemysost	Lakkalikööri
ALFKI	Alfreds Futterkiste	17	6	21	2	20	20	15

It's a little easier to understand the filter's actions when you can see the effect limited to one set of related rows, as shown above.

This "pivot" action denormalizes the data on the fly and, among other applications, can be really useful when having to work with data imported from spreadsheets.

To configure a Crosstab Filter, you specify datalayer columns as the **Crosstab Column**, the **Label Column**, and the **Value Column**, and select a **Value Function** (*Any*, *Average*, *Count*, *DistinctCount, Median*, *Mode*, *StdDev*, or *Sum*).

When the filter runs, a row is generated in the resulting datalayer for each distinct value in the designated Label Column, and a column is generated for each distinct value found in the designated Crosstab Column. The columns or "cells" in the resulting datalayer are derived from the designated Value Column by adding, counting, averaging, etc. (depending on the Value Function

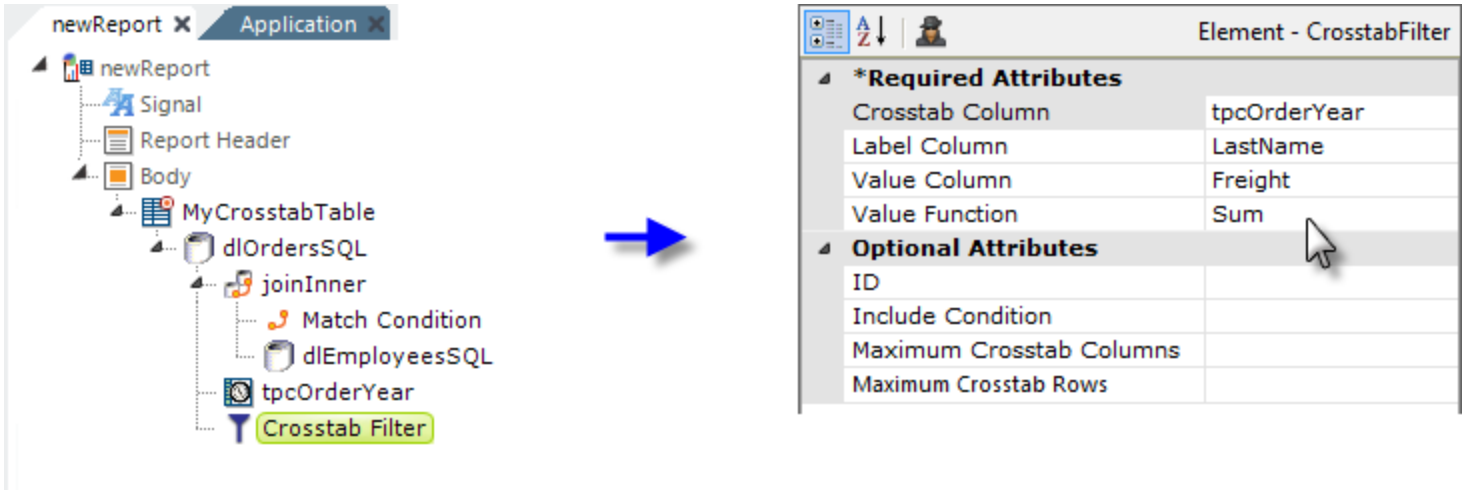
chosen) all the values unique to each Crosstab Column and Label Column. You can also group and summarize Value Columns and Label Columns by adding **Secondary Label Columns** and **Summary Rows** to your Crosstab Filter.

Datalayer rows that initially have blank Label Column values or blank Crosstab Column values are *not* included by default in the resulting datalayer. To include these rows, add a Calculated Column child element to the datalayer *before* the Crosstab Filter, configure it to provide a non-blank value if a blank value is found, then use it as the Crosstab Column or Label Column.

Other datalayer filter elements you may choose to add to your definition must not be configured to affect the columns designated when configuring the Crosstab Filter.

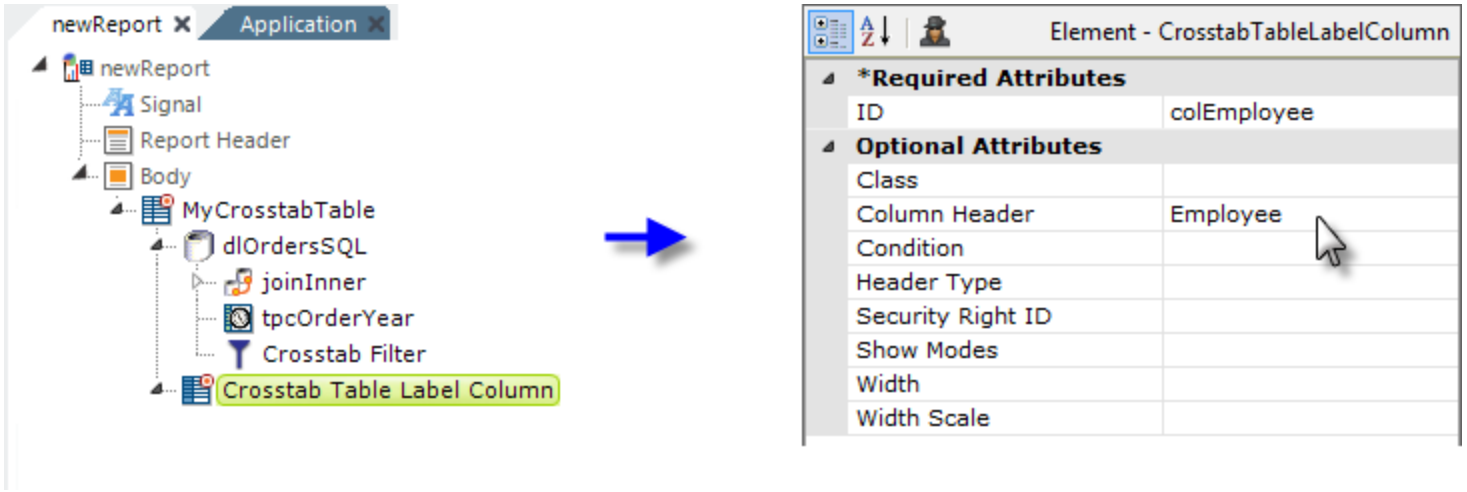
Applying the Crosstab Filter to a Crosstab Table

The following example illustrates how the **Crosstab Filter** is used with a Crosstab Table:

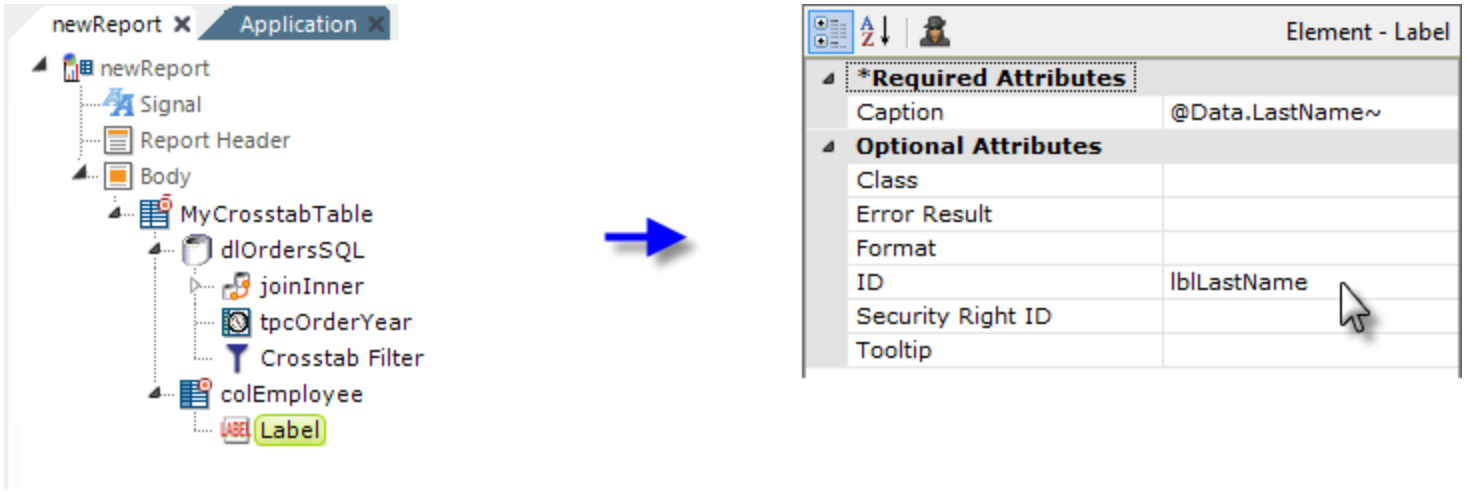


1. Our goal is show total freight costs per employee per year. As shown above left, we start by retrieving Order data and joining it with to Employee data, based on Employee ID, which gives us the order data and the last name of the employee responsible for the order.
2. We want to show yearly summaries so, because the Order data is time-related, we use a **Time Period Column** element to parse out the Year value from the *OrderDate* column. Time Period Columns make working with parts of dates incredibly easy.
3. Then we add a **Crosstab Filter** element beneath a datalayer, and configure is as follows:
4. Set its **Crosstab Column** attribute to the data columns we want to see across the top. These will be the OrderDate years, so enter the *tpcOrderYear* column name here.
5. Set its **Label Column** attribute to the row labels we want to see down the left. These will be the employees' last names, so enter the *LastName* column here.

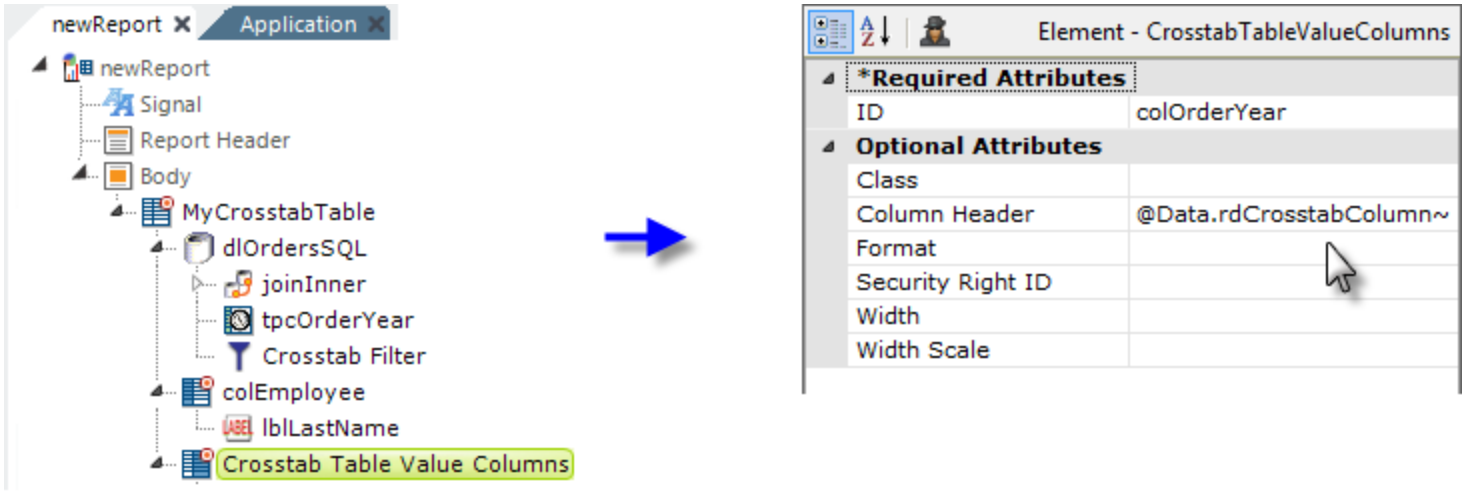
6. Set its **Value Column** attribute to the aggregated values we want to see at the intersections of the previous to items. This will be the freight cost, so enter the Freight column name here.
7. Finally, set its Value Function to the type of aggregation we want performed on the freight cost, in this case *Sum*.



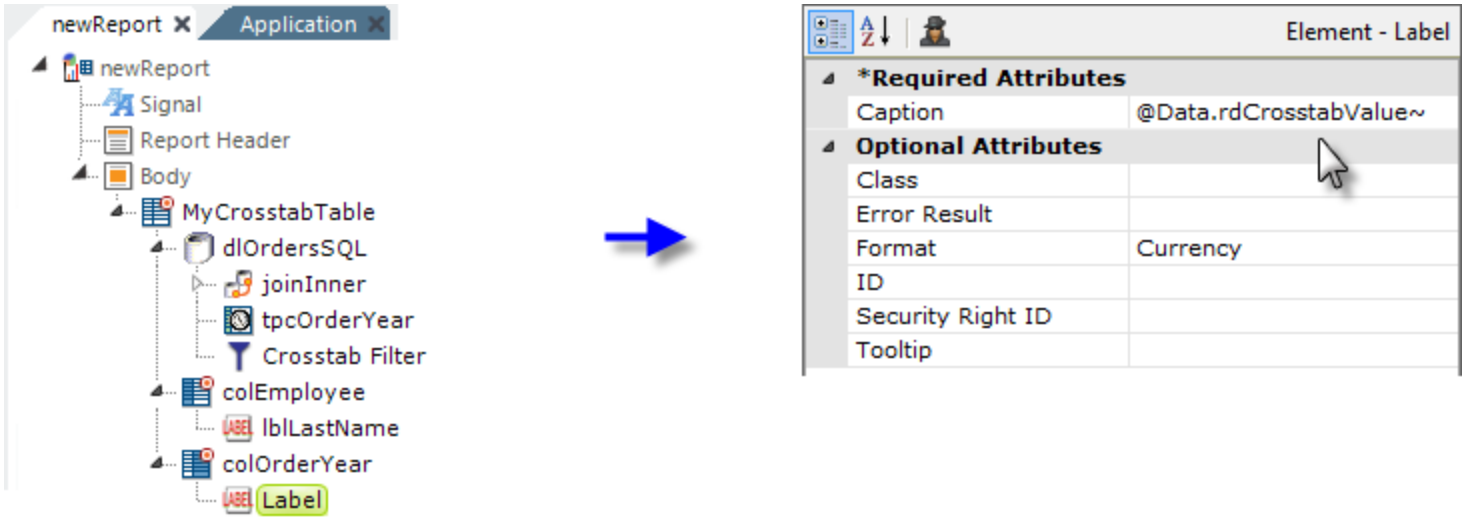
8. Next we add a **Crosstab Table Label Column** element, as shown above left. This is the column our "label" information (the employee last name) will appear in and it's configured as shown.



9. Then, as shown above, we add a **Label** element beneath the column element to actually display the data, and set its **Caption** to the @Data token for the value (the employee last name) we want to show.



10. Now things get a little more unusual: next we add a **Crosstab Table Value Columns** element, as shown above. This creates the columns (more than one) that will contain the annual aggregated freight value for each employee. So, it's unusual in that it creates *multiple* columns and also because it uses one of the **special Crosstab Table tokens** in its Column Header attribute, as shown above right. This token allows the column header to change depending on the data.



11. And finally, we add a Label element to display the data. Its Caption attribute value is provided using another special Crosstab Table token, as shown above. This provides the aggregated freight value for the right employee and year.

Now, let's see the effect of all this work:

EmpID	OrderID	CustID	OrderDate	Required	Shipped	Freight	LastName	tpcOrderYear
3	10251	VICTE	7/8/2010 12:00:00 AM	8/5/2010 12:00:00 AM	7/15/2010 12:00:00 AM	41.34	Leverling	2010
3	10253	HANAR	7/10/2010 12:00:00 AM	7/24/2010 12:00:00 AM	7/16/2010 12:00:00 AM	58.17	Leverling	2010
3	10256	WELLI	7/15/2010 12:00:00 AM	8/12/2010 12:00:00 AM	7/17/2010 12:00:00 AM	13.97	Leverling	2010

If we look at the data *prior* to passing it through the Crosstab Filter, the datalayer looks like the example shown above.

Employee	2010	2011	2012
Davolio	$1,871.04	$4,584.47	$2,381.13
Fuller	$973.18	$3,796.41	$3,926.82
Leverling	$880.03	$6,918.34	$3,086.37
Peacock	$1,968.59	$6,648.60	$2,728.95
Buchanan	$1,365.37	$1,184.75	$1,368.59
Suyama	$766.10	$2,114.17	$900.20
King	$664.32	$3,240.66	$2,760.46
Callahan	$1,258.40	$2,935.28	$3,294.20
Dodsworth	$532.84	$1,046.09	$1,747.33

And here's what it looks like in the Crosstab Table.

Let's review the **special Crosstab Table tokens**, two of which were used to produce the results shown above:

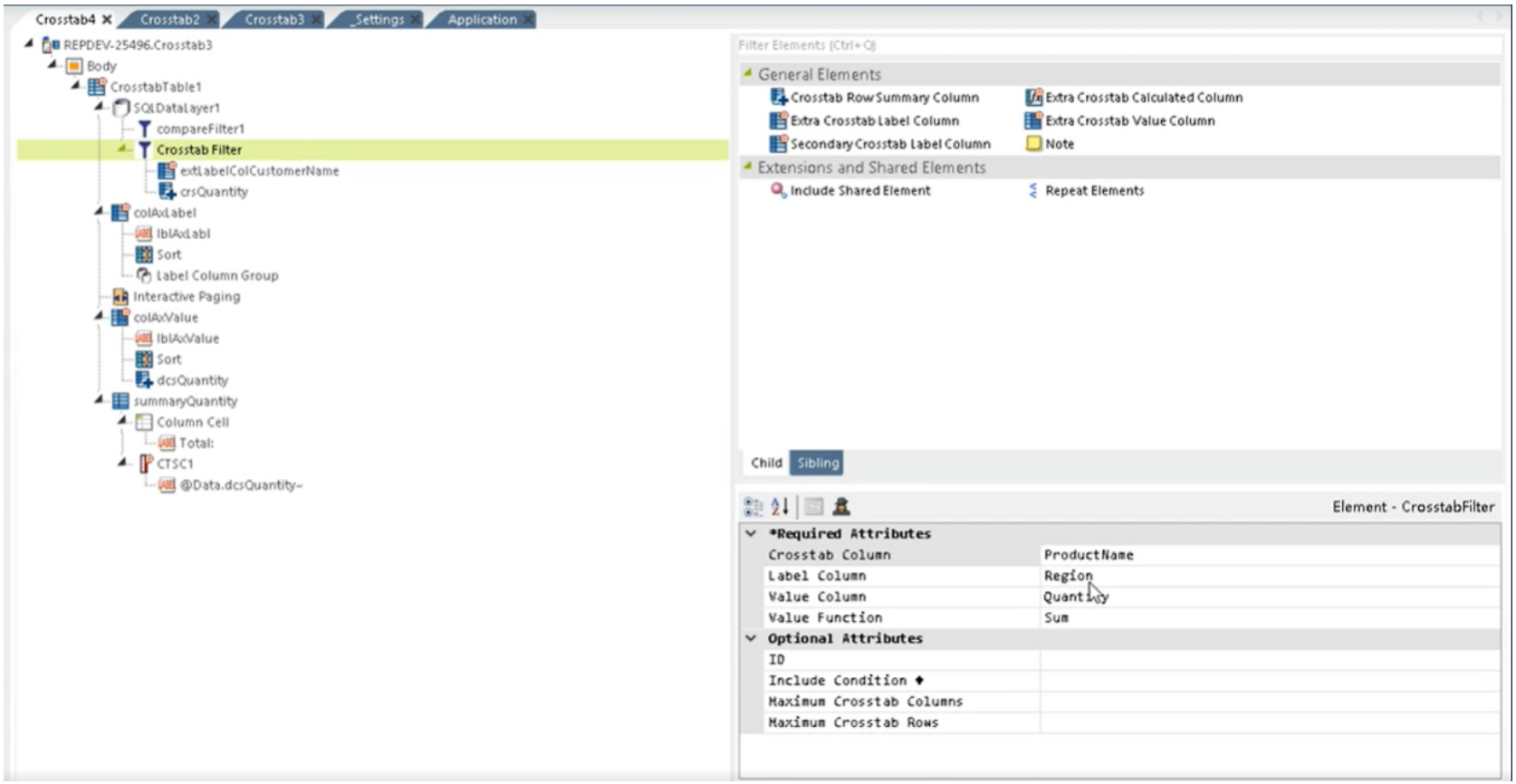
Token	Description
@Data.rdCrosstabColumn~	The current Crosstab Table column header value.
@Data.rdCrosstabValue~	The aggregated value for each label and column.
@Data.rdCrosstabValCount~	The number of values aggregated to get the previous token's value.

As we've seen, the Crosstab Table, using the Crosstab Filter, is a powerful tool for aggregating data and grouping it, especially by time periods.

Crosstab Grouping

You can incorporate Crosstab grouping by adding the **Secondary Crosstab Label Columns** to your Crosstab Filter. This element works like the Extra Crosstab Label Columns element, except it can be used to group Value Columns and Label Columns together.

In this example, the Crosstab Filter is grouped by the Label Column *Region*:

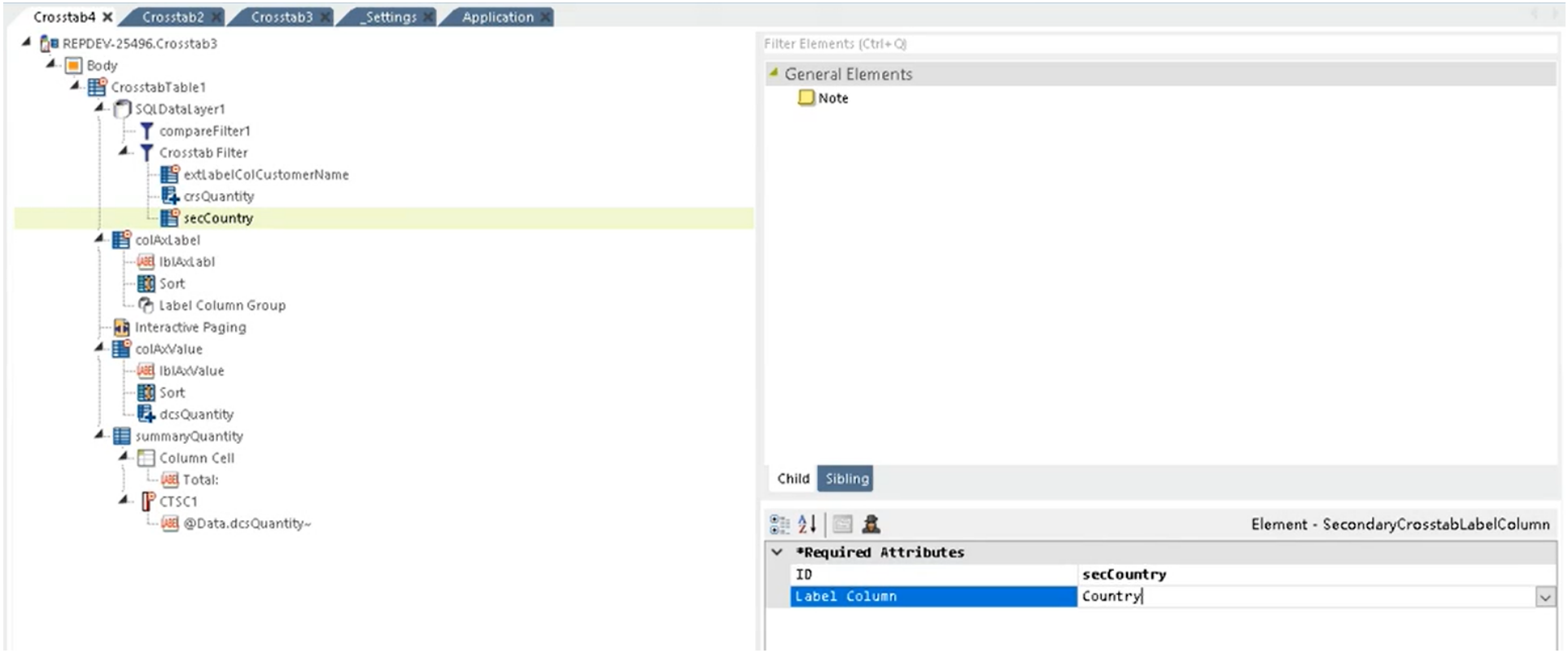


Here's what the Crosstab Table looks like, as is:

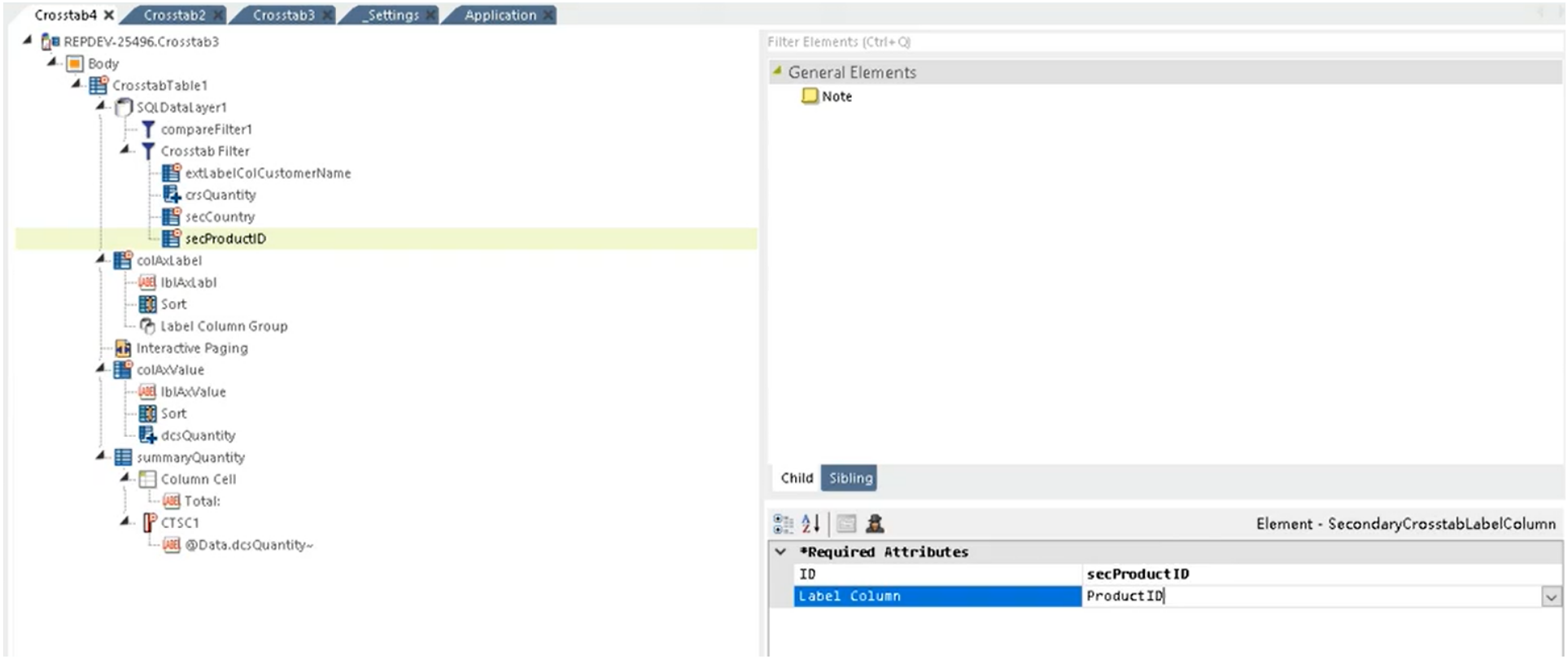
ProductName by Country on Sum of Quantity

Region	Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
Asia-Pacific	994	19,276	5,034	656							
Europe, Middle East, Africa	1,624	15,255	1,572		1,319	2,044	1,200	3,355	1,200	3,355	615
Latin America		27,684					22,761				
North America	665	3,419		1,779	79	9,162	2,894	3,824		39,928	
Total:	3283	65634	6606	2435	1398	11206	26855	7179	1200	43283	615

1. First, add a **Secondary Crosstab Label Column** and give it an ID and assign a Label Column. In this example, we're adding "secCountry" with the Label Column "Country":

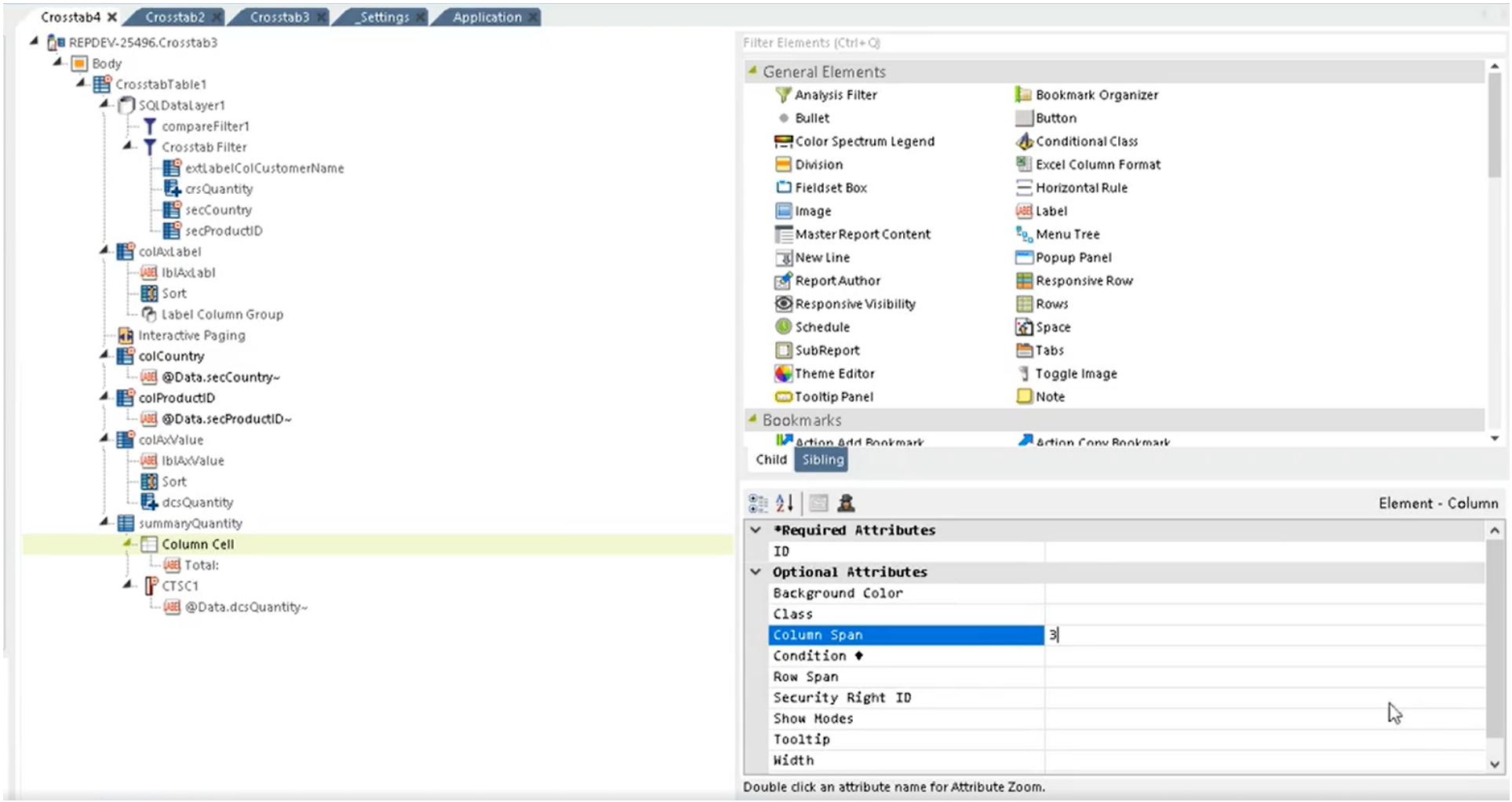


2. Then, add another **Secondary Crosstab Label Column** and give it an ID and assign a Label Column. In this example, we're adding "secProductID" with the Label Column "ProductID":



3. Add two **Crosstab Table Label Columns** to the Crosstab Filter that reflect the Secondary Crosstab Label Columns. In this example, we're adding a Crosstab Table Label Column with the ID "colCountry", Column Header titled "Country", and a Label element captioned "@Data.secCountry~" and another Crosstab Table Label Column with the ID "colProductID", Column Header titled "Product ID", and a Label element captioned "@Data.secProductID~".

4. Adjust the Column Span of the Summary Column to reflect the number of Label Columns you are grouping together. In this example, we're adjusting the span to 3 columns:

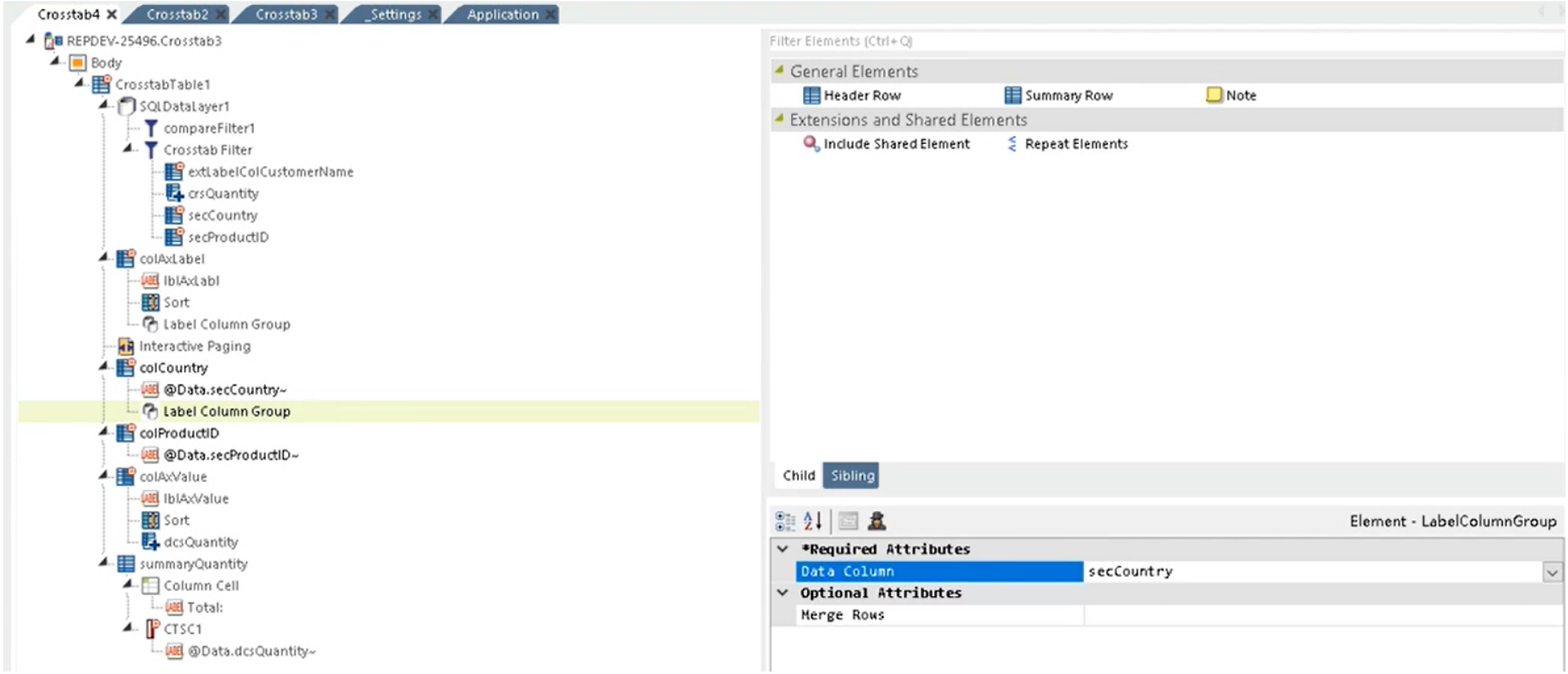


5. Select **Save** and **refresh** your report. Info displays the newly added *Country*, and *ProductID* Secondary Crosstab Label Columns:

ProductName by Country on Sum of Quantity

Region	Country	Product ID	Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
Asia-Pacific	Australia	1	798										
	Australia	6		3,306									
	China	6		9,738									
	China	23			5,034								
	Japan	6		6,232									
	Malaysia	1	196										
	Malaysia	2				656							
Europe, Middle East, Africa	Belgium	1	854										
	Belgium	13					264						
	Belgium	20						2,044					
	Belgium	23			1,572								
	Poland	6		5,517									
	Poland	8							1,200				
	Russia	1	343										
	Russia	13					1,055						
	Saudi Arabia	6		9,738									
	Spain	1	427										
	Spain	25								3,355			
	Spain	26									1,200		
	Spain	28										3,355	
Total:			3283	65634	6606	2435	1398	11206	26855	7179	1200	43283	615

6. You can go one step further and add a **Label Column Group** element for each of the Data Columns you want to group. In this example, we're adding a Label Column Group for "Region" and "secCountry":



You can also change the Merge Rows attribute to "True" (default value is "False") to merge the cells grouped together. You must do this for each Label Column Group you add.

7. Select **Save** and **refresh** your report. Info groups the *Region* and *Country* columns:

ProductName by Country on Sum of Quantity

Region	Country	Product ID	Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
Asia-Pacific	Australia	1	798										
		6		3,306									
	China	6		9,738									
		23			5,034								
	Japan	6		6,232									
	Malaysia	1	196										
		2				656							
Europe, Middle East, Africa	Belgium	1	854										
		13					264						
		20						2,044					
		23			1,572								
	Poland	6		5,517									
		8							1,200				
	Russia	1	343										
		13					1,055						
	Saudi Arabia	6		9,738									
	Spain	1	427										
		25								3,355			
		26									1,200		
		28										3,355	
Total:			3283	65634	6606	2435	1398	11206	26855	7179	1200	43283	615

Here's what your Crosstab Table looks like if you *merged* the cells:

ProductName by Country on Sum of Quantity

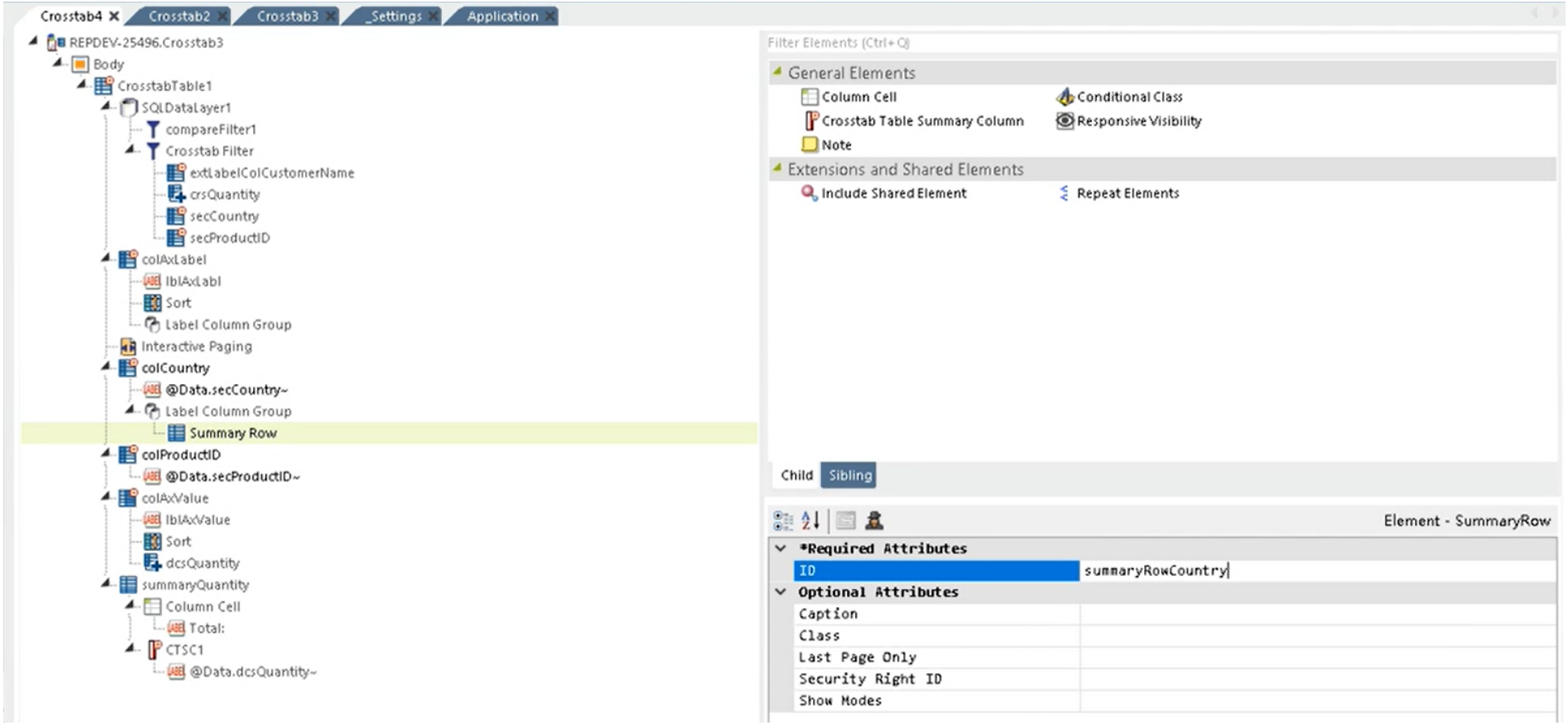
Region	Country	Product ID	Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
Asia-Pacific	Australia	1	798										
		6		3,306									
	China	6		9,738									
		23			5,034								
	Japan	6		6,232									
	Malaysia	1	196										
		2				656							
Europe, Middle East, Africa	Belgium	1	854										
		13					264						
		20						2,044					
		23			1,572								
	Poland	6		5,517									
		8							1,200				
	Russia	1	343										
		13					1,055						
	Saudi Arabia	6		9,738									
	Spain	1	427										
		25								3,355			
		26									1,200		
		28										3,355	
Total:			3283	65634	6606	2435	1398	11206	26855	7179	1200	43283	615

Crosstab Summary

After you apply a Crosstab Filter, you can add Header Rows and Summary Rows to total the Data Columns you have grouped together.

Summary Rows

1. Begin by adding a **Summary Row** element to your Secondary Crosstab Label Columns. In this example, we're adding a Summary Row to the "colCountry" Secondary Crosstab Label Column and giving it the ID "summaryRowCountry":



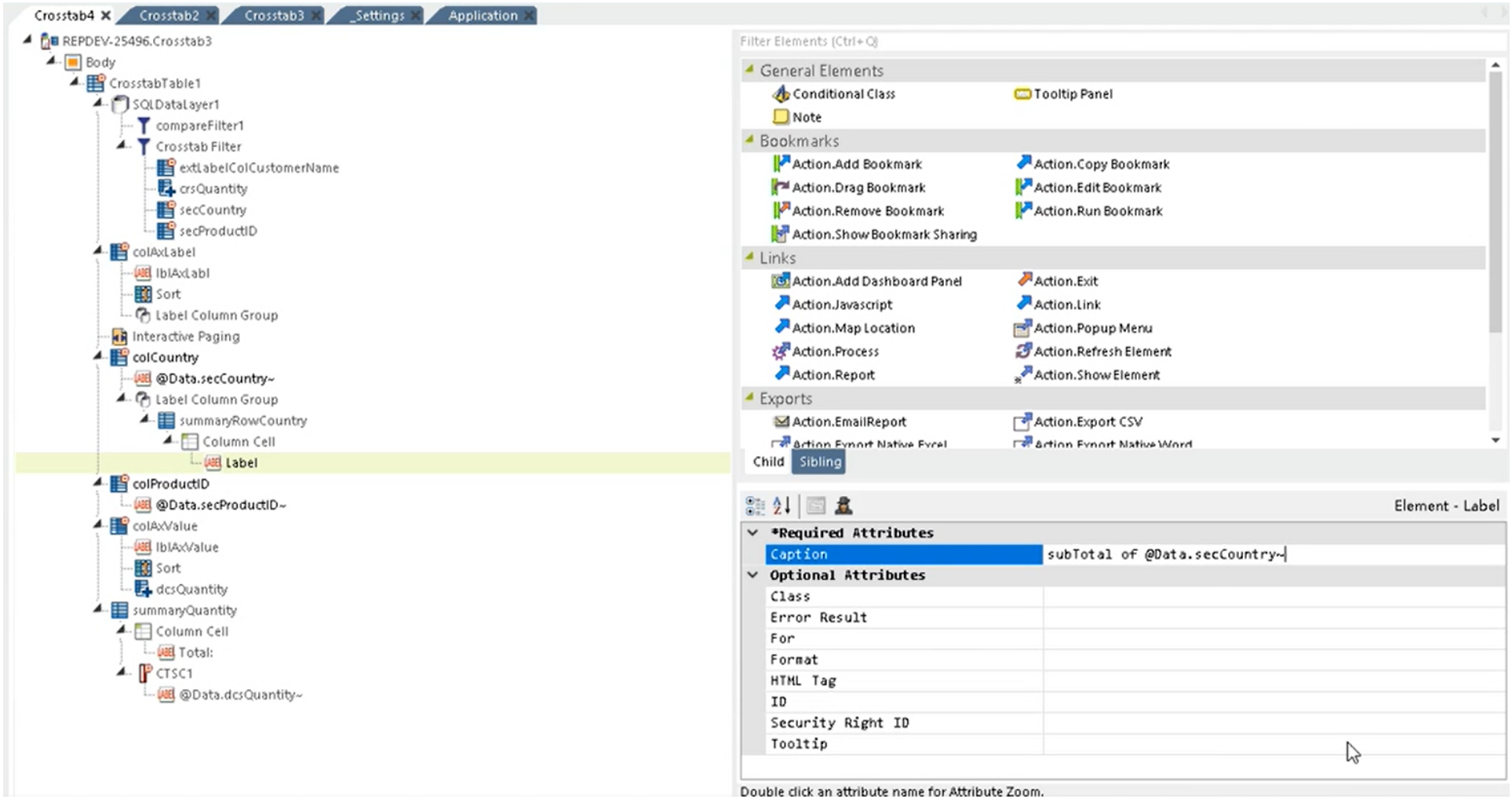
2. **Save** and **refresh** your report. Info displays the Crosstab Table with a Summary Row for the *Country* column:

ProductName by Country on Sum of Quantity

Region	Country	Product ID	Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
Asia-Pacific	Australia	1	798										
		6		3,306									
			798	3,306									
	China	6		9,738									
		23			5,034								
				9,738	5,034								
	Japan	6		6,232									
				6,232									
	Malaysia	1	196										
		2				656							
			196			656							

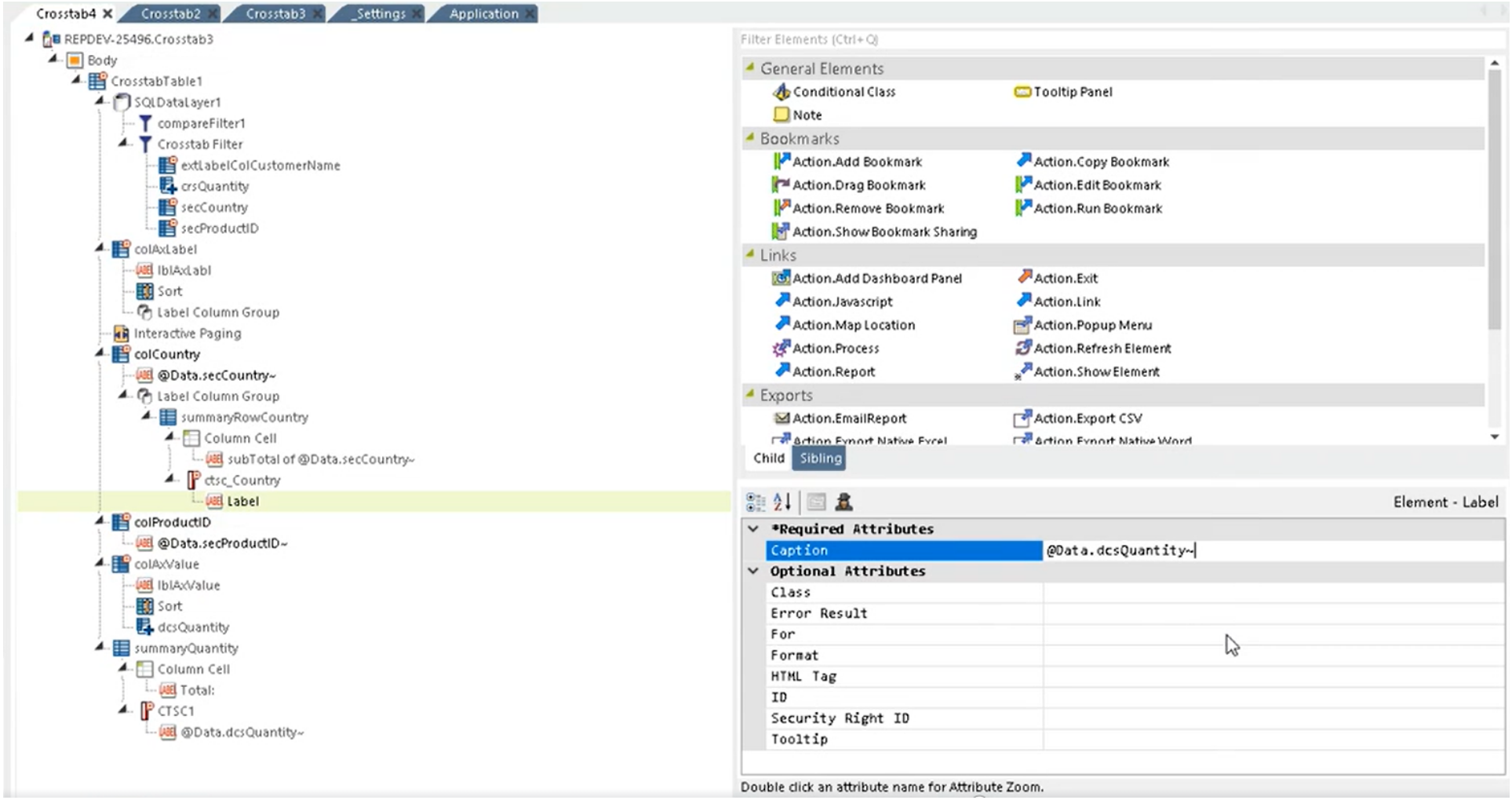
3. You can adjust the Summary Row span by adding a **Column Cell** element. In this example, we're adjusting the Column Span attribute to 2.

4. Then, add a **Label** element to the Column Cell element and add a Caption based on the Data Column you wish to tally, like below:


Crosstab4
Crosstab2
Crosstab3
_Settings
Application
REPDEV-25496.Crosstab3
Body
CrosstabTable1
SQLDataLayer1
compareFilter1
Crosstab Filter
extLabelColCustomerName
crsQuantity
secCountry
secProductID
colAxLabel
lblAxLabl
Sort
Label Column Group
Interactive Paging
colCountry
@Data.secCountry~
Label Column Group
summaryRowCountry
Column Cell
Label
colProductID
@Data.secProductID~
colAxValue
lblAxValue
Sort
dcsQuantity
summaryQuantity
Column Cell
Total:
CTSC1
@Data.dcsQuantity~
Filter Elements (Ctrl+Q)
General Elements
Conditional Class
Tooltip Panel
Note
Bookmarks
Action.Add Bookmark
Action.Copy Bookmark
Action.Drag Bookmark
Action.Edit Bookmark
Action.Remove Bookmark
Action.Run Bookmark
Action.Show Bookmark Sharing
Links
Action.Add Dashboard Panel
Action.Exit
Action.Javascript
Action.Link
Action.Map Location
Action.Popup Menu
Action.Process
Action.Refresh Element
Action.Report
Action.Show Element
Exports
Action.EmailReport
Action.Export CSV
Child
Sibling
Element - Label
*Required Attributes
Caption
subTotal of @Data.secCountry~
Optional Attributes
Class
Error Result
For
Format
HTML Tag
ID
Security Right ID
Tooltip
Double click an attribute name for Attribute Zoom.

5. Next, add a **Crosstab Table Summary Column** to your Summary Row element and give it an ID.

6. Below the Crosstab Table Summary Column, add a **Label** element and assign a Caption, based on the Data Column you are summarizing:



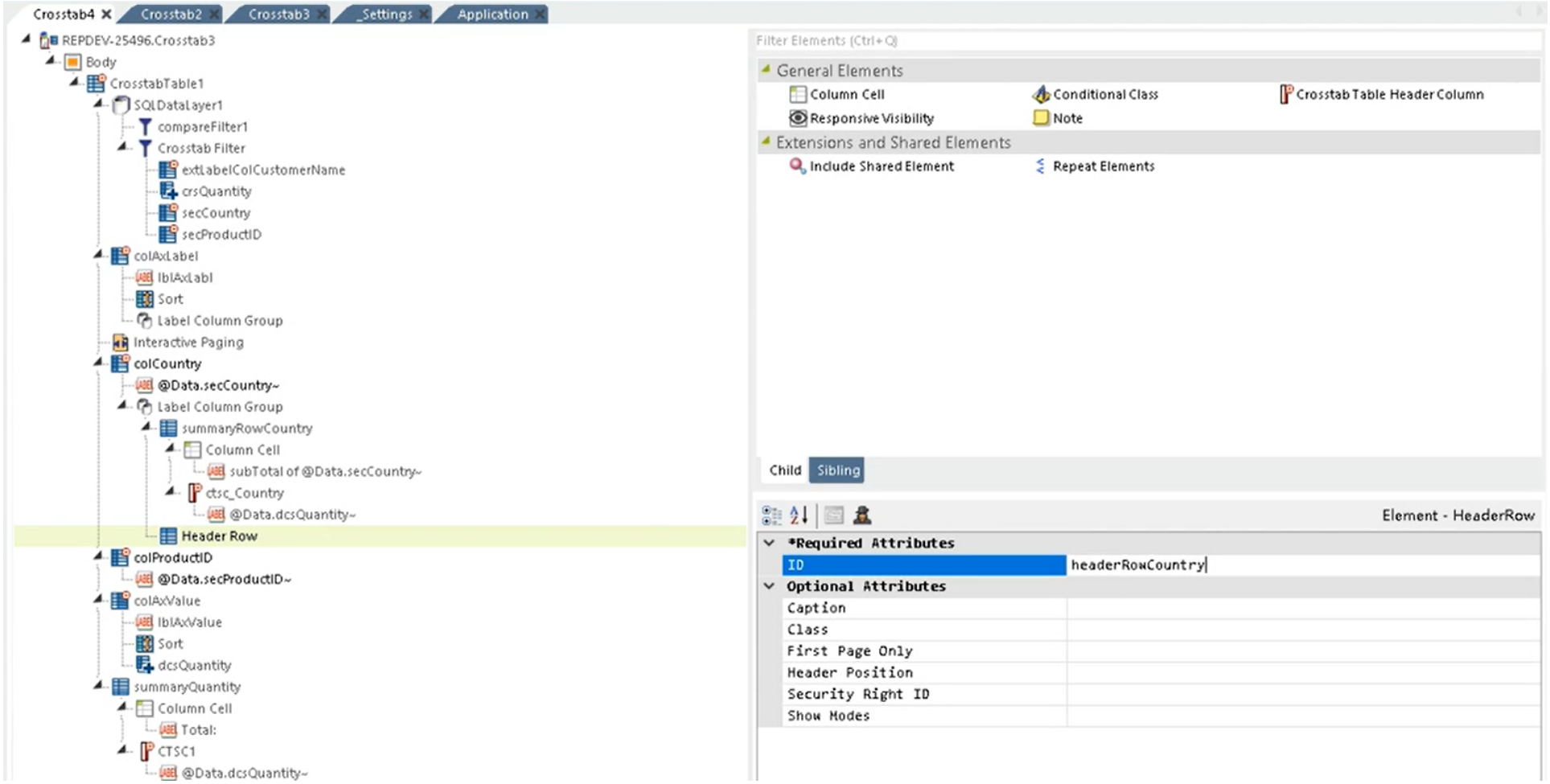
7. **Save** and **refresh** your report. Info displays the Crosstab Table with the summarized column(s):

ProductName by Country on Sum of Quantity

Region	Country	Product ID	Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
Asia-Pacific	Australia	1	798										
		6		3,306									
	subTotal of Australia		798	3306									
	China	6		9,738									
		23			5,034								
	subTotal of China			9738	5034								
	Japan	6		6,232									
	subTotal of Japan			6232									
	Malaysia	1	196										
		2				656							
	subTotal of Malaysia		196			656							
Europe, Middle East, Africa	Belgium	1	854										
		13					264						
		20						2,044					
		23			1,572								
	subTotal of Belgium		854		1572		264	2044					
	Poland	6		5,517									
		8							1,200				
	subTotal of Poland			5517					1200				
	Russia	1	343										
		13					1,055						

Header Rows

1. Add a **Header Row** element beneath your Label Column Group element and give it an ID:



2. **Save** and **refresh** your report. Info displays the Crosstab Table with the new Header Row:

ProductName by Country on Sum of Quantity

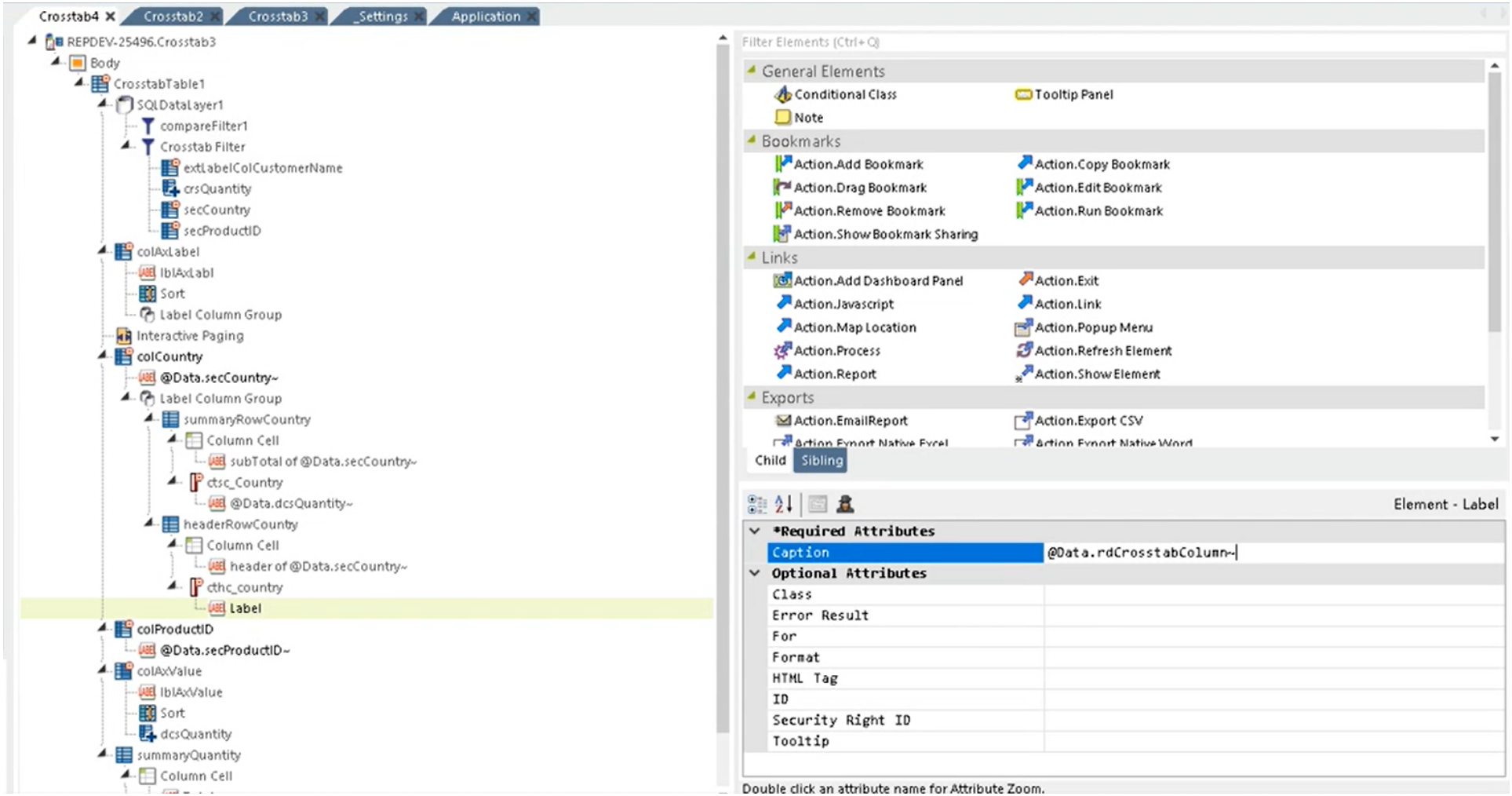
Region	Country	Product ID	Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
			798	3,306									
	Australia	1	798										
		6		3,306									
	subTotal of Australia		798	3306									
				9,738	5,034								
	China	6		9,738									
		23			5,034								
Asia-Pacific	subTotal of China			9738	5034								
				6,232									
	Japan	6		6,232									
	subTotal of Japan			6232									
			196			656							
	Malaysia	1	196										
		2				656							
	subTotal of Malaysia		196			656							
			854		1,572		264	2,044					
	Belgium	1	854										
		13					264						
		20						2,044					
		23			1,572								
	subTotal of Belgium		854		1572		264	2044					

3. You can adjust the span of the Header Row by adding a **Column Cell** element and changing the Column Span attribute. In this example, we're adjusting the Column Span to 2.

4. Add a **Label** element beneath the Column Cell and give it a Caption based on the Data Column.

5. Next, add a **Crosstab Table Header Column** and give it an ID.

6. Then, add a **Label** element beneath the Crosstab Table Header Column and give it a Caption using the special Crosstab Table token " @Data.rdCrosstabColumn~", shown below:



7. Select **Save** and **refresh** your report. Info displays the Crosstab Table with the new Header Columns:

ProductName by Country on Sum of Quantity

Region	Country	Product ID	Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
Asia-Pacific	header of Australia		Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
	Australia	1	798										
		6		3,306									
	subTotal of Australia		798	3306									
	header of China		Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
	China	6		9,738									
		23			5,034								
	subTotal of China			9738	5034								
	header of Japan		Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
	Japan	6		6,232									
	subTotal of Japan			6232									
	header of Malaysia		Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
	Malaysia	1	196										
		2				656							
	subTotal of Malaysia		196			656							
	header of Belgium		Espresso Roast	Ethiopia Sidamo	French Roast	Espresso Roast Decaf	Brazil Ipanema Bourbon	Ethiopian Harrar	Breakfast Blend	Chocolate Hazelnut	French Vanilla	Blue Mountain	Colombia El Tambo
	Belgium	1	854										
		13					264						
		20						2,044					
		23			1,572								
	subTotal of Belgium		854		1572		264	2044					

You can repeat the previous steps for the Crosstab Filter Label Column (in these examples, *Region*) and additional Secondary Crosstab Label Column(s) (in these examples, *Product Id*).

Crosstab Extras

There are five child elements that can be used beneath a Crosstab Filter, as follows:

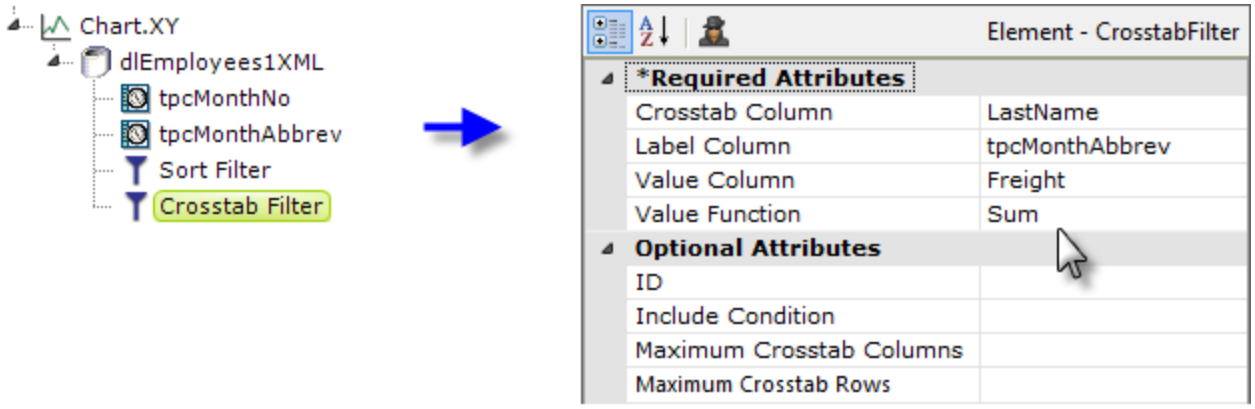
Token	Description
Crosstab Row Sum-	Use this element to generate aggregated values for the table's columns. This basically adds an additional column to the datalayer. You can view the aggregated values by adding a Crosstab Table Label Column

Token	Description
mary Column	and specifying `@Data.x~`, where x is the ID of this element
Extra Crosstab Calculated Column	By default, the Crosstab Filter produces a single value for each column and row combination. This element allows a formula to be used to create additional values. These values can be displayed in a Crosstab Table Value Column element. In the Label element use the special token: `@Data.rdCrosstabValue-myID~` where "myID" is the ID of this element. The formula used may reference special crosstab tokens. For example, if the Crosstab Value Column is "Cost" and an Extra Crosstab Value Column is "Price", we can calculate "Profit" with: `@Data.rdCrosstabValue-Price~ - @Data.rdCrosstabValue~`In this case "Price" is the ID value of an Extra Crosstab Value Column element. `@Data.rdCrosstabValue~` represents the "Cost" because it's the main value created with the Crosstab Filter.
Extra Crosstab Label Column	Crosstab Tables always have a label column that determines how values are grouped together. For some Crosstab Tables, you may want to display additional columns so that they can be included in a Crosstab Table element. For example, you may use a field called "EmployeeID" as the main label column to create the Crosstab. But, if you want to also show the employees' names, you should add an Extra Crosstab Label Column for the first and last name columns. These extra label columns are for display only; they are not used when determining the Crosstab Table rows. Reference the value with `@Data.myID~`, where myID is the value of this element's ID attribute.
Extra Crosstab Value Column	By default, the Crosstab Filter produces a single value for each column and row combination. This element provides for additional values, which can be displayed in a Crosstab Table Value Columns element. In the Label element use the special token: `@Data.rdCrosstabValue-myID~`where "myID" is the ID of this element.

Token	Description
Secondary Crosstab Label Column	This element acts like the Extra Crosstab Label Column, but it allows the columns to be grouped together. Secondary Label Columns are for display only; they are not used when determining the Crosstab Table rows.

Applying the Crosstab Filter to a Chart

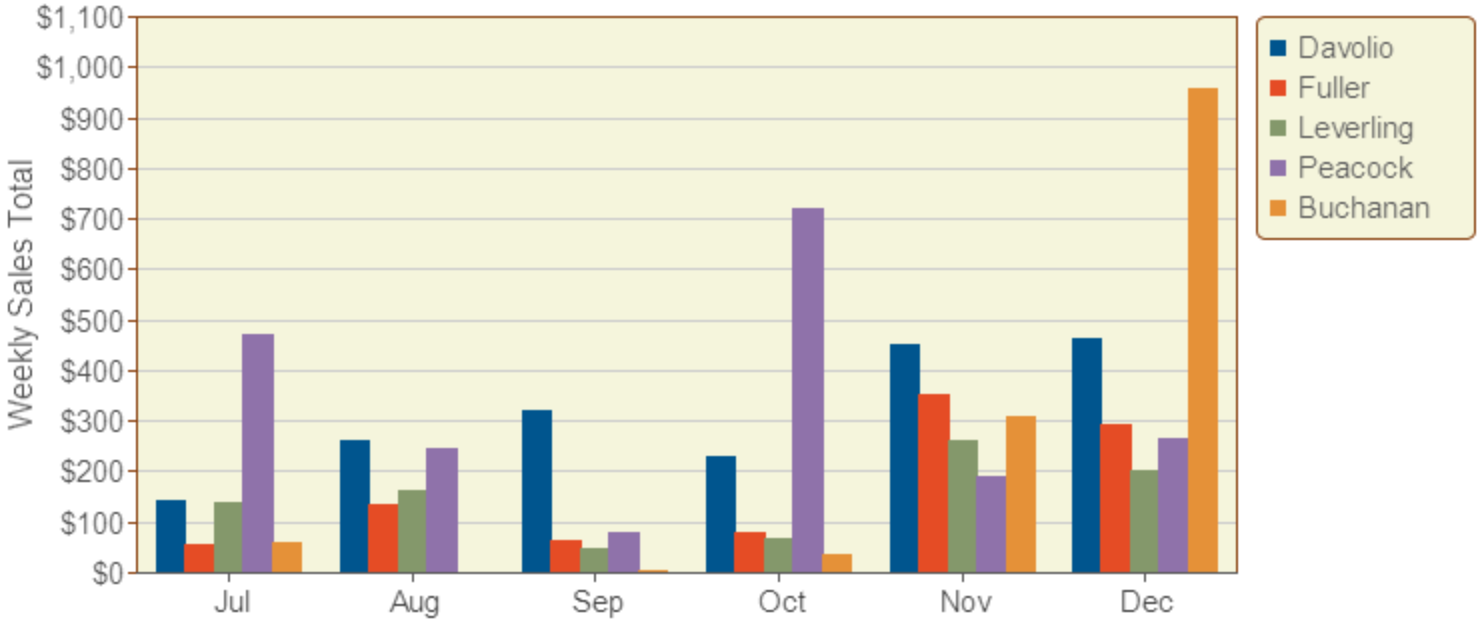
The following example illustrates how the **Crosstab Filter** can be used with a chart:



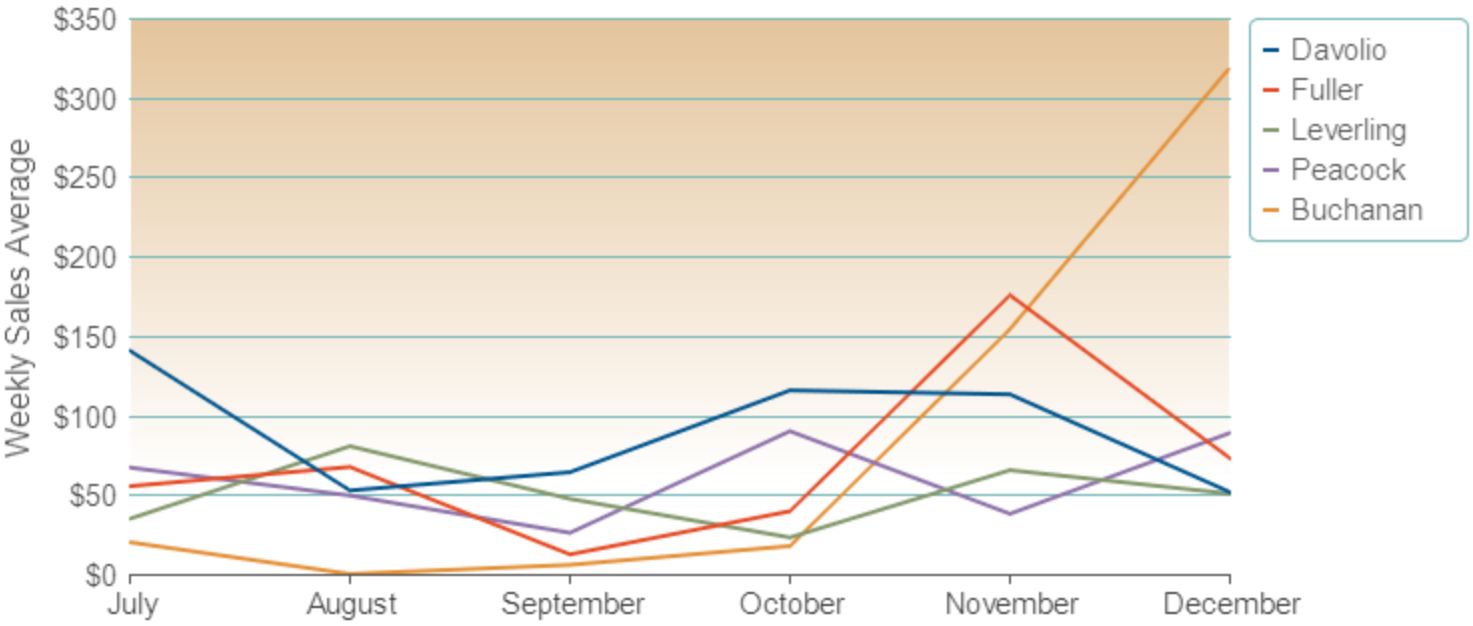
1. In a departure from general practice, when you use a static chart with a Crosstab Filter, you *do not* need to specify the chart's normally-required **Data Column Y-axis** attribute value. Instead, leave it blank.
2. As shown above, an **Crosstab Filter** element is added as a child beneath a datalayer element for a Bar Chart.
3. Its **Crosstab Column** attribute value specifies the datalayer column used to create the chart's bars: each distinct value in this column will create a new bar in the chart.
4. The **Label Column** attribute specifies the column that provides label text along the X-axis, beneath the bars. In this example, we've applied a **Time Period Column** element to the OrderDate data to produce abbreviations for the names of months.
5. The **Value Column** attribute specifies the column that provides the chart's Y-axis values.
6. The **Value Function** attribute specifies how Y-axis values are calculated; in this example we're summing the data for each employee.

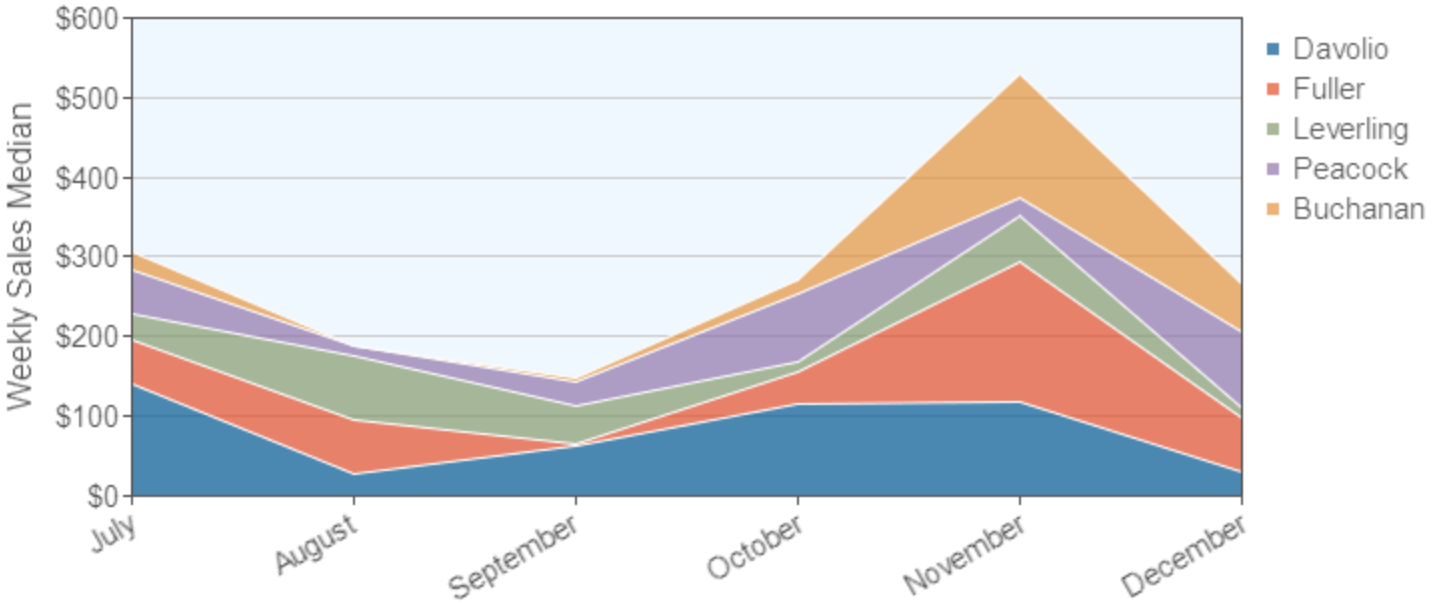
7. In order to create the right monthly order in the X-axis, we've also applied a second Time Period Column element to the OrderDate data to produce just the month number and then we've sorted the datalayer on those month numbers using a **Sort Filter**, *before* applying the Crosstab Filter.

This relatively simple element configuration produces a nice bar chart:



Adding a **Legend** element to the chart products a list of the values from the Crosstab Filter element's **Crosstab Column** attribute. Here are two different chart styles, Line and Area, created using almost the same technique as the chart above:


$350
$300
$250
$200
$150
$100
$50
$0
Weekly Sales Average
July
August
September
October
November
December
Davolio
Fuller
Leverling
Peacock
Buchanan


$600
$500
$400
$300
$200
$100
$0
Weekly Sales Median
July
August
September
October
November
December
Davolio
Fuller
Leverling
Peacock
Buchanan

The difference in these charts is that Crosstab Filter element's **Value Function** attribute has been set to *Average* and *Median*, respectively, rather than to *Sum*, and there are some cosmetic configuration differences in their X-axis labels and legends.

While the element rules in Studio will allow you to add child elements beneath the Crosstab Filter, such as the **Extra Crosstab Value Column** element, they will be *ignored* when the filter is being used with charts.

Using Dynamic Crosstab Filters

Like all filters, the Crosstab Filter element has an **Include Condition** attribute. If the value of this attribute is left blank or contains a formula that evaluates to *True*, the filter is applied to the datalayer. If the value evaluates to *False*, the filter is ignored and does not affect the datalayer. This powerful feature allows developers to dynamically determine if the datalayer will be altered or not.

The DeDuplicate Filter

The **DeDuplicate Filter** element is used with datalayer elements to filter out data rows. It's analogous to using a *DISTINCT* clause in a SQL query but, unlike a query, it's applied to the data *after* it's been retrieved into the datalayer.

The following topics discuss the DeDuplicate Filter element:

- Applying the DeDuplicate Filter
- Using Dynamic DeDuplicate Filters

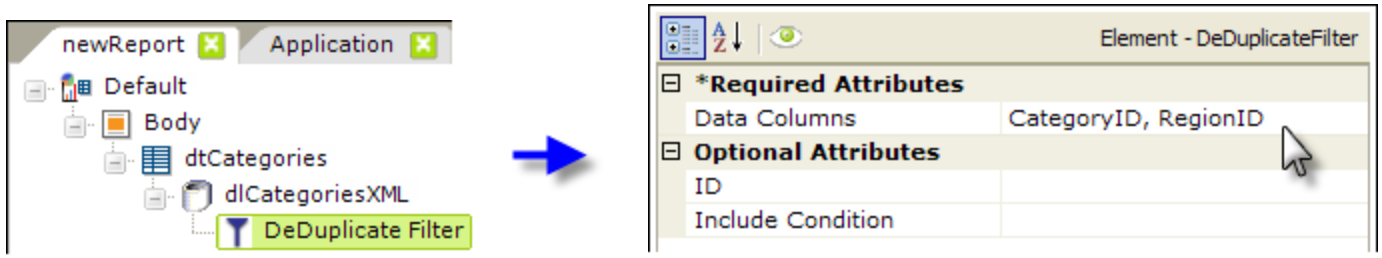
About the DeDuplicate Filter

The DeDuplicate Filter element is available for use with all datalayer elements and is particularly useful in conjunction with those lacking query capabilities, such as DataLayer.XML and DataLayer.Web Service.

In operation, the element basically examines the values in one or more specified data columns and removes all rows from the datalayer where duplicate values are found in those columns. The *first* row encountered in a set of rows with duplicate values is the one that's retained.

Applying the DeDuplicate Filter

The following example illustrates how the **DeDuplicate Filter** is used:



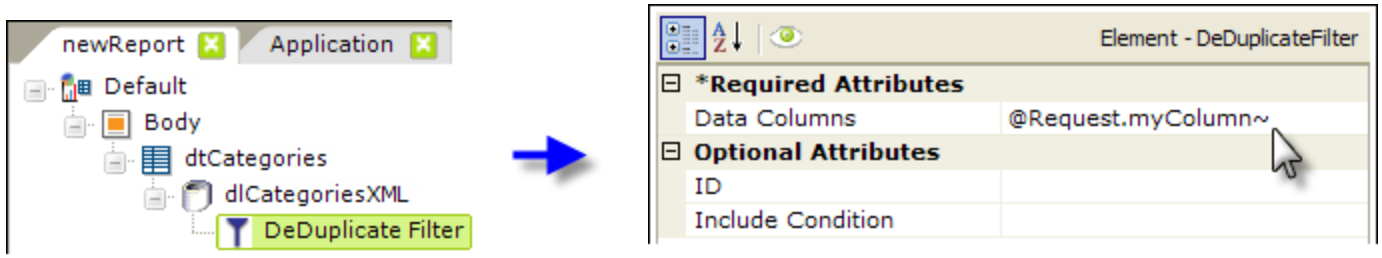
1. As shown above, a **DeDuplicate Filter** element is added as a child to a datalayer element.
2. Its **Data Columns** attribute value is set to the names of one or more datalayer columns, usinga comma-separated list, whose values will be examined.

In the example above, the sequence of events is that the datalayer reads in all the data from the XML data file, then the DeDuplicate Filter **removes** all rows where duplicate values are found in the specified columns. All that remains in the datalayer for use in the report are rows with unique values in the specified columns.

Using Dynamic DeDuplicate Filters

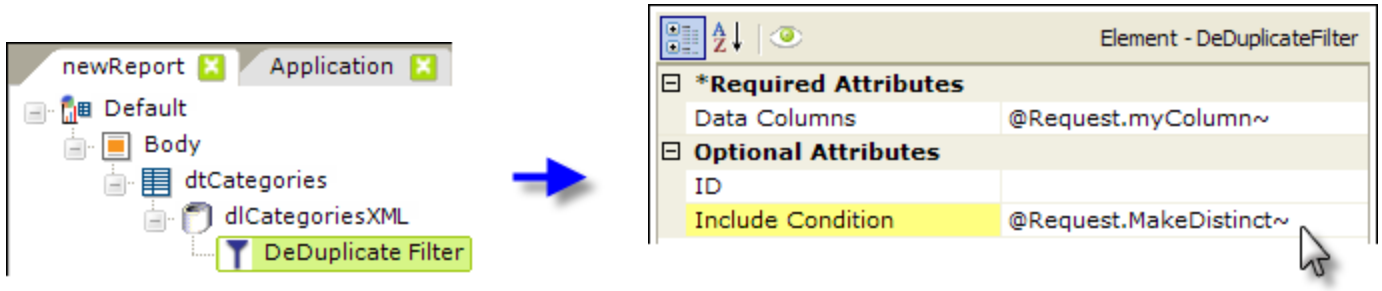
This topic describes how to use dynamic DeDuplicate Filters.

1. Developers can make DeDuplicate Filter elements **dynamic** by using tokens:



As shown above, the Data Columns attribute value can accept tokens such as @Request, so that column names can bedynamic at runtime. Sometimes it is necessary to ensure that tokens have default values when running a report. When using @Request tokens, the **Default Request Params** element can be used to create default values for each **@Request** token.

2. The DeDuplicate Filter element also has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows developers to dynamically determine if the datalayer will be filtered or not.

The File Column (BLOBs)

Many database servers allow you to store Binary Large Objects **(BLOBs**) or Character Large Objects (**CLOBs**) in a table column. Logi Info developers can manipulate this kind of data using the **File Column** element.

The following topics discuss the File Column element:

- Preparing Data in a Datalayer
- Working with Retrieved Data
- Example: Retrieving Dashboard Configuration Data

About BLOBs and CLOBs

The ability of some database servers to store large amounts of data in individual table columns provides a interesting challenge for data storage and retrieval. A common BLOB implementation example is an HR table that stores employee data and includes an actual *photo* of each employee in one of its columns. Other examples of BLOBs include audio and multimedia objects, and even executable code.

Other implementations may store large amounts of *text*, called CLOBs, in the same manner. For example, the HTML for entire web pages could be stored in a single table column as part of a content management system. Other examples of CLOBs include XML data, documents, and logs.

BLOB and CLOB data may be stored as a collection of bytes right in the database's standard storage area or they may reside outside it, in auxiliary files, referenced by pointers in the table column record. The size of a BLOB or CLOB can be quite large; for example, ODBC and JDBC specs allow up to 4GB of data.

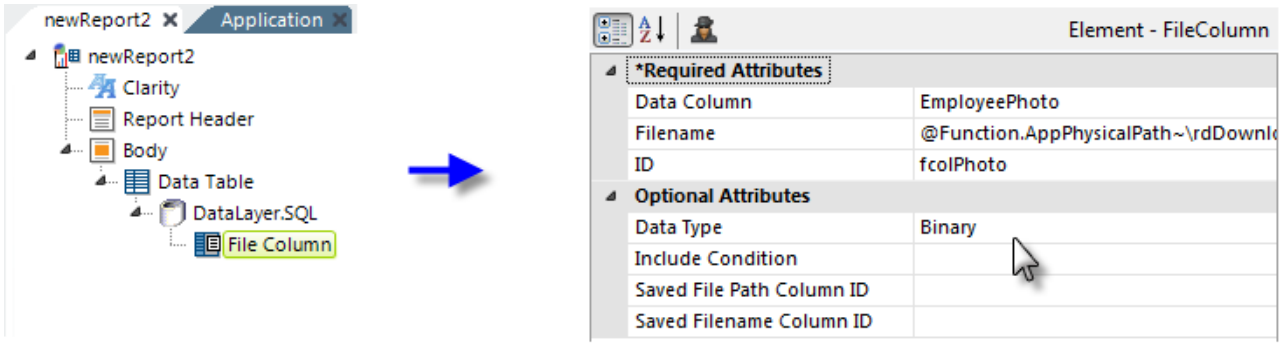
Let's see how this kind of data can be retrieved for use in a Logi application.

Preparing Data in a Datalayer

Data from a database is usually retrieved in Logi applications using a datalayer element. The **File Column** child element accesses data in a BLOB or CLOB column by manipulating the columns in a datalayer. This is similar to the **Calculated Column** element, which can be used to add additional columns to a datalayer. The File Column element performs these operations:

- *Reads* the BLOB or CLOB data from the datalayer and *saves* it to a temporary file, then deletes it from the datalayer.
- *Adds* two columns to the datalayer and inserts values into them for the temporary file's path and filename.

The following example illustrates how the File Column element is used:



1. As shown above, a **File Column** element is added beneath a datalayer; multiple File Column elements can be used as necessary.
2. Its required **Data Column** attribute specifies the name of the column in the datalayer that contains the BLOB or CLOB data.
3. Its required **Filename** attribute specifies the fully-qualified path and filename for the temporary file that will hold the BLOB or CLOB data. The example above uses one @Function token to get the Logi application's base folder path and another to create a unique identifier for the actual file name. The complete value might look like this:

```
@Function.AppPhysicalPath~\rdDownload\@Function.GUID~.png
```

Use of the `rdDownload`folder as the location for temporary files within your application folder ensures that they'll be "cleaned-up" automatically soon after each user's session ends. Don't forget to provide a *file extension* appropriate to the data type, such as `.png`, `.jpg`, `.doc`, `.xml`, etc.

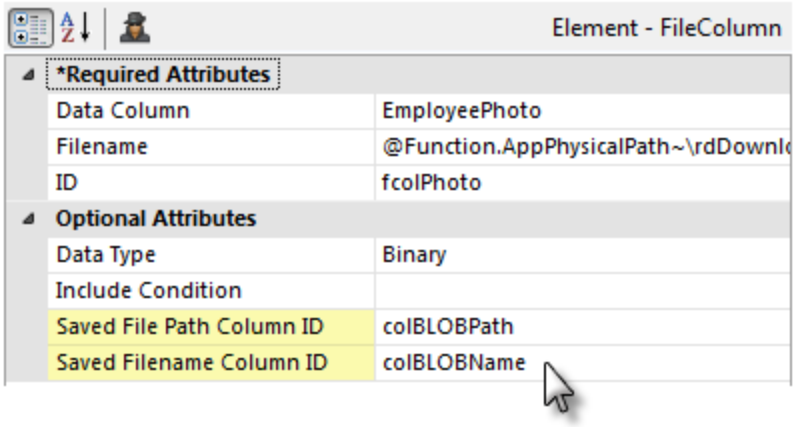
If a file of the same name already exists in the specified location, it will be overwritten without warning.

4. Its required **ID** attribute specifies a unique identifier for the File Column element.
5. Its optional **Data Type** attribute specifies the type, *Text* or *Binary*, of the BLOB or CLOB data. Binary values should be Base64-encoded (this is typical).

The configuration of the File Column element thus far has satisfied the first of the two operations mentioned earlier, *saving* the data to a temporary file and then automatically *removing* it from the datalayer. The final two optional File Column element attributes are discussed in "Working with the Retrieved Data" on the next page.

Working with the Retrieved Data

In order to be able to work with or display the data retrieved from a BLOB or CLOB column, you need to be able to accurately reference the temporary file created to hold it. After saving the data to a temporary file, the File Column element adds two columns to the datalayer, to hold the file path and file name. The element's final two attributes allow you to give IDs to those columns:

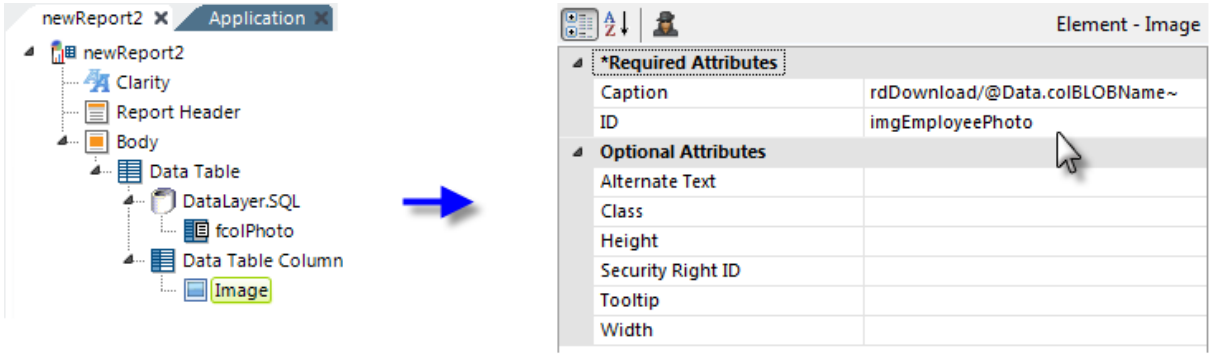


1. **Saved File Path Column ID** - Specifies a namefor the column that will be added to the datalayer to contain the temporary file's path data.
2. **Saved Filename Column ID** - Specifies a namefor the column that will be added to the datalayer to contain the temporary file's file name data.

These column names should be unique; don't use the nameof an existing column in the datalayer. At runtime, these two columns will automatically be *added* to the datalayer and populated with the actual path and filenames that were used to save the BLOB or CLOB data to temporary files.

Using the Data

Now you're ready to actually put the BLOB or CLOB data into your report. The following example displays BLOB data in an **Image** element; the techniques for displaying CLOB data in a Label element are similar.



1. As shown above, a **Data Table Column** element has been added to the definition, and a child **Image** element beneath it.
2. The Image element's **Caption** attribute has been set to the proper folder and the @Data token for the column containing the file name of the temporary file holding the BLOB data:

   ```
   rdDownload/@Data.colBLOBName~
   ```

Remember when using a **Label** element to display CLOB data that, if the data is HTML, it may be useful to set the Label element's **Format** attribute value to *HTML*.

Image and Label elements now include "HTML Attribute Params", enabling you to apply your application to work with other frameworks or libraries easier. With the HTML Attribute Params, you have the option to include "Id", "type", and "value" para-

meters. If an attribute was set in both Element and HTML attribute params, the one set in the HTML attribute params will be ignored.

Example: Retrieving Dashboard Configuration Data

Here's an example scenario where the configuration data for a super-element, the Dashboard, which is usually saved to an XML file, is also saved to a database table and retrieved using the **File Column** element.

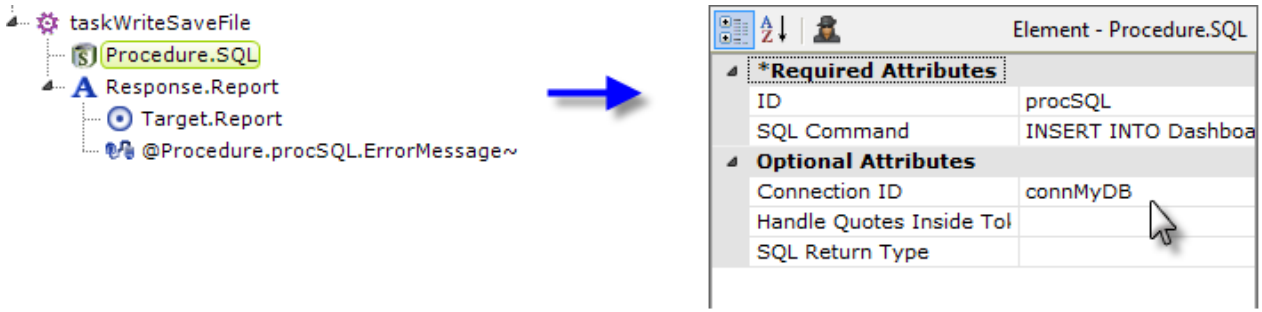
Requirements

- Your database server must be capable of saving a CLOB in a data column. This example uses Microsoft SQL Server 2008 R2, which has an "XML" data type (also available on MS SQL Server 2012).
- The database account used to connect to the server must have permissions for **Bulk** insert/update operations.
- A database table must exist that includes the following schema:

```
UserNameVARCHAR(50)-- a unique identifier for retrieving the record SaveFile XML
```

Writing the Data

We'll use a Process task to write the data to the database. This task can be called by a user action, such as clicking a "Save" button or link, or logging out of the application, or by something more automatic, such as the end of the session.



An example of the task is shown above. Writing the data relies on a SQL command that reads directly from a file:

```
INSERT INTO DashboardData
          SELECT '@Function.UserName~', SaveFile
          FROM (
          SELECT * FROM OPENROWSET (BULK
 '@Function.AppPhysicalPath~\SavedDashboards\@Function.UserName~.xml',
 SINGLE_CLOB)
          AS SaveFile) AS R(SaveFile)
```

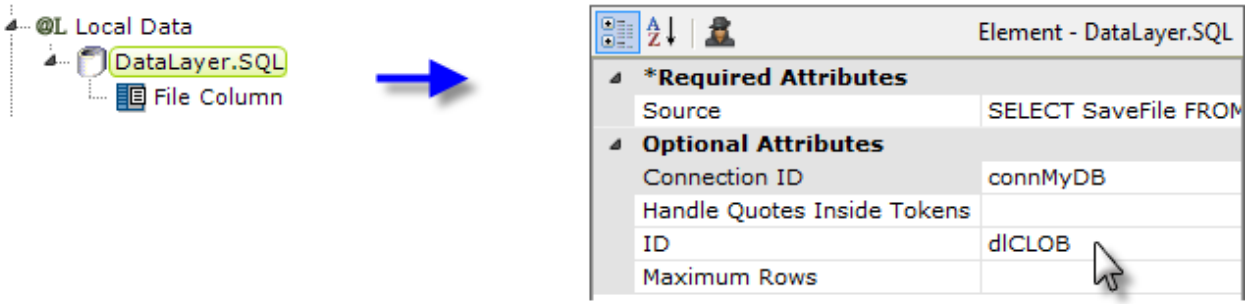
 The second SELECT statement line is wrapped on this page and should be all one line.

This example assumes that the user name is available as a token, either through Logi Security or another security scheme. The user name is then used in the SQL command to provide a value for the unique identifier column, and for the name of the Dashboard save file. This syntax is correct for MS SQL Server 2012 and 2008 R2 but may vary for other database servers.

Naturally, the Dashboard will have to be used at least once so its configuration is saved to its Save File before the data can be written to the database.

Reading the Data

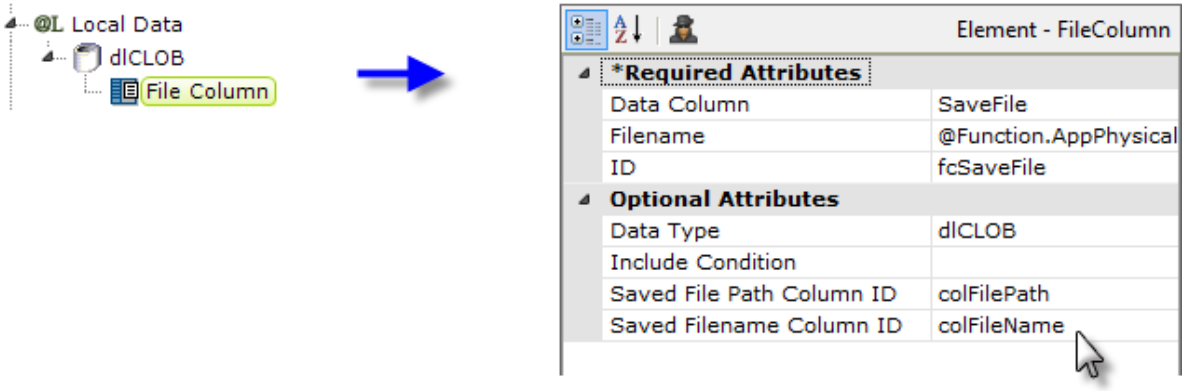
When running the report, the data stored in the database needs to be read once and saved to an XML data file on the web server that the Dashboard element can use.



We'll do this, as shown above, with a **Local Data** element. Its **Condition** attribute can be set in some manner to ensure that it only runs the first time the page is displayed (@Request token, cookie, etc.). Data retrieval will take place using **DataLayer.SQL** using a query similar to:

```
SELECT SaveFile FROM DashboardData WHERE UserName = 'Bob'
```

The use of Local Data allows important data to be available, using an @Local token, anywhere in the rest of the definition.



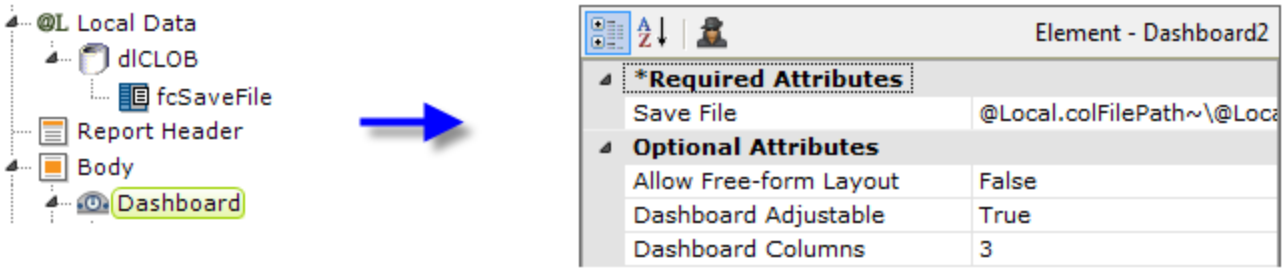
A shown above, a File Column element is used to extract the stored data from the database column and save it as a file on the web server. The complete Filename attribute would be something like:

```
@Function.AppPhysicalPath~\SavedDashboards\@Function.UserName~.xml
```

which you'll recognize as matching the file path and name used in the SQL INSERT command in the previous section. File Column will overwrite any existing file when it saves the XML data.

Using the Data in the Dashboard

Finally, the Dashboard element needs to be configured to use the newly-written data file:



When configuring the Dashboard element's **Save File** attribute, you can use the `@Function.AppPhysicalPath~\...` string we used earlier, or you can use a combination of Local Data tokens created by the File Column element, such as:

```
@Local.colFilePath~\@Local.colFileName~
```

if you aren't tying the Save File to individual users. This approach, storing configuration data in an XML data type column in a database table, is also applicable to other super-elements, such as the Analysis Grid.

The Flattener

The **Flattener** element "flattens" hierarchical data into a simpler, tabular set of rows and columns, making the information usable for a variety of Logi elements.

The following topics discuss the Flattener element:

- Flattener Attributes
- Flattener Usage Examples

About the Flattener

Some data sources return information in hierarchical formats, which may contain multiple levels of parent-child relationships. This structure can create difficulties for further processing. The **Flattener** element processes hierarchical data into a tabular set of rows and columns, so that it can be used with a range of other Logi elements.

The Flattener is available for use with a limited number of datalayers, including:

- DataLayer.JSON File
- DataLayer.Mongo Find
- DataLayer.Mongo Map Reduce
- DataLayer.Mongo Run Command
- DataLayer.REST
- DataLayer.Web Service
- DataLayer.XML File

The Flattener adds an attribute, *rdFlattenedElement*, to each flattened element in the output data to indicate the input element's name.

Flattener Attributes

The attributes of the Flattener element are described below:

Attribute	Description
Bottom Level Element Names	Specifies one or more names of elements in the input data, in a comma-separated list, where the flattening operation should end. Any elements in this list will not have any of their children processed. The default (no value) causes flattening to end at the lowest child level.
Data Row Element Names	Specifies one or more names of elements in the input data, in a comma-separated list, which should generate output data rows. If left blank, an output data row is produced for each child element of the root element.
ID	Specifies a unique element ID.
Ignore Deep Attributes	Specifies whether all attributes of the elements that generate data rows will be added to the output, even attributes from lower-level child elements. When set to *True*, those lower-level attributes are *not* copied. The default (blank) value is *False*.
Ignore Text Elements	Specifies whether text nodes (elements) are moved into attributes of their parent element. When set to *True*, text nodes are *not* moved. The default (blank) value is *False*.
Prepend Element Names	Specifies whether the element name should be prepended to the attribute when it's flattened. For example, if set to *True* (the default value), `<parent child="x" />` will be output as `<datalayerid parent_child="x" />`. If set to *False*, it would be output as `<datalayerid child="x" />`.

Attribute	Description
Top Level XPath	Specifies an XPath query to be used to select the portion of the dataset to be flattened. The default value is *, which selects the entire dataset.

The effects of some of these attributes on data are shown in "Flattener Usage Examples" on the next page.

Flattener Usage Examples

Here are some simple examples of the Flattener's effects.

```
<Library commonName="Library of Congress">
                <Book ID="B1" />
 <Book ID="B2">
                                <Category name="Fiction" />
                                <Category name="Mystery" />
                                <PublishDate year="1900" />
                                <Author>John
Doe</Author>
                </Book>
                <Book ID="3" >
                                <Category name="Nonfiction" />
                </Book>
                <Video ID="V1" />
</Library>
```

The data shown above is the hierarchical input data for the following examples.

```
<dataLayer rdFlattenedElement="Book" ID="B1" />
<dataLayer rdFlattenedElement="Book" ID="B2" Author="John Doe"/>
<dataLayer rdFlattenedElement="Book" ID="B3" />
```

1. In this simple example, *no* Flattener element attributes are set and it transposes the data into the tabular layout shown above.
Text node elements are treated the same way as attributes.

```
<dataLayer rdFlattenedElement="Book" commonName="Library of Congress" ID="B1" />
<dataLayer rdFlattenedElement="Book" commonName="Library of Congress" ID="B2" name="Mystery" year-
r="1900" Author="John Doe" />
<dataLayer rdFlattenedElement="Book" commonName="Library of Congress" ID="B3" name="Nonfiction" />
```

2. Using the same input data, the example above shows the flattened data created by specifying these Flattener element attributes:

Data Row Element Names = *Book*
Top Level XPath = *//Library*

```
<dataLayer rdFlattenedElement="Book" Library_commonName="Library of Congress" Book_ID="B1" />
<dataLayer rdFlattenedElement="Book" Library_commonName="Library of Congress" Book_ID="B2" Category_
name="Mystery" PublishDate_year="1900" Book_Author="John Doe" />
<dataLayer rdFlattenedElement="Book" Library_commonName="Library of Congress" Book_ID="B3" Category_
name="Nonfiction" />
<dataLayer rdFlattenedElement="Video" Library_commonName="Library of Congress" Video_ID="V1" />
```

3. Using the same input data, the example above shows the flattened data created by specifying these Flattener element attributes:
Data Row Element Names = *Book,Video*
Prepend Element Names = *True*
Top Level XPath = *//Library*

```
<dataLayer rdFlattenedElement="Book" ID="B2" name="Fiction" Author="dave was here" />
<dataLayer rdFlattenedElement="Book" ID="B2" name="Mystery" Author="dave was here" />
<dataLayer rdFlattenedElement="Book" ID="B2" year="1900" Author="dave was here" />
<dataLayer rdFlattenedElement="Book" ID="B3" name="Nonfiction" />
```

4. Using the same input data, the example above shows the flattened data created by specifying these Flattener element attributes:
Data Row Element Names = *Category,PublishDate*
Prepend Element Names = *True*
Top Level XPath = *//Library*

The Forecasting Elements

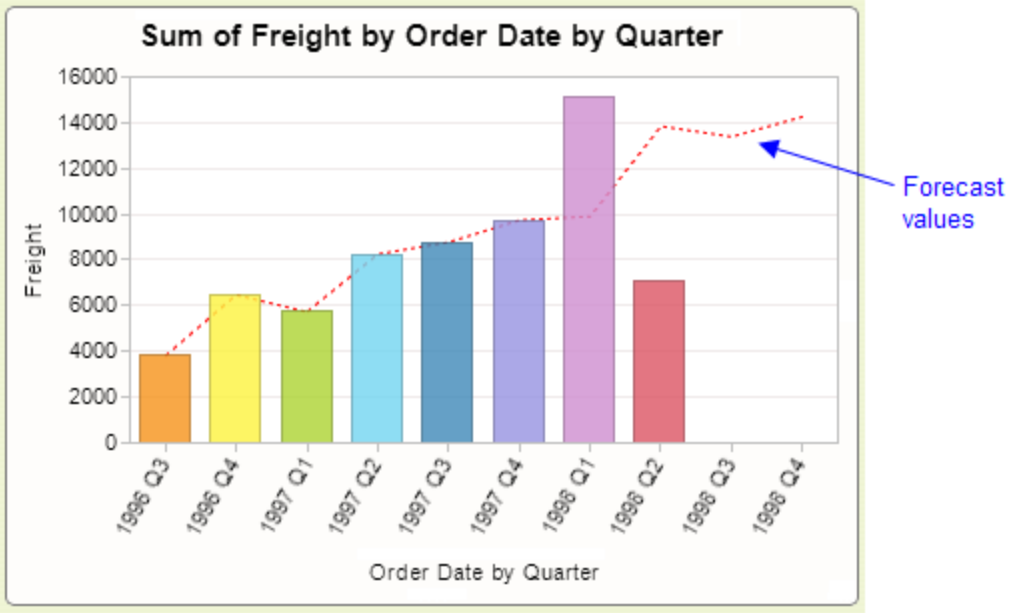
Forecasting elements use a variety of techniques to produce projected values by analyzing existing values. The future values they "predict" are, in most cases, added as rows or columns to a datalayer so the data can be displayed along with the existing data.

The following topics discuss Forecasting elements:

- Using Forecast.Current Time Period
- Using Forecast.Regression
- Using Forecast.Time Period Decomp

About Forecasting

Data forecasting is the process of generating values based on events that have *not yet* occurred. "Prediction" is a similar but more general term. Forecasting refers to formal, statistical methods that use time series, cross-sectional, or longitudinal data. Logi Forecast elements are applied as the children of datalayer elements (forecasting features available in some of our "super elements", such as the Analysis Chart, use separate elements and attributes that enable them). The forecasted data values are generated by analyzing the existing values in the datalayer, and the resulting "future" values are added to the datalayer so they can be displayed along with the existing values.



Typically, forecasts are displayed most effectively on charts. In the example shown above, quarterly freight costs have been forecast using a Time Series Decomposition method and are displayed as a **dotted line** on a bar chart. Data that best lends itself to forecasting can be:

- **Time Series**, consisting of data in a natural time-related order with a strong interval, where the "independent data" (X-axis) is of DateTime type and the "dependent data"(Y-axis) is a number. For example, "Product Sales by Month".
- **Causal**, consisting of dependent numeric values ordered by a numeric independent variable. Example: a count of unique website visitors and a count of the number of ads they clicked on.

There is no single correct forecasting method to use in any given situation. There are a large number of methods, including the following just for time series forecasts:

- Moving average
- Weighted moving average
- Exponential smoothing
- Autoregressive moving average (ARMA)
- Autoregressive integrated moving average (ARIMA) e.g. Box-Jenkins
- Extrapolation
- Linear prediction
- Trend estimation
- Growth curve
- Regression analysis

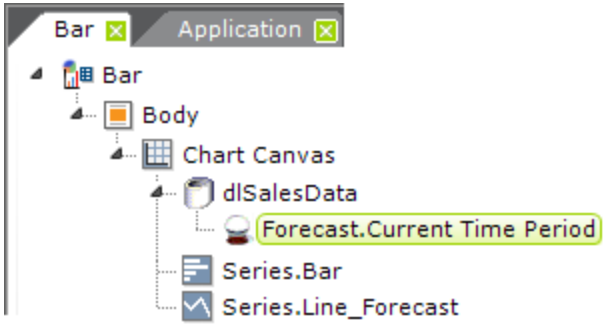
Some of these methods are easy to understand and some are not. The Logi Forecast elements use a combination of some of these methods.

Forecasting Methodologies

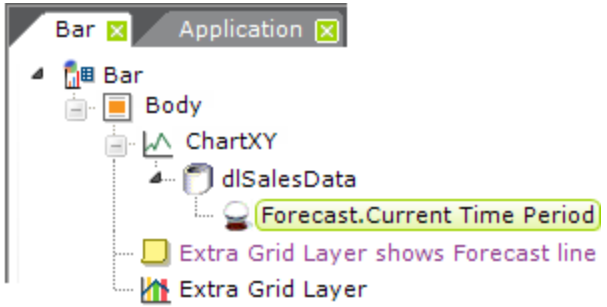
Our reference topic, "Forecast Methodologies" on page 473, provides more detailed information about our forecasting implementations.

Using Forecast.Current Time Period

The **Forecast.Current Time Period** element produces its forecast by analyzing a value from the last row in the datalayer and "prorating" it into the future, through completion of a specific time period. Its calculations *do not* take working hours per day, holidays, or weekends into consideration.



with Chart Canvas chart



with Classic static chart

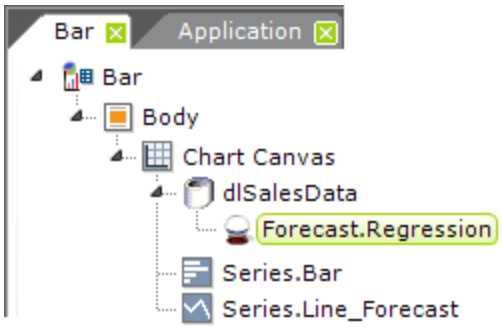
The Forecast.Current Time Period element is a child of any datalayer, as shown above, and its attributes include:

Attribute	Description
Data Column	(Required) Specifies the name of a column in the datalayer that contains the values which will be "prorated" across the time period.
DateTime Data Column	(Required) Specifies the name of a column in the datalayer that contains date values (and, optionally, a time) that are the starting point in time from which to prorate the Data Column value.
ID	(Required) A unique element ID.

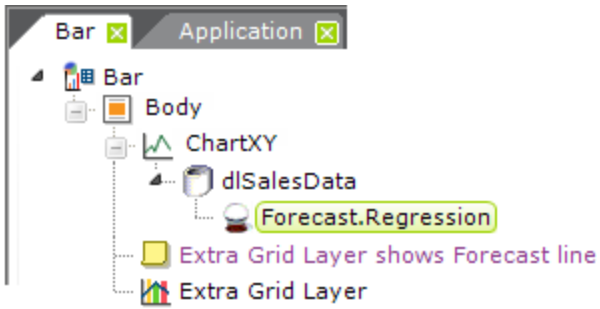
Attribute	Description
Time Period	(Required) Specifies the time period through which values should be forecast. Options include: Hour, Day, Week, Month, Quarter, Year.
Forecast Difference Column ID	Specifies the name of a new column that will be created in the datalayer to hold the value of the difference between the starting data value and each prorated forecast value.
Forecast Indicator Column ID	Specifies the name of a new column to be added to the datalayer, with value set to "True", for each row used in the forecast analysis. If left blank, the column will automatically be named "{ElementId}" + "ForecastIndicator"
Forecast Value Column ID	Specifies the name of a new column that will be created in the datalayer to hold each forecast value.

Using Forecast.Regression

The **Forecast.Regression** element uses one of several regression analysis functions. Regression analysis is recommended when the focus is on a relationship between a dependent value and one or more independent values. It's useful, for example, when values in the X-axis are "uneven" (are separated by varying differences).



with Chart Canvas chart



with Classic static chart

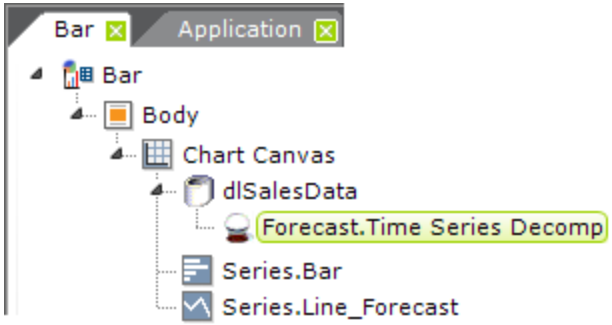
The Forecast.Regression element is a child of any datalayer, as shown above, and its attributes include:

Attribute	Description
Dependent Data Column	(Required) Specifies the name of a column returned into the datalayer. It represents data values for the Y-axis.
ID	(Required) A unique element ID.
Independent Data Column	(Required) Specifies the name of a column returned into the datalayer. It represents data values for the X-axis.

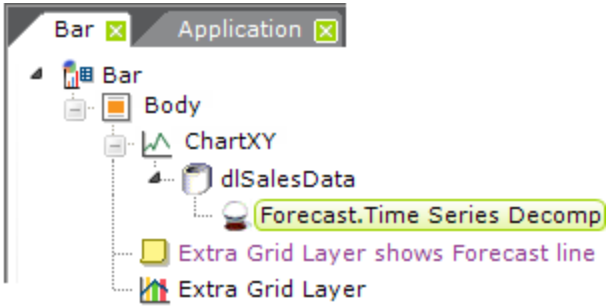
Attribute	Description
Regression Type	(Required) Specifies the type of regression analysis to be applied. • Select *Linear* to calculate predictive values based on a trend line. • Select *Autoregressive* when attempting to predict an output of a system based on previous outputs. The estimation technique used is based on "Burg's" method. Set the optional Autoregressive Order attribute (below) to define how many previous values to use. • Select a non-linear regression type (*Exponential*, *Logarithmic*, *Polynomial*, or *Power*) to display the relationship between dependent and independent variables as a curvilinear function, which may provide more accuracy than a linear regression.
Autoregressive Order	Used only when Regression Type is set to *Autoregressive*. Specify a value between *1 - 8* (default =*3*) to define how many previous values have to be observed for analysis.
Forecast Indicator Column ID	Specifies the name of a new column to be added to the datalayer, with value set to *True*, for each row used in the forecast analysis. If left blank, the column will automatically be named "{ElementId}" + "ForecastIndicator".
Forecast Length	Specifies the actual number of rows to analyze. A value entered that is greater than the total number of rows will be reduced to the total number of rows minus one. If left blank, 20% of the total number of rows will be used.
Forecast Value Column ID	Specifies the name of a new column that will be created in the datalayer to hold each forecast value. If this value is left blank, the forecast value will be *added* to the value of the Dependent Data Column.

Using Forecast.Time Series Decomp

The **Forecast.Time Series Decomp** element uses a technique that's optimized for analyzing data across daily, weekly, or other time periods. The data to be analyzed should be grouped and detail rows discarded, so there is one data row for each time period. If possible, time intervals should be consistent, such as 12 months in a year, not 5 months spread across one year.



with Chart Canvas chart



with Classic static chart

The Forecast.Time Series Decomp element is a child of any datalayer, as shown above, and its attributes include:

Attribute	Description
Dependent Data Column	(Required) Specifies the name of a column returned into the datalayer. It represents data values for the Y-axis.
ID	(Required) A unique element ID.
Independent Data Column	(Required) Specifies the name of a column returned into the datalayer. It represents data values for the X-axis.

Attribute	Description
Cycle Time Span	Specifies the periodic variation or cycle to be used in the analysis of the data. Options include: • Hour: 1 day (a 24-hour cycle) • Daily: 1 week (a 7-day cycle) • Week: 1 month (a 4-week cycle) • Month: 1 year (a 12-month cycle) • Quarter: 1 year (a 4-quarter cycle) • Year: 1 year (a 1-year cycle)
Forecast Indicator Column ID	Specifies the name of a new column to be added to the datalayer, with value set to *True*, for each row used in the forecast analysis. If left blank, the column will automatically be named "{ElementId}" + "ForecastIndicator".
Forecast Length	Specifies the actual number of rows to analyze. A value entered that is greater than the total number of rows will be reduced to the total number of rows minus one. If left blank, 20% of the total number of rows will be used.
Forecast Value Column ID	Specifies the name of a new column that will be created in the datalayer to hold each forecast value. If this value is left blank, the forecast value will be *added* to the value of the Dependent Data Column.

The Formatted Column

The **Formatted Column** element is used with datalayer elements to *add a new column*, containing formatted data, to the datalayer and provides a significant performance improvement. This topic discusses use of this element;.

- About the Formatted Column Element
- Usage Examples

About the Formatted Column Element

The **Formatted Column** element is similar to other elements designed to extend the data available in a datalayer. Like the Calculated Column element, the Formatted Column is added as a child element beneath a datalayer and *adds anew column* to the datalayer at runtime. This new column contains formatted data which is accessible using an @Data token.

Formatting data in the datalayer is particularly useful when working with charts whose data attributes may not allow application of scripting functions or formats. In addition, Tooltips can also benefit from being able to use formatted data.

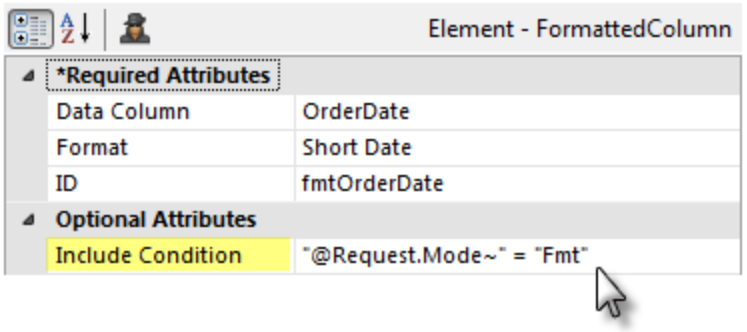

Data in a Formatted Column may not be of the same data *type* as the column from which it is derived. For example, when a numeric column, with a value of 1234, is used to create a Formatted Column that applies the *Currency* format, the result will be string-type data: "$1,234.00". Thus, in this case, the Formatted Column is useful for chart axis labels, but may not work for chart data, which requires the original numeric value.

The pre-configured format options available in the Formatted Column element are identical to the **Format** attribute value options in elements such as the Label. In addition, custom formatting can be entered manually.

For more information about formatting options and custom formats, see *Format Data*.

Dynamic Formatting

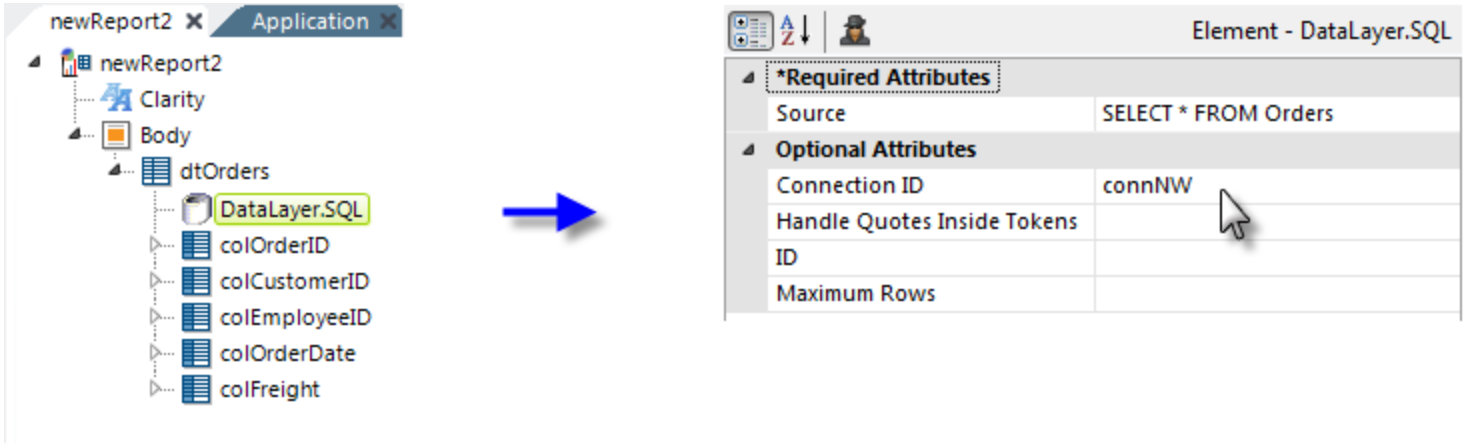
The Formatted Column element has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows you to dynamically determine if a new column of formatted data will be added to the datalayer or not.

Usage Examples

Let's look at some examples of common Formatted Column use-cases.



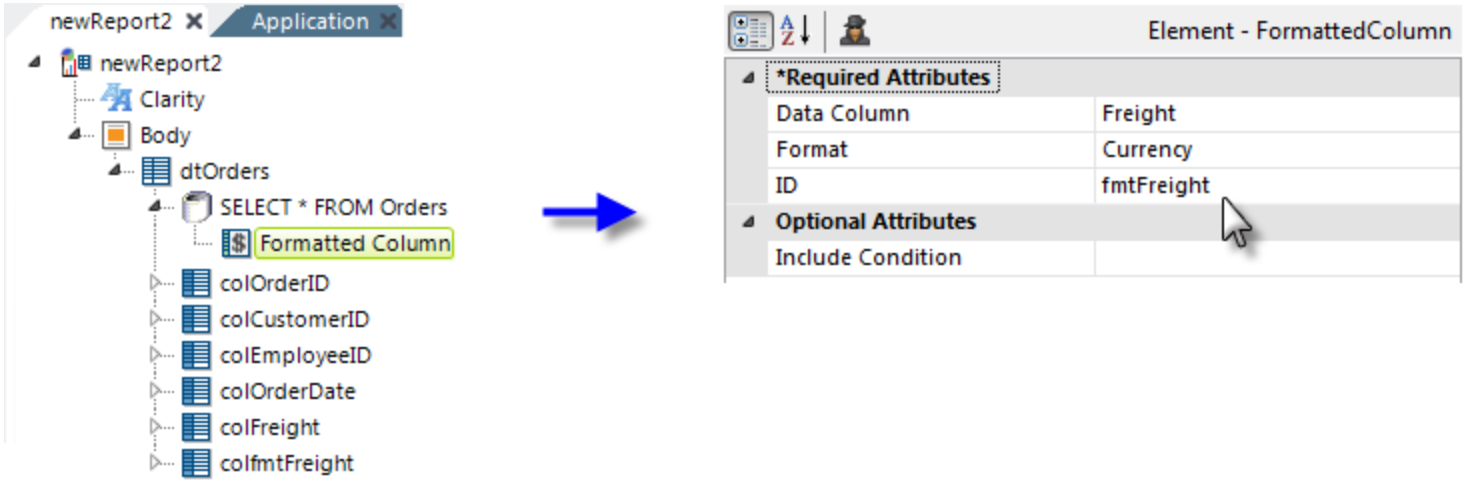
For each of the examples, imagine that the SQL query shown above has been used to retrieve data for a Data Table.

OrderID	CustomerID	EmpID	OrderDate	Freight
10248	VINET	5	2010-07-04T00:00:00.000	32.38
10249	TOMSP	6	2010-07-05T00:00:00.000	11.61
10250	HANAR	4	2010-07-08T00:00:00.000	65.83
10251	VICTE	3	2010-07-08T00:00:00.000	41.34
10252	SUPRD	4	2010-07-09T00:00:00.000	51.3

The data retrieved into the datalayer looks like the example shown above. Next, we'll add different kinds a formatted columns.

Format: Currency

It's often useful to apply the standard *Currency* format to monetary data.



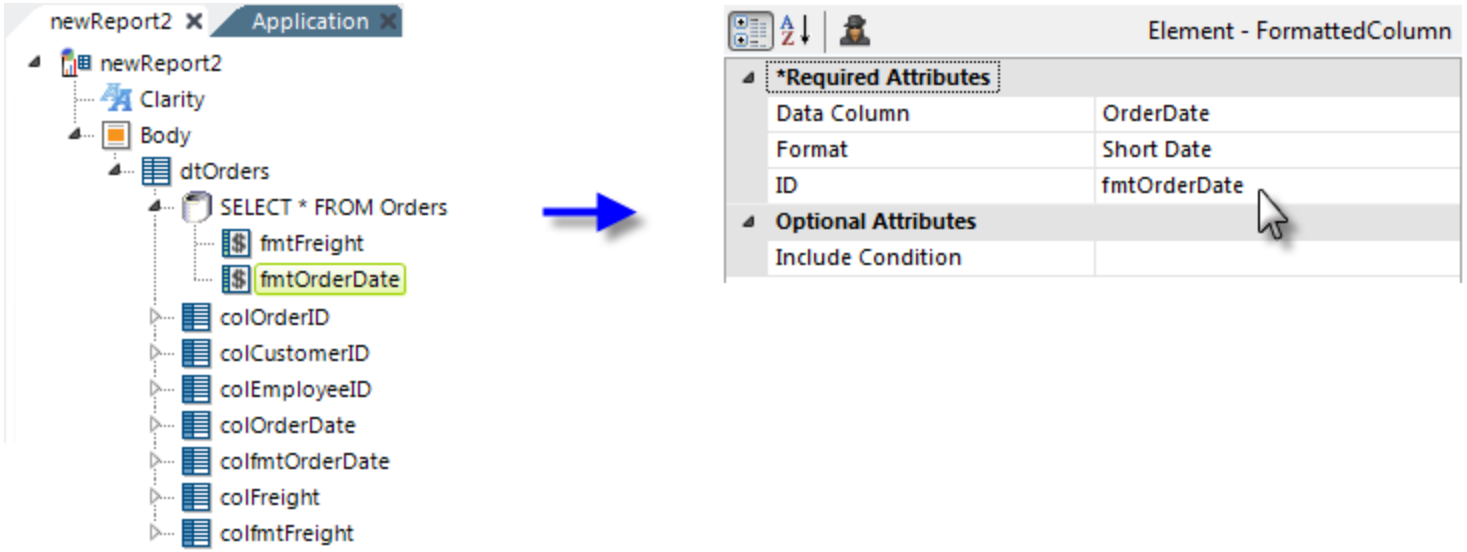
If we add a **Formatted Column** element beneath the datalayer, as shown above, set its attributes as shown, and provide another Data Table Column element to display the column added to the datalayer,

OrderID	CustomerID	EmpID	OrderDate	Freight	fmtFreight
10248	VINET	5	2010-07-04T00:00:00.000	32.38	$32.38
10249	TOMSP	6	2010-07-05T00:00:00.000	11.61	$11.61
10250	HANAR	4	2010-07-08T00:00:00.000	65.83	$65.83
10251	VICTE	3	2010-07-08T00:00:00.000	41.34	$41.34
10252	SUPRD	4	2010-07-09T00:00:00.000	51.3	$51.30

the output will change as shown above. In this case, the new column added to the datalayer contains the values from the *Freight* column, formatted as *Currency*.

Format: Short Date

Another common requirement is for date formatting. As we've seen in earlier output examples, the ISO format used to store DateTime values in many databases isn't very presentable.



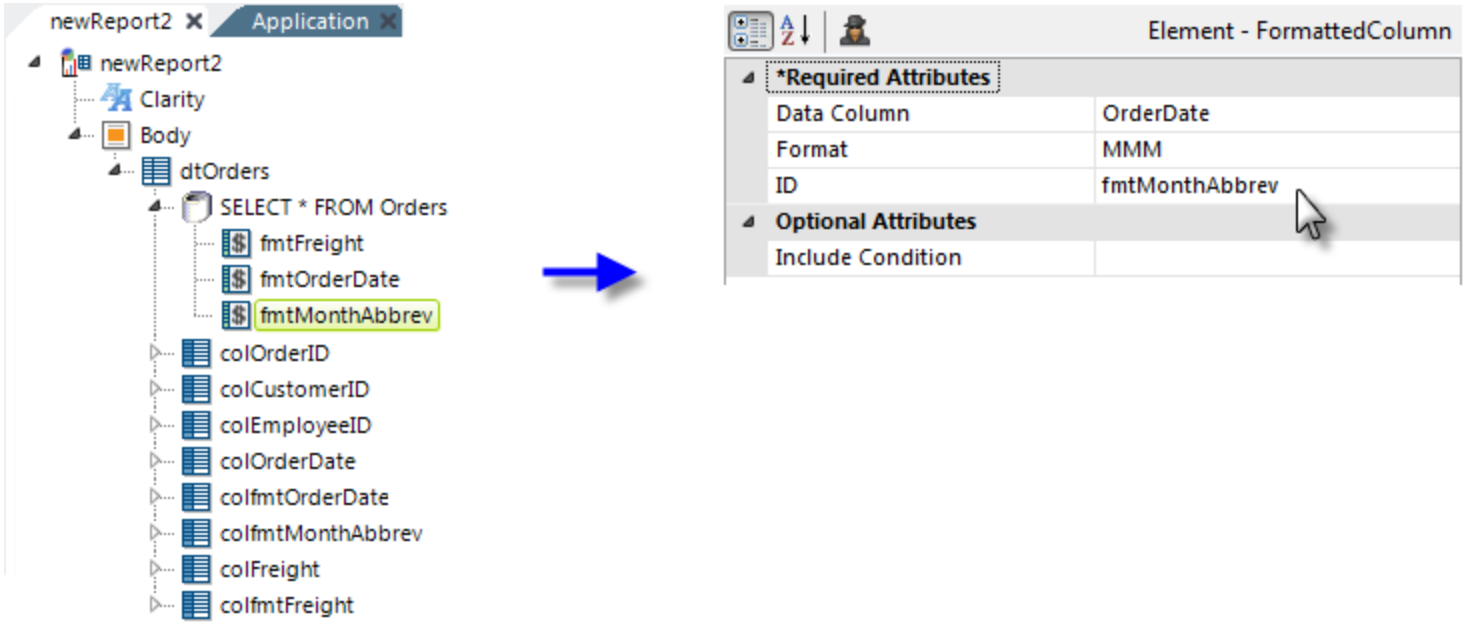
The example shown above illustrates how a Formatted Column element can be added to format dates using one of the standard date formats, *Short Date*,

OrderID	CustomerID	EmpID	OrderDate	fmtOrderDate	Freight	fmtFreight
10248	VINET	5	2010-07-04T00:00:00.000	7/4/2010	32.38	$32.38
10249	TOMSP	6	2010-07-05T00:00:00.000	7/5/2010	11.61	$11.61
10250	HANAR	4	2010-07-08T00:00:00.000	7/8/2010	65.83	$65.83
10251	VICTE	3	2010-07-08T00:00:00.000	7/8/2010	41.34	$41.34
10252	SUPRD	4	2010-07-09T00:00:00.000	7/9/2010	51.3	$51.30

and the resulting output is shown above.

Format: Month Abbreviations

It can be very useful, particularly when working with charts, to have a column that contains the *month name abbreviation* of a date value.



The example above shows how a custom format can be used; in this case the custom format value *MMM* will produce a 3-letter month abbreviation,

OrderID	CustomerID	EmpID	OrderDate	fmtOrderDate	fmtMonAbbrev	Freight	fmtFreight
10248	VINET	5	2010-07-04T00:00:00.000	7/4/2010	Jul	32.38	$32.38
10249	TOMSP	6	2010-07-05T00:00:00.000	7/5/2010	Jul	11.61	$11.61
10250	HANAR	4	2010-07-08T00:00:00.000	7/8/2010	Jul	65.83	$65.83
10251	VICTE	3	2010-07-08T00:00:00.000	7/8/2010	Jul	41.34	$41.34
10252	SUPRD	4	2010-07-09T00:00:00.000	7/9/2010	Jul	51.3	$51.30

and the resulting output is shown above. Incidentally, a format value of *MM* will produce the **month number** and *M* will produce the **month name and day** ("July 15"). All of these format values must be in uppercase.

As mentioned earlier, formatting data in the datalayer is much more efficient than formatting it elsewhere, so using Formatted Column elements can speed up report performance.

The Group Filter

The **Group Filter** groups the data rows in a datalayer and gives youthe ability to create aggregate values from the grouped rows.

The following topics discuss the Group Filter element:

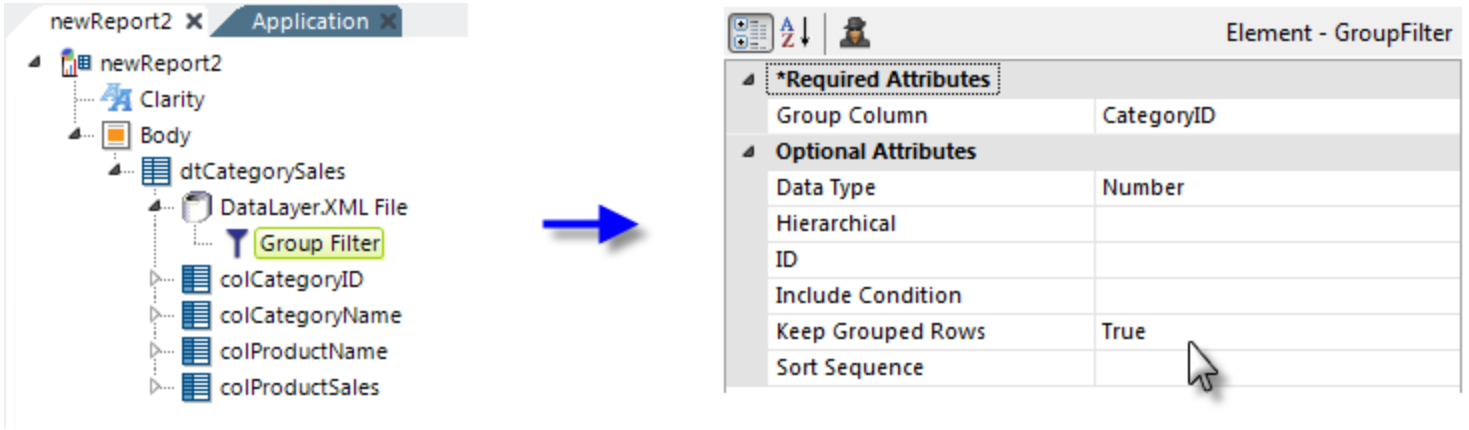
- Grouping Data
- Grouping for Unique Rows
- Grouping on Multiple Columns
- Conditional Grouping
- Aggregating Grouped Data
- Using Group DrillThrough

About the Group Filter

The behavior of the Group Filter element is similar to that of the GROUP BY clause in a SQL query. However, unlike a query, the element applies its grouping to data *after* it's been retrieved into the datalayer. This is very useful as it allows grouping to be applied to *all* kinds of data, including data from sources that lack SQL-like features, such as XML files and web services.

Grouping Data

The following example uses a **Group Filter** element to group the data in the datalayer on a specific column:



1. As shown above, a **Group Filter** element is added as a child of the datalayer element.
2. Its **Group Column** attribute is set to the name of a column in the datalayer. The data will be grouped based on the values in this column.
3. Its **Data Type** column is set to the data type of the column named in the Group Column attribute. This ensures unambiguous ordering of the data and, though optional, setting it is recommended.
4. Its **Keep Grouped Rows** attribute is set to *True* to ensure that all grouped rows are retained in the datalayer. This is necessary if you're going to aggregate the grouped rows. If you want to work with unique rows, leave this attribute blank (see below).

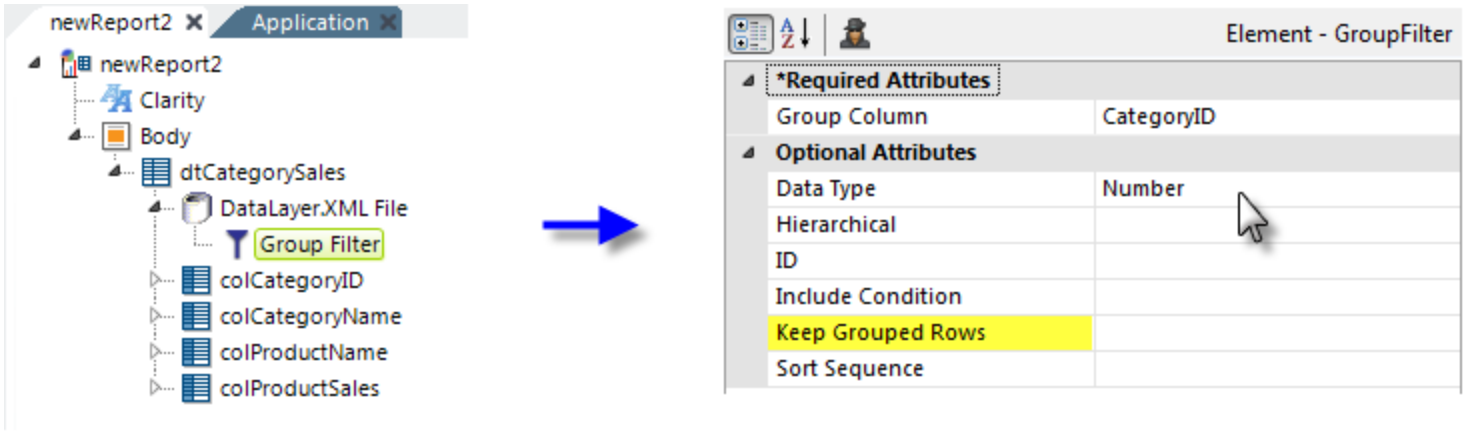
CategoryID	CategoryName	ProductName	ProductSales
1	Beverages	Chai	4887
1	Beverages	Chang	7038.55
1	Beverages	Chartreuse verte	4475.7
1	Beverages	Côte de Blaye	49198.09
2	Condiments	Aniseed Syrup	1724
2	Condiments	Chef Anton's Cajun Seasoning	5214.88
2	Condiments	Chef Anton's Gumbo Mix	373.63
3	Confections	Chocolade	1282.01
4	Dairy	Camembert Pierrot	20505.4
6	Meat-Poultry	Alice Mutton	17604.6
8	Seafood	Boston Crab Meat	9814.73
8	Seafood	Carnarvon Tigers	15950

The resulting output is shown above. The rows are grouped by CategoryID and all rows for each category are included. Other optional Group Filter attributes include:

- **Hierarchical** - Groups the rows into a hierarchical structure. See *Hierarchical Data* for more details.
- **Include Condition** - Specifies whether grouping will occur when the report runs. If left blank or if it contains an expression that evaluates to *True*, grouping will occur. If it evaluates to *False*, the element is ignored and grouping will not occur. More than one group filter can be put under any level, specified by the attribute "IncludeCondition" for each group filter.
- **Sort Sequence** allows you to control the way in which grouped rows are sorted. The default (blank) value causes *Ascending* order to be used, but you can change the value to *Descending* if desired.

Grouping for Unique Rows

There are many situations in which you may need to use only *unique* rows, for instance, to populate an **Input Select List** element. This can be done with grouping, as the following example illustrates:



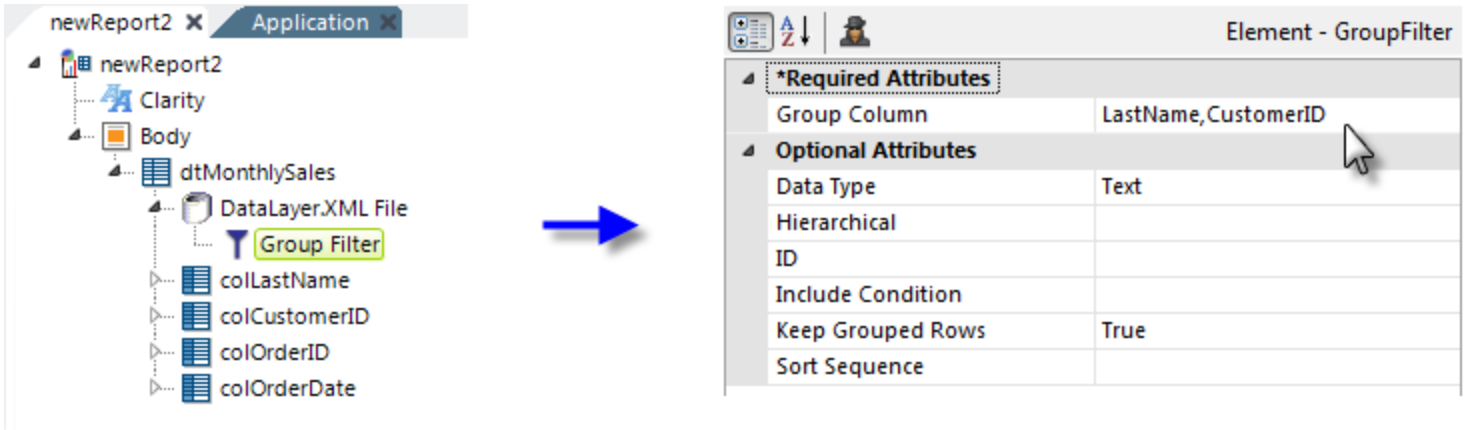
1. As shown above, a **Group Filter** element is added as a child to the datalayer element.
2. Its **Group Column** attribute is set to the name of a column in the datalayer. The data will be grouped based on the values in this column.
3. Its **Data Type** column is set to the data type of the column named in the Group Column attribute. This ensures unambiguous ordering of the data and, though optional, setting it is recommended.
4. The **Keep Grouped Rows** attribute is left *blank*; its default value is *False*. This causes all rows after the first one in each group to be *removed* from the datalayer, creating a list of unique values.

CategoryID	CategoryName	ProductName	ProductSales
1	Beverages	Chai	4887
2	Condiments	Aniseed Syrup	1724
3	Confections	Chocolade	1282.01
4	Dairy	Camembert Pierrot	20505.4
6	Meat-Poultry	Alice Mutton	17604.6
8	Seafood	Boston Crab Meat	9814.73

The resulting output is shown above. The rows are grouped by CategoryID and only one row exists for each category.

Grouping on Multiple Columns

Grouping can be accomplished on more than one column in the datalayer at the same time. This example shows the effects of grouping multiple columns:



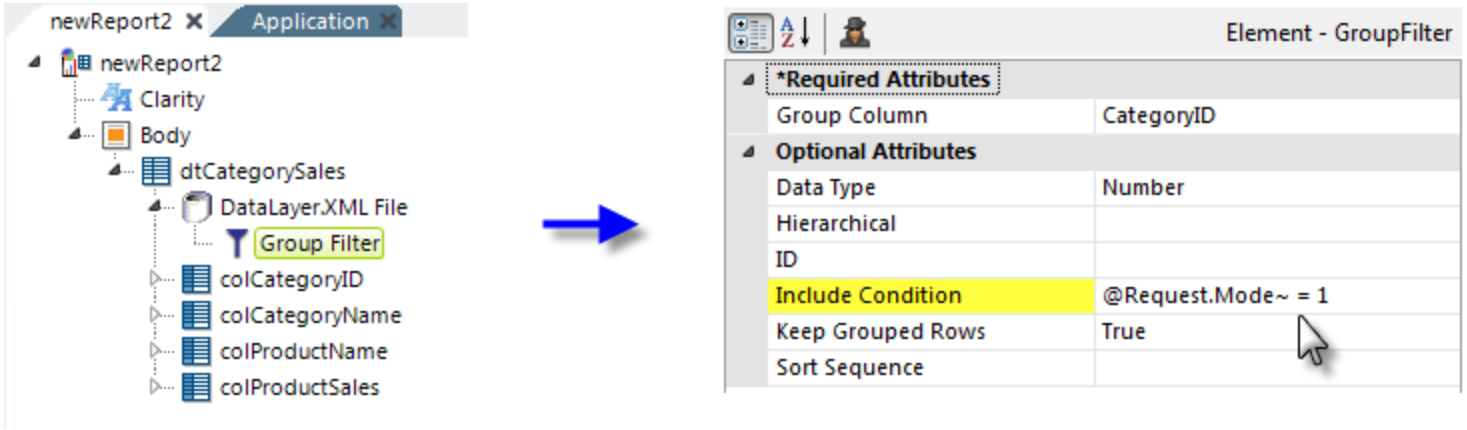
1. As shown above, a **Group Filter** element is added as a child to the datalayer element.
2. Its **Group Column** attribute is set to the names of columns in the datalayer, in a comma-separated list. The data will be grouped based on the values in these columns, in the order of the column names in the list.
3. Its **Data Type** column is set to the data type of the column named in the Group Column attribute. This ensures unambiguous ordering of the data and, though optional, setting it is recommended. If the columns listed in Group Column are of different data types, individual data types can also be entered here in a comma-separated list as well.
4. Its **Keep Grouped Rows** attribute is set to *True* to ensure that all rows are retained in the datalayer.

LastName	CustomerID	OrderID	OrderDate
Callahan	CONSH	10435	2/4/2011
Callahan	ERNSH	10402	1/2/2011
Callahan	ERNSH	10408	1/8/2011
Callahan	QUEDE	10421	1/21/2011
Callahan	REGGC	10443	2/12/2011
Callahan	VICTE	10450	2/19/2011
Callahan	WARTH	10412	1/13/2011
Callahan	WARTH	10416	1/16/2011
Callahan	WARTH	10437	2/5/2011
Davolio	EASTC	10400	1/1/2011
Davolio	LINOD	10405	1/6/2011
Fuller	FAMIA	10414	1/14/2011
Fuller	FAMIA	10422	1/22/2011
Fuller	OTTIK	10404	1/3/2011
Fuller	OTTIK	10407	1/7/2011

The resulting output is shown above. The rows are grouped first by LastName and then by CustomerID. The optional **Sort Sequence** attribute allows you to control the way in which grouped rows are sorted. The default (blank) value causes *Ascending* order to be used, but you can change the value to *Descending* if desired. When grouping on multiple columns, you can control the sort sequence for each grouping by entering multiple sort order values here, in a comma-separated list that corresponds to the column list.

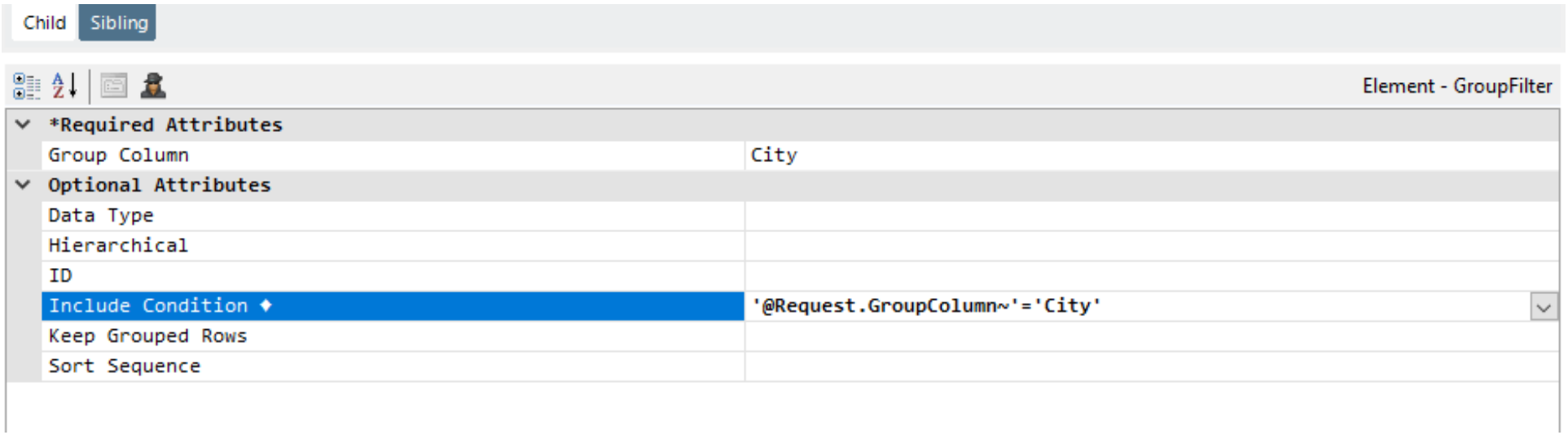
Conditional Grouping

The Group Filter element has an **Include Condition** attribute.



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows you to dynamically determine if the data will be grouped or not.

More than one group filter can be put under any level, specified by the attribute "IncludeCondition" for each group filter. For example, @Request.GroupColumn~ = 'City' or IncludeCondition =@Request.GroupColumn~ = 'Sales'. Once the Report is run without an error, there will be different grouping based on the given token value 'City' or 'Sales'.

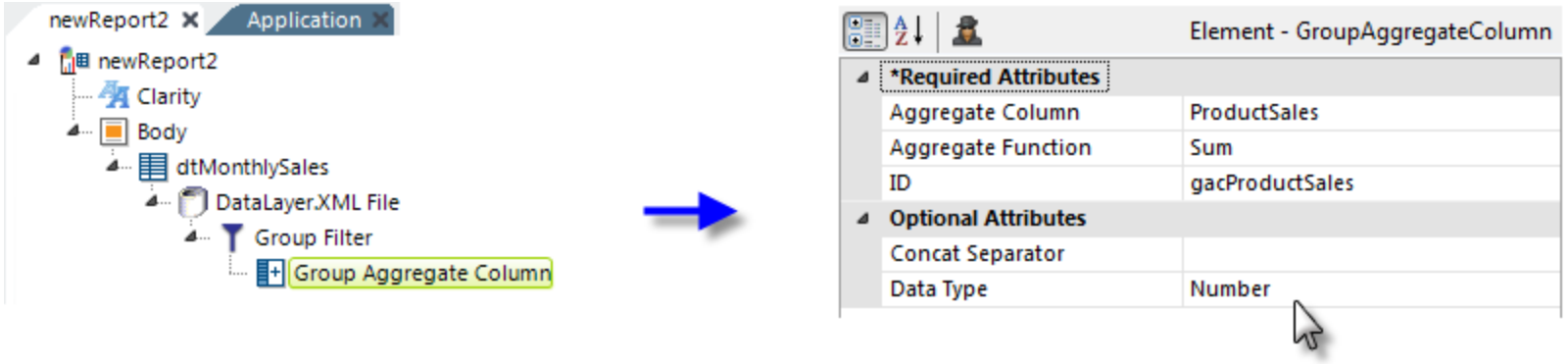


In this case, the token value set in the parameters was 'City' grouping the filter by City:

City	Total Sales
Berlin	$400.00
New York	$180.00
Toronto	$120.00

Aggregating Grouped Data

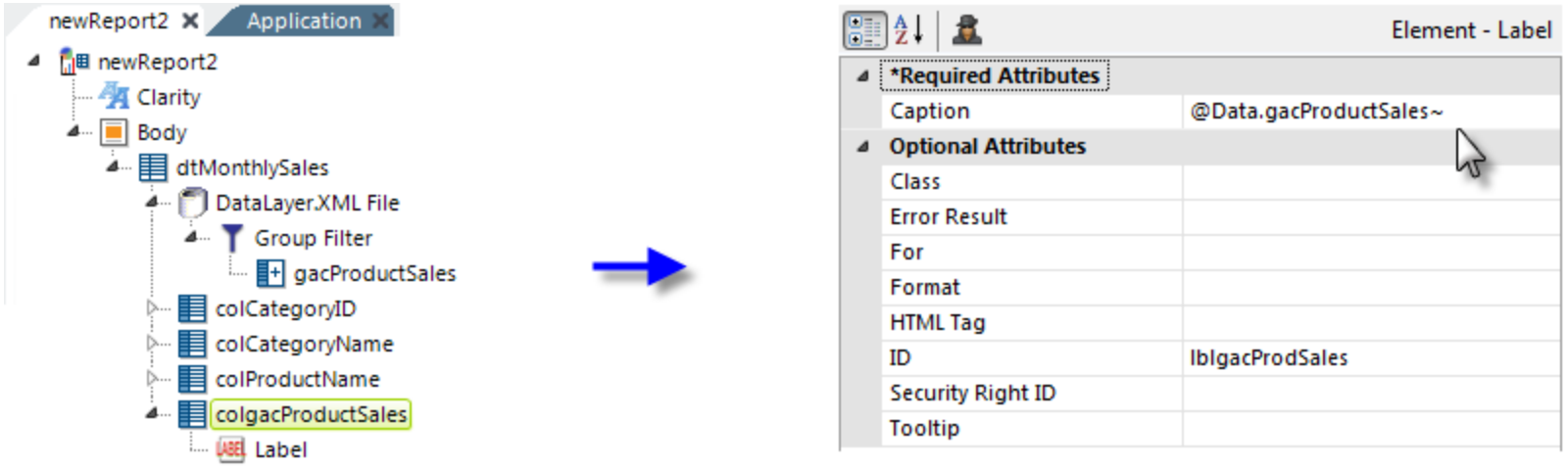
A very common requirement when grouping data is to perform some kind of *aggregation*, such as summing or counting, of the grouped values. The **Group Aggregate Column** element gives you the ability to generate this summary data for each grouped column in the datalayer. The following example builds on the earlier example in "Grouping for Unique Rows" on page 331, creating a total product sales value for each product category. For simplicity, the necessary Data Table Column elements are not shown.



1. As shown above, a **Group Aggregate Column** element is added to the earlier example.
2. Its **Aggregate Column** attribute value is set to the name of the column that we want to aggregate.
3. Its **Aggregate Function** attribute value is set to *Sum*. Other options include *Average*, *Concat, Count*, *DistinctCount*, *Max*, *Median*, *Min*, *Mode*, and *Stdev*.

   Aggregations *exclude* columns with Null values by default. If you want to include them create a constant, `rdCalculationsIncludeNulls`, in your _Settings definition and set its value to *True*.
4. Its **ID** attribute specifies the name of a new column that will be added to the datalayer, containing the summarized data values.

5. Its **Data Type** attribute specifies the data type of the column named in the Aggregate Column attribute.



The data in the **new column** in the datalayer is available at runtime like any other, as shown above, through the use of an @Data token.

CategoryID	CategoryName	ProductName	Total Sales
1	Beverages	Chai	65599.34
2	Condiments	Aniseed Syrup	7312.51
3	Confections	Chocolade	1282.01
4	Dairy	Camembert Pierrot	28944.16
5	Grains-Cereals	Filo Mix	2124.15
6	Meat-Poultry	Alice Mutton	17604.6
8	Seafood	Boston Crab Meat	27841.01

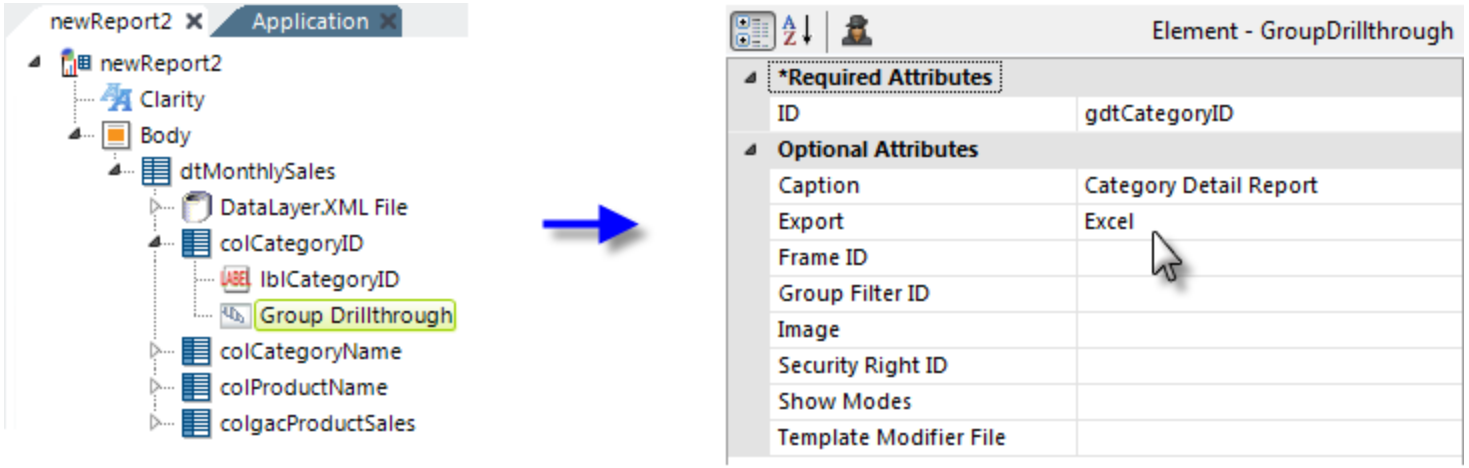
And the resulting output is shown above, in a new column given the header caption "Total Sales". All records have been grouped by CategoryID, only the first record for each group has been retained, and ProductSales values for each group have been summed and are now available in the new column.

Using Group DrillThrough

Logi Info includes special "drill-through" features that allow you to view the detailed data that was used in the grouping operation, in an automatically-generated tabular report. Two elements, **Group Drillthrough** and its optional child, **Drillthrough Column**, produce the view. They may be used beneath these elements:

- Chart Canvas and many Series elements
- Crosstab Table Value Column
- Data Table Column

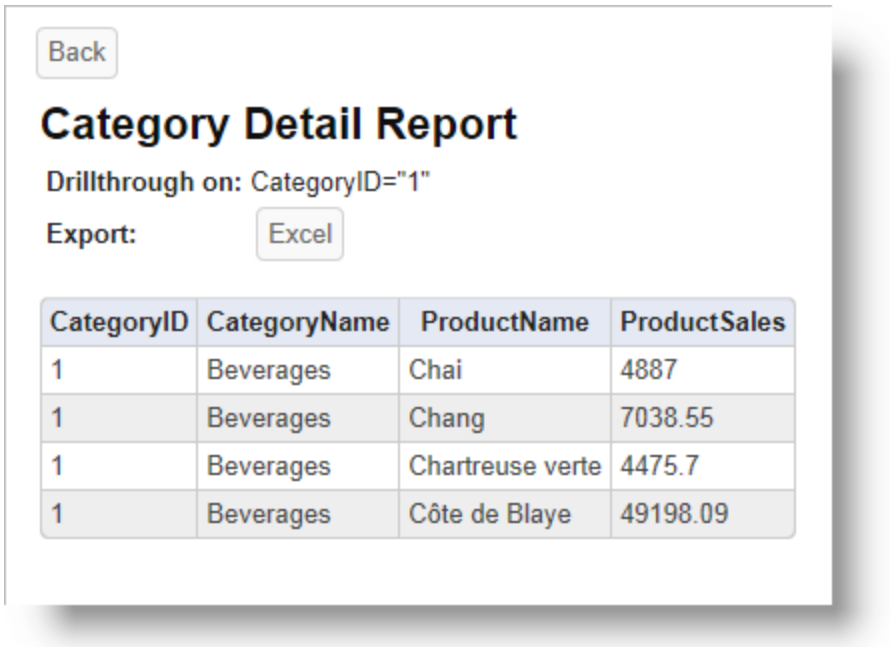
Let's start with the example we saw in "Aggregating Grouped Data" on page 337: grouped data, with unique rows and a group aggregate, summing the sales values. Our goal is to be able to review the individual values that went into the total for each group.



We'll add a **Group Drillthrough** element, as shown above, beneath the Data Table Column element for the column used to do the grouping (CategoryID). All of the element's attributes, other than ID, are optional but we'll set a custom caption for the detail report and an export button. More information about its attributes can be found in the Group Drillthrough element's Element Reference page.

CategoryID	CategoryName	ProductName	Total Sales
1	Beverages	Chai	65599.34
2	Condiments	Aniseed Syrup	7312.51
3	Confections	Chocolade	1282.01
4	Dairy	Camembert Pierrot	28944.16
5	Grains-Cereals	Filo Mix	2124.15
6	Meat-Poultry	Alice Mutton	17604.6
8	Seafood	Boston Crab Meat	27841.01

When the table is displayed and the cursor is hovered over the CategoryID column, a drillthrough icon will now appear, as shown above.



When the icon is clicked, the detail report shown above will be displayed. The report is generated automatically and you do not need to do anything other than include the Group Drillthrough element to get the standard report. However, the element's attributes do allow you to customize the icon image and report and, by including **Drillthrough Column** elements beneath it, you can set the number columns that will appear in the report and customize their appearance, too.

The RegEx Filter

The **RegEx Filter** element is used with datalayer elements to filter out data rows. It works by applying pattern matching using regular expressions, providing developers with a great deal of flexibility in making the comparison.

The following topics discuss the RegEx Filter:

- Applying the RegEx Filter
- Using Multiple and Dynamic RegEx Filters

About the RegEx Filter

"RegEx" is an abbreviation of "regular expression", which provides a concise and flexible method for "matching" (specifying and recognizing) text strings, including particular characters, words, and/or patterns of characters.

The **RegEx Filter** element is available for use with all datalayer elements and is particularly useful in conjunction with those lacking query capabilities,such as DataLayer.XML and DataLayer.Web Service.

In operation, the element basically attempts to match a specified data column value against a specified regular expression pattern and it removes all rows from the datalayer where there is no match (the match result is *False*).

The RegEx Filter's pattern matching can be very useful for finding and removing "bad" data; that is, data that's malformed or incomplete. Consider this example regular expression: `^3[47][0-9]{13}$`which was written to validate data that's supposed to be American Express credit card numbers. The expression indicates that each number should start with either 34 or 37 and has to have 15 digits.

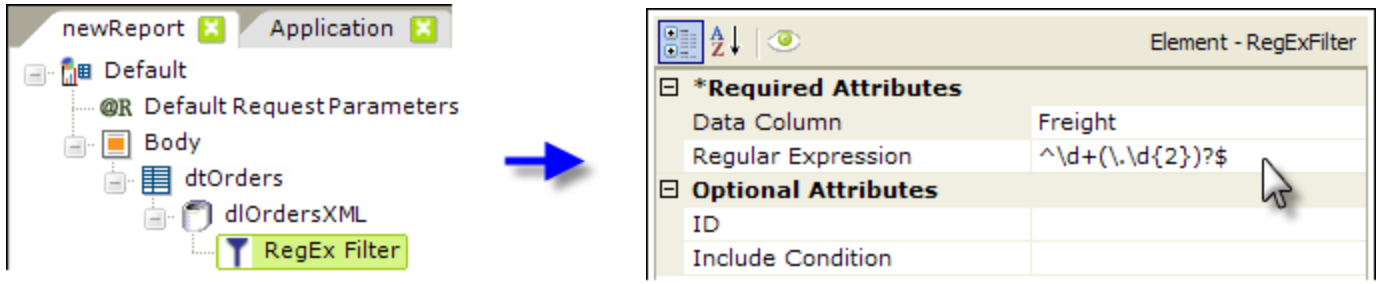
For developers who may not be familiar with regular expressions, the following external site provides useful information:

http://www.regular-expressions.info/examples.html

The RegEx Filter element is similar to the "The Condition Filter" on page 248 element, however, it frequently provides *better performance* because it's implemented as a native part of the Logi Data Engine and so does not need to execute script, as the Condition Filter does.

Applying the RegEx Filter

The following example illustrates how the **RegEx Filter** is used:



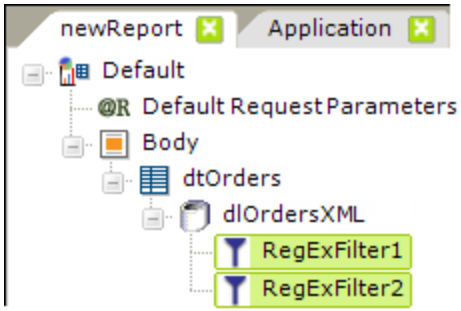
1. As shown above, a **RegEx Filter** element is added as a child to a datalayer element.
2. Its **Data Column** attribute value is set to the column in that datalayer whose value will be examined.
3. The **Regular Expression** attribute is set to define the desired regular expression code.

Using the example above, the sequence of events is that the datalayer reads in all the data from the datasource, then the RegEx Filter **removes** all rows whose Freight column values don't match the regular expression pattern (unlimited digits before the decimal point, two digits after it, and no dollar signs, commas, text, or symbols). All that remains in the datalayer for use in the report are rows whose Freight values match the pattern.

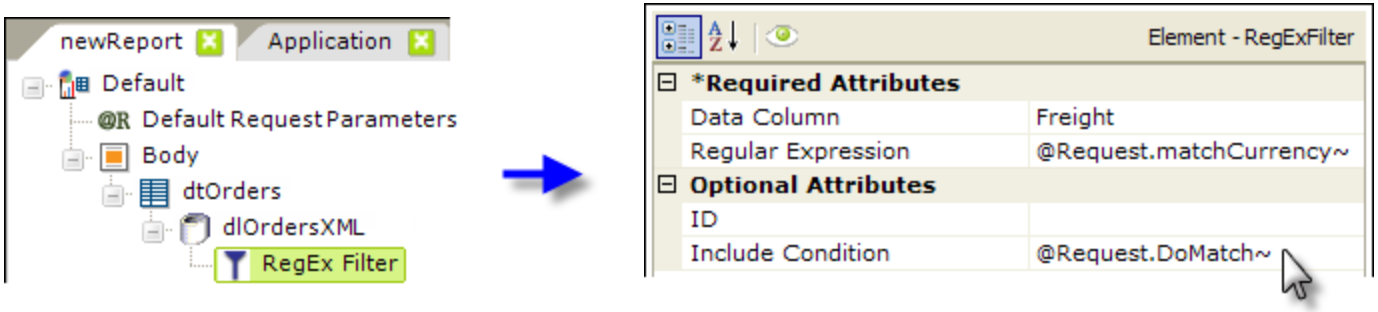
Using Multiple and Dynamic RegEx Filters

This topic introduces how to use multiple and dynamic RegEx filters.

1. Developers can use two or more **consecutive** RegEx Filter elements:



In this case, the matches will be evaluated **sequentially**, based on their arrangement from top to bottom in the element tree. So, after the first RegEx Filter is applied, only records that match its pattern will remain in the datalayer to be matched to the second RegEx Filter. 2. The RegEx Filter element also has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows developers to dynamically determine if the datalayer will be filtered or not.

The Relevance Filter

As a Logi application developer, you may find yourself working with data that contains records that are *not relevant* to the analysis. This data could, for example, have values that are too small for individual inclusion. The **Relevance Filter** element provides a method of distinguishing between *relevant* and *irrelevant* data.

- About the Relevance Filter
- Applying the Relevance Filter

About the Relevance Filter

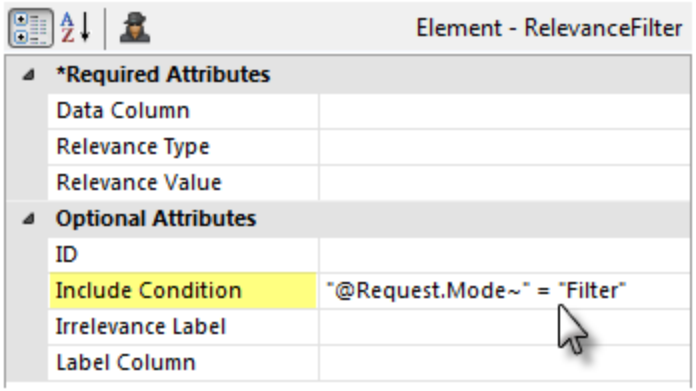
The concept of *irrelevant data* is at the heart of understanding the **Relevance Filter**. Data relevance is determined by some criteria you establish - cost or quantity, for example - and data that fails to meet that criteria should be left out of your analysis or report.

At the same time, it may be useful instead to *gather* the irrelevant data and present it collectively, rather than leave it out altogether. It might be useful to present it, for example, grouped together as "Other" data.

The **Relevance Filter** element, used with a datalayer, allows you to establish relevance criteria and cause data rows that don't meet that criteria to be removed from the datalayer. Optionally, the irrelevant data rows can be retained and grouped together for presentation along with relevant data groupings.

Dynamic Filtering

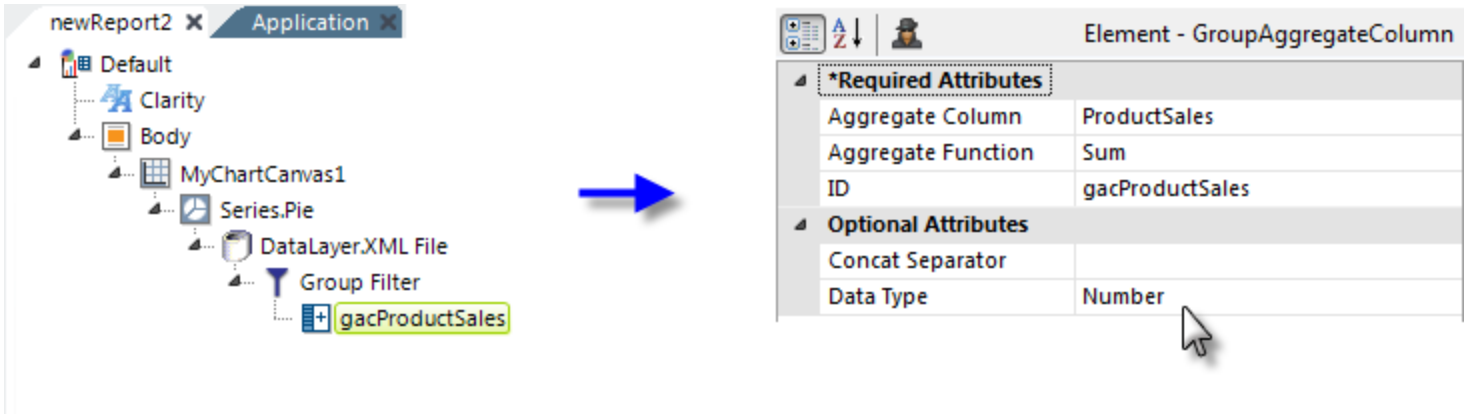
The Relevance Filter element has an **Include Condition** attribute:



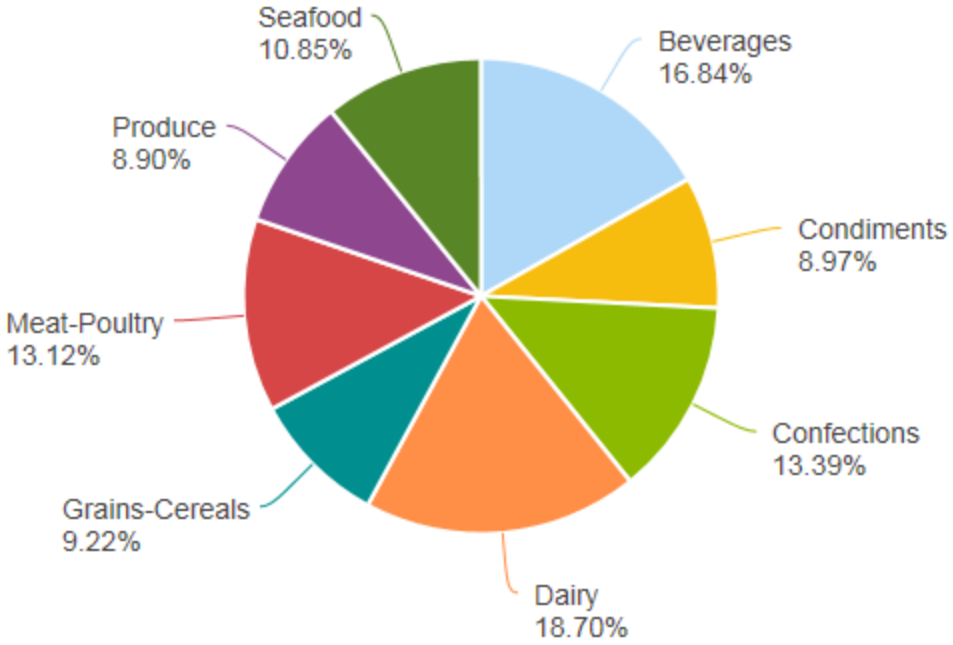
If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows you to determine at runtime if the datalayer will be filtered or not.

Applying the Relevance Filter

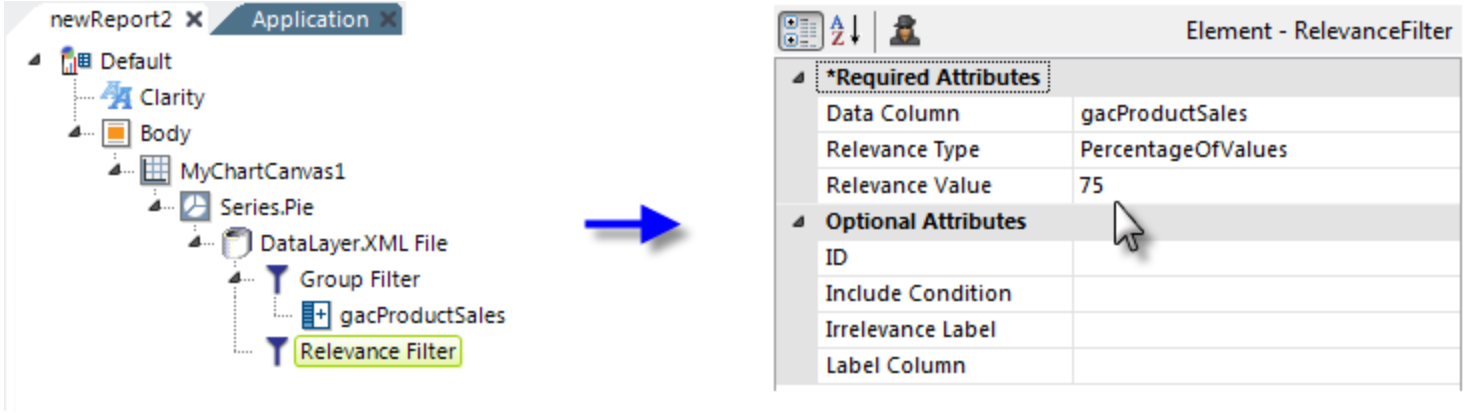
The following examples show how a **Relevance Filter** can be used with a pie chart:



1. As shown above, we start with a simple pie chart, working with product data, and use a **Group Filter** element to group on the *CategoryName* column.
2. A **Group Aggregate Column** element is added beneath the Group Filter to sum the *ProductSales* column for each category group. This produces a new column in the datalayer, named *gacProductSales*.

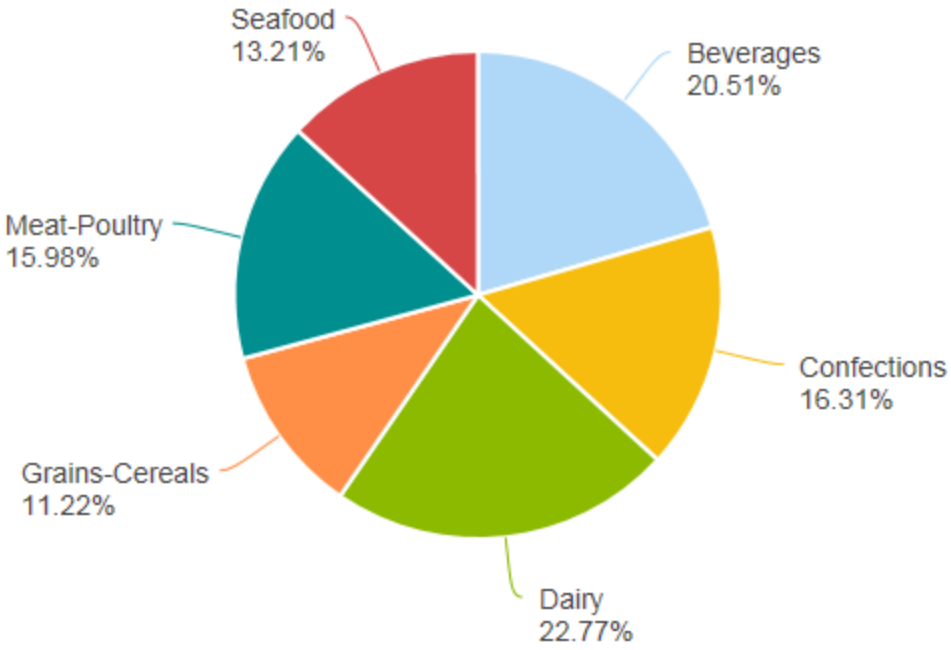


When the report is run, the pie chart shown above is displayed. There are several smaller wedges for the **Produce**, **Condiments**, and **Grain/Cereals** categories. These are the categories (and data) we're going to target in the next example as *irrelevant*.

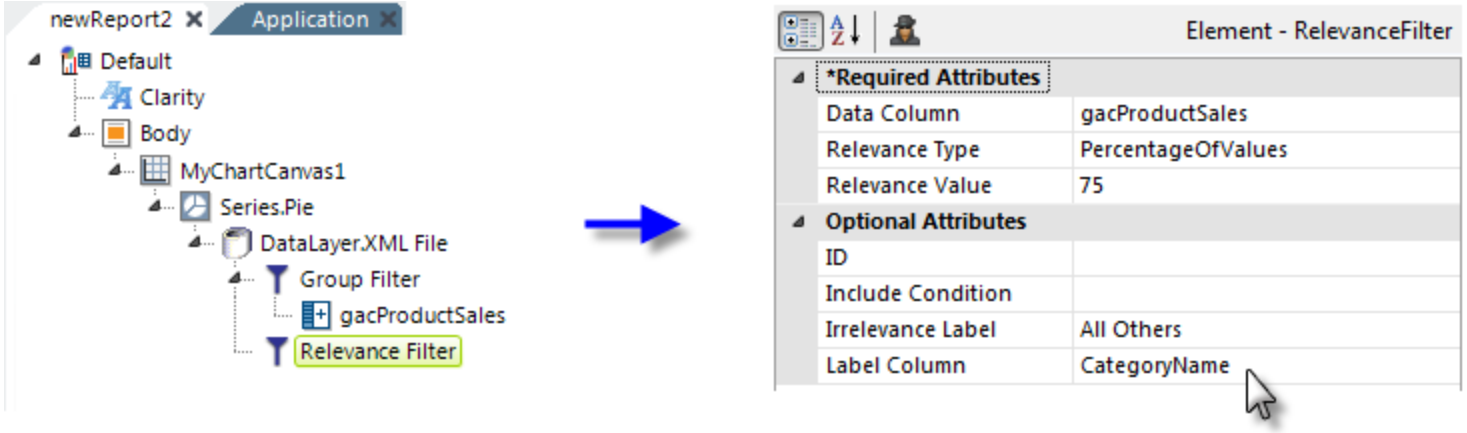


1. In this example, a **Relevance Filter** element is added beneath the datalayer, as shown above. Its **Data Column** attribute is set to the ID of the column that sums the number of units in stock for the categories.
2. Its **Relevance Type** attribute value is set to *PercentageOfValues* and its **Relevance Value** attribute to *75*. This indicates that rows in the datalayer whose *gacProductSales* column value is less than 25% of the total will be removed. In other

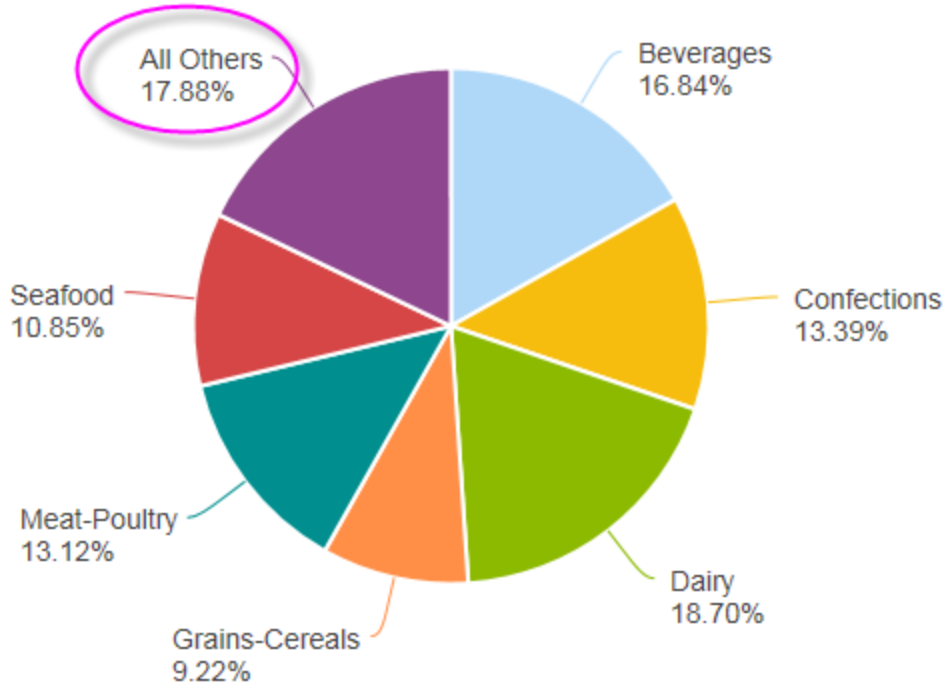
words, the "bottom 25%" will be dropped.



Now you can see, as shown above, that the smaller wedges - the irrelevant data - no longer appear in the pie chart. We also have the option of keeping the irrelevant data, collected in a separate group, as shown in the next example.



1. In order to use the irrelevant data as a group, the Relevance Filter element's **Irrelevance Label** attribute is given the value we want to appear identifying this group in the chart.
2. Its **Label Column** attribute is set to the column in the datalayer whose value we want to replace with the Irrelevance Label value we just specified, in this case *CategoryName*.



And, as shown above, the irrelevant data has been grouped into a category of its own, called "All Others", and added to the chart. The Relevance Filter can use three different methods for calculating relevant values:

- **Percentage of Values** - as we've seen, this method removes rows that fall below a certain percentage of the total value.
- **Number of Rows** - this method retains the top *n* rows based on value and discards the rest.
- **Value** - this method retains rows whose value falls at or above below the value specified and discards the rest.

By using the Relevance Filter, we're able to reduce the visual clutter that a lot of small wedges might create, while still accounting for their value in the chart (the wedges still add up to 100%).

The **Relevance Reverse** attribute, when set to *True*, causes the filter to return the rows with the *smallest* values, rather than the largest.

The Sort Filter

The **Sort Filter** sorts data rows in a datalayer element, giving youcontrol over the *sort sequence* of the data. This topic discusses the use of this element.

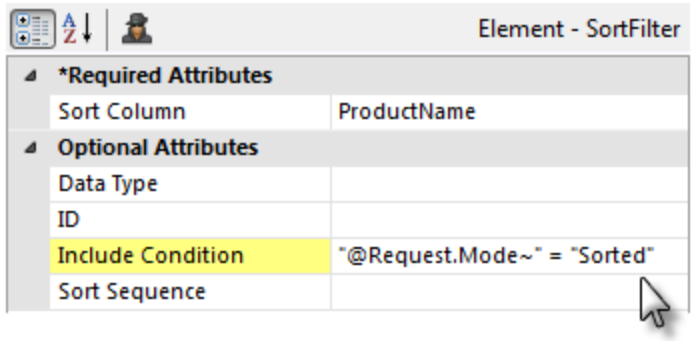
- About the Sort Filter
- Applying the Sort Filter

About the Sort Filter

The **Sort Filter** element's behavior is similar to the *ORDER BY* clause in a SQL statement but, unlike a query, it's applied to data *after* it's been retrieved into the datalayer. The Sort Filter is available to all datalayer elements and is especially useful for non-SQL datasources, such as DataLayer.XML and DataLayer.Web Service.

Dynamic Sorting

The Sort Filter element has an **Include Condition** attribute:



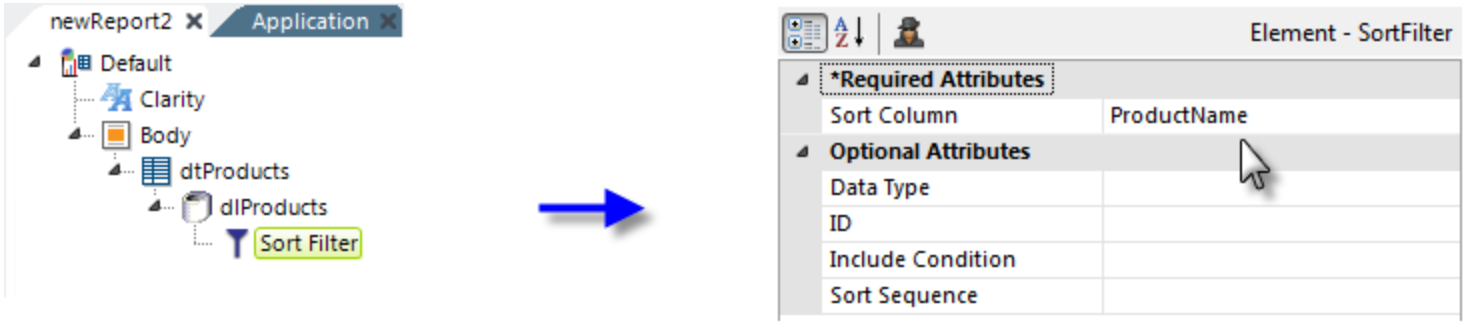
If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows you to dynamically determine if the datalayer will be sorted or not.

Applying the Sort Filter

The following examples illustrate how the Sort Filter can be used:

ProductID	CategoryName	ProductName	UnitPrice	UnitsInStock
1	Beverages	Chai	$18.00	39
2	Beverages	Chang	$19.00	17
3	Condiments	Aniseed Syrup	$10.00	13
4	Condiments	Chef Antons Cajun Seasoning	$22.00	53
5	Condiments	Chef Antons Gumbo Mix	$21.35	0
6	Condiments	Grandmas Boysenberry Spread	$25.00	120
8	Condiments	Northwoods Cranberry Sauce	$40.00	6
9	Meat/Poultry	Mishi Kobe Niku	$97.00	29
7	Produce	Uncle Bobs Organic Dried Pears	$30.00	15
10	Seafood	Ikura	$31.00	31

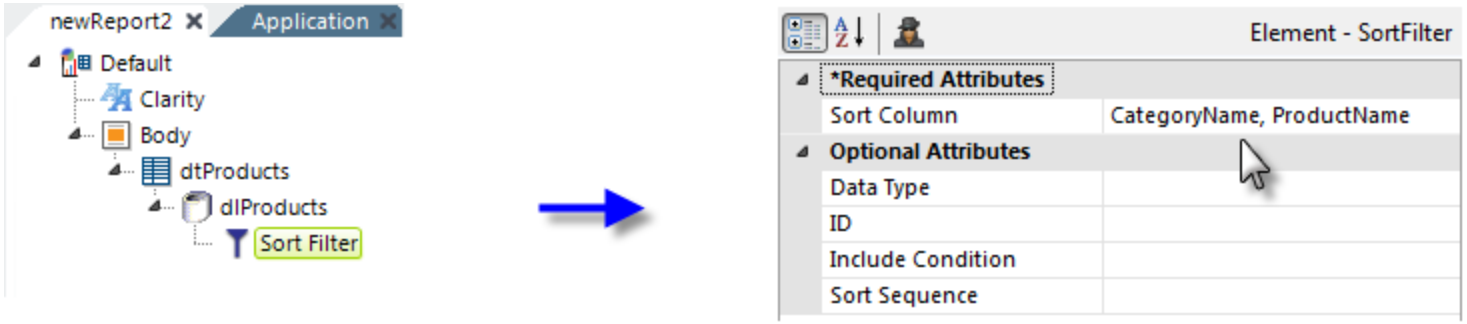
The table above shows the data as it appears in the datalayer, without any sorting. Note the order of the ProductName column data. Now let's sort that column:



1. As shown above, add **Sort Filter** element beneath a datalayer element.
2. Set its **Sort Column** attribute value to the name of the column in the datalayer, *ProductName*, that the sort will be based on.
3. The optional **Data Type** attribute value, which defaults to *Text*, specifies the data type of the column being sorted on and, to avoid any ambiguity, should always be set if the column data type is *Number* or *Date*.
4. The optional **Sort Sequence** attribute value, which defaults to *Ascending*, can be set to *Descending* if desired.

ProductID	CategoryName	ProductName	UnitPrice	UnitsInStock
3	Condiments	Aniseed Syrup	$10.00	13
1	Beverages	Chai	$18.00	39
2	Beverages	Chang	$19.00	17
4	Condiments	Chef Antons Cajun Seasoning	$22.00	53
5	Condiments	Chef Antons Gumbo Mix	$21.35	0
6	Condiments	Grandmas Boysenberry Spread	$25.00	120
10	Seafood	Ikura	$31.00	31
9	Meat/Poultry	Mishi Kobe Niku	$97.00	29
8	Condiments	Northwoods Cranberry Sauce	$40.00	6
7	Produce	Uncle Bobs Organic Dried Pears	$30.00	15

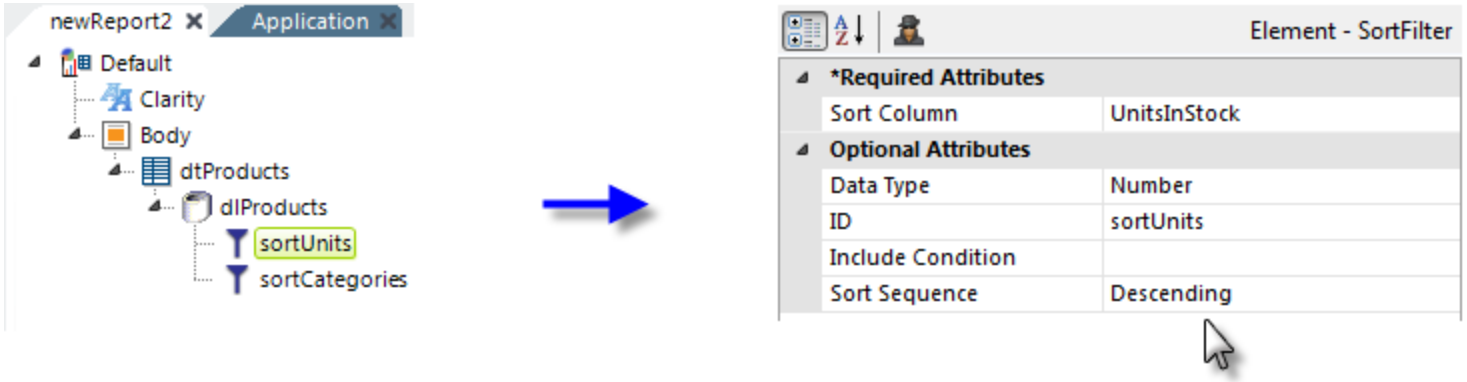
When the report is run again, the data will now be sorted in Ascending order on the *ProductName* column, as shown above.



5. If a second datalayer column is added to the first in the **Sort Column** attribute value, separated by a comma as shown above, the sort will occur on multiple columns.

ProductID	CategoryName	ProductName	UnitPrice	UnitsInStock
1	Beverages	Chai	$18.00	39
2	Beverages	Chang	$19.00	17
3	Condiments	Aniseed Syrup	$10.00	13
4	Condiments	Chef Antons Cajun Seasoning	$22.00	53
5	Condiments	Chef Antons Gumbo Mix	$21.35	0
6	Condiments	Grandmas Boysenberry Spread	$25.00	120
8	Condiments	Northwoods Cranberry Sauce	$40.00	6
9	Meat/Poultry	Mishi Kobe Niku	$97.00	29
7	Produce	Uncle Bobs Organic Dried Pears	$30.00	15
10	Seafood	Ikura	$31.00	31

The data will now be sorted first on one column and then on the second, as shown above. However, what about situations in which the columns to be sorted are **not** the same data type and/or need to be sorted in **opposite directions** (one in *Ascending* order and one in *Descending* order)?



6. In that case, additional **Sort Filter** elements can be used, one for each column, as shown above.

   The *order* of the two Sort Filter elements in the element tree is significant. In this example, the second Sort Filter has been added *above* the sort filter for *CategoryName*.

ProductID	CategoryName	ProductName	UnitPrice	UnitsInStock
1	Beverages	Chai	$18.00	39
2	Beverages	Chang	$19.00	17
6	Condiments	Grandmas Boysenberry Spread	$25.00	120
4	Condiments	Chef Antons Cajun Seasoning	$22.00	53
3	Condiments	Aniseed Syrup	$10.00	13
8	Condiments	Northwoods Cranberry Sauce	$40.00	6
5	Condiments	Chef Antons Gumbo Mix	$21.35	0
9	Meat/Poultry	Mishi Kobe Niku	$97.00	29
7	Produce	Uncle Bobs Organic Dried Pears	$30.00	15
10	Seafood	Ikura	$31.00	31

And the data will now be sorted in *ascending* order on one column and then in *descending* order on the second, as shown above.

The Sort Filter is very useful and can be added before or after grouping, datalayer links, and other elements to produce different data sets.

The Switch Column

The **Switch Column** element is used with datalayer elements to make data replacements. It's analogous to using both a CASE structure and a REPLACE function in a SQL query but, unlike a query, it's applied to the data *after* it's been retrieved into the datalayer.

The following topics discuss the Switch Column element:

- Applying the Switch Column
- Using Multiple and Dynamic Switch Columns

About the Switch Column

The **Switch Column** element is available for use with all datalayer elements and is particularly useful in conjunction with those lacking query capabilities, such as DataLayer.XML and DataLayer.Web Service.

In operation, the element basically searches a specified data column in each datalayer row for a specified value and, if found, *replacesit* with another value. Optionally, the replacement value can be put into a new column that's added to the datalayer, leaving the original column's data intact.

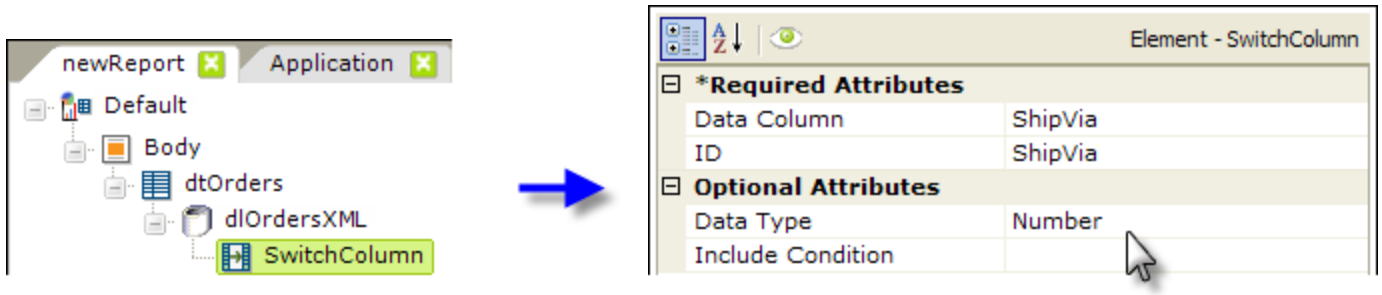
This is useful for replacing storage-friendly codes, such "1,2,3..." withreadable text, such as "Overnight Delivery,Second Day Delivery, Ground Delivery..." or "1, 0" into "True, False". These replacements are temporary (they happen onlyin the cached datalayer) and don't effect the original data in the datasource, so they're great for increasinglegibility in reports.

Applying the Switch Column

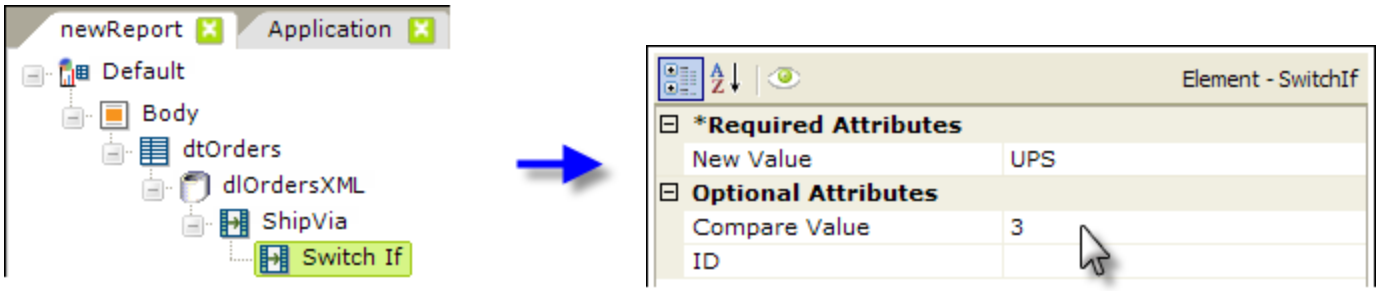
The following example illustrates how the **Switch Column** is used:

```
<dtOrders OrderID="10248" CustomerID="VINET" EmployeeID="5" ShipVia="3" ShipName="Vins Chevalier"
  ShipAddress="59 rue de l'Abbaye"  ShipCity="Reims" ShipPostalCode="51100" />
```

First, look at the example datalayer row above which shows the data as retrieved, with a numeric code for the **ShipVia** field.



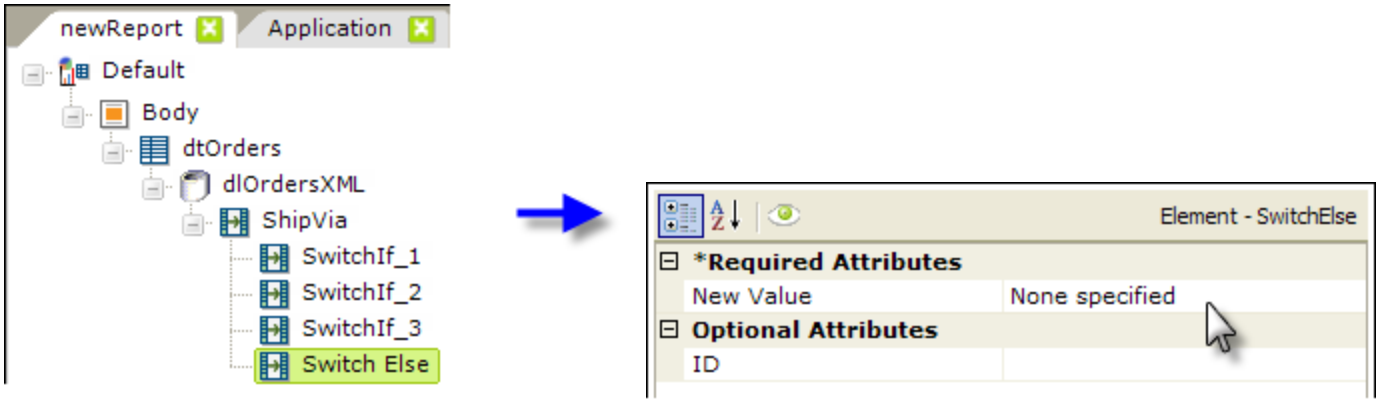
1. As shown above, a **Switch Column** element is added as a child beneatha datalayer element.
2. Its **Data Column** attribute value is set to the column in that datalayer whose value will be evaluated.
3. The **ID** attribute is set to name of a column that will receive the replacement data. If it's the *same* as the Data Column, as shown above, the original data will be replaced. If it's *different*, a new column by that name will be added to the datalayer.
4. The optional **Data Type** attribute helps the comparison routine when the data may be ambiguous.



5. Next add one (or more) **Switch If** elements beneath the Switch Column element, as shown above. These specify the **Compare Value** (3) which, when found, will trigger a replacement using the **New Value** ("UPS"). The evaluation is always "if the original value *equals* the comparison value, substitute the new value". You can leave the Compare Value *blank* in order to deliberately replace the original value in all rows.

```
<dtOrders OrderID="10248" CustomerID="VINET" EmployeeID="5" ShipVia="UPS" ShipName="Vins Chevalier"
  ShipAddress="59 rue de l'Abbaye"  ShipCity="Reims" ShipPostalCode="51100" />
```

If we look at the data again, we'll see that the ShipVia field has been changed.



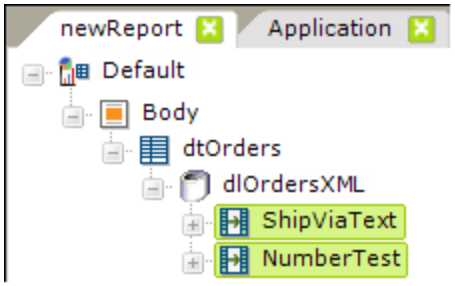
6. Finally, if desired, you can add an optional **Switch Else** element as a child of Switch Column which will replace the original value with a **New Value**, shown above, in any row unaffected by the preceding Switch If elements.

In the complete example above, the sequence of events is that the datalayer reads in all the data from the XML data file, then the Switch Column examines the specified data column in each datalayer row and, depending on the original value found there, replaces the original value with the specified new value. Any rows that aren't affected by one of the Switch If elements have their original value changed by the Switch Else element.

Using Multiple and Dynamic Switch Columns

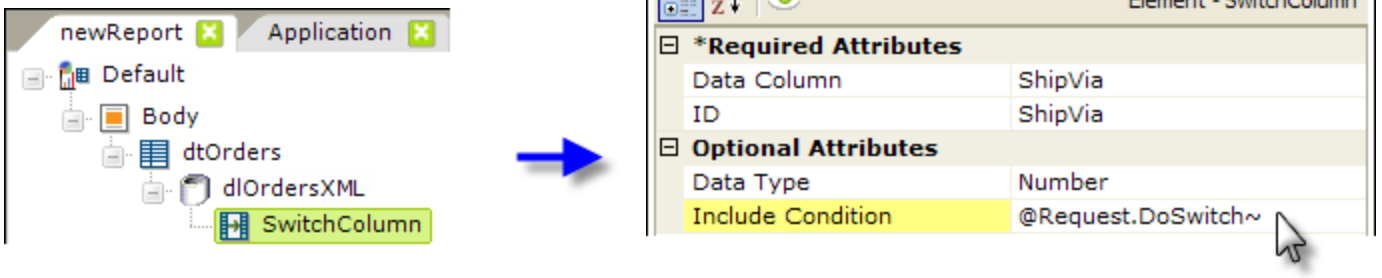
This topic introduces how to use multiple and dynamic Switch Columns.

1. Developers can use two or more *consecutive* Switch Column elements:



In this case, the operations will be conducted **sequentially**, based on their arrangement from top to bottom in the element tree. So, after the first Switch Column is applied, rows that may have been changed by it will be evaluated bythe second Switch Column.

2. The Switch Column element also has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows developers to dynamically determine if the datalayer will be altered or not.

Time Period Columns

Developers frequently need to work with parts of **dates** and **timestamps** and often employ various string parsing techniques to do so. The **Time Period Column** element, described in this topic, allows you to directly access the date and time parts of data without resorting to JavaScript functions and complicated parsing.

The following topics discuss the Time Period Columns element:

- Time Period Column Attributes
- Time Period Column Simple Usage Example
- Time Period Column Advanced Usage Example

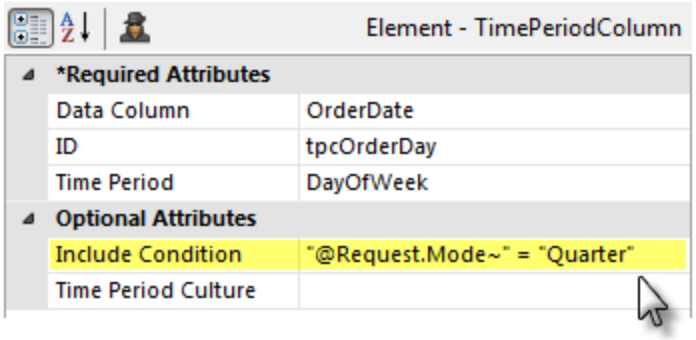
About the Time Period Column

The Time Period Column is similar to other elements designed to extend the data available in a datalayer. Like the **Calculated Column** element, the Time Period Column is added as a child element beneath a datalayer and **adds a new column** to the datalayer. This column contains derived data which is accessible using an @Data token. In this case, the derived data can be any of the parts of a DateTime value.

Time Period Columns can also be configured to be *culture-aware* so that the values derived will agree with the culture settings of the user's browser.

Dynamic Application

The Time Period Column element has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows developers to dynamically determine if the Time Period Column will be added to the datalayer or not.

Time Period Columns Attributes

The attributes for the Time Period Column element are described below:

Attribute	Description
Data Column	(Required) Specifies the name of the column in the datalayer that contains the DateTime data from which the Time Period Column will derive its data.
ID	(Required) Specifies a unique identifier for the element.
Time Period	(Required) Specifies the time period to be derived from the data column. Options include: • Date (removes Time from ISO 8601 *YYYY-MM-DD* date format) • DayOfMonth (number: *1-31*) • DayOfWeek (number: *0-6*) • DayOfWeekAbbreviation (3 characters: *Mon, Tue, Wed*, etc.) • DayOfWeekName (string: *Sunday, Monday, Tuesday*, etc.) • DayOfYear (number: *1-365*) • FirstDayOfFiscalQuarter (DateTime) *see note below* • FirstDayOfFiscalYear (DateTime) *see note below* • FirstDayOfMonth (number: *1-31*) • FirstDayOfQuarter (DateTime) • FirstDayOfWeek (DateTime) • FirstDayOfYear (DateTime) • FirstHourOfDay (DateTime) • FirstMillisecondOfSecond (number: *1-9999*) • FirstMinuteOfHour (DateTime)

Attribute	Description
	• FirstSecondOfMinute (DateTime) • FiscalQuarter (number: *1-4*) • FiscalYear (number: 4-digit year) *see note below* • Hour (number: *1-12* or *1-24*) • LastDayOfFiscalQuarter (DateTime) • LastDayOfFiscalYear (DateTime) *see note below* • LastDayOfMonth (DateTime) • LastDayOfQuarter (DateTime) • LastDayOfWeek (DateTime) • LastDayOfYear (DateTime) • Minute (number: *0 -59*) • Month (number: *1-12*) • MonthAbbreviation (3 characters: *Jan, Feb, Mar*, etc.) • MonthName (string: *January, February, March*, etc.) • Quarter (number: *1-4*) • Second (number: *0-59*) • Week (number: *1-52*) • Year (number: 4-digit year) This attribute can be set using tokens, such as an @Request token.
Include Condition	If the value of this attribute is left blank or contains a formula that evaluates to *True*, the Time Period column is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer.
Time Period	If set, applies a specific user culture (see discussion below). Default: *en US*

Attribute	Description
Culture	

Fiscal Periods - Results are dependent on the Fiscal-related values set in the _Settings definition using the **Globalization** element (see next section). **DateTime Values** - Options that create DateTime values in the datalayer put them in ISO 8601 date format:

```
YYYY-MM-DDThh:mmTZD (e.g. 1997-07-16T19:20+01:00)
```

Globalization Settings

The value for the first day of the week can be set in the _Settings definition using the **Globalization** element. Its **First Day of Week** attribute can be set to one of these values:

- 0 = Sunday (the default value)
- 1 = Monday
- 2 = Tuesday
- 3 = Wednesday
- 4 = Thursday
- 5 = Friday
- 6 = Saturday

This attribute value will affect the "FirstDay..." and "LastDay..." values the element calculates (except for FiscalQuarter).

Another Globalization attribute that affects calendars is **First Day of Fiscal Year**. Set this attribute to determine when the fiscal year will start, if not on January 1st. This attribute value specifies the starting month and day, in the format *MM/DD*.

The starting point for the "FiscalQuarter..." values the element calculates can be specified by creating a **constant** in the _Settings definition, named FirstDayOfQ1. Its value specifies the month and year, in the format *MM/YY*.

Culture Settings

The Time Period Column element itself can be made **culture-aware**. This can affect the text returned by time periods for **names** and **abbreviations** and the numbers returned for the day of the week. The element's **Time Period Culture** attribute can be set to a specific culture value or configured to use the client browser's culture settings by entering the @Function.UserCulture~ token. The default is the "invariant" culture, which is basically US English.

Here are three examples of **DayOfWeekName** values, as returned by using the default, French, and Spanish culture settings:

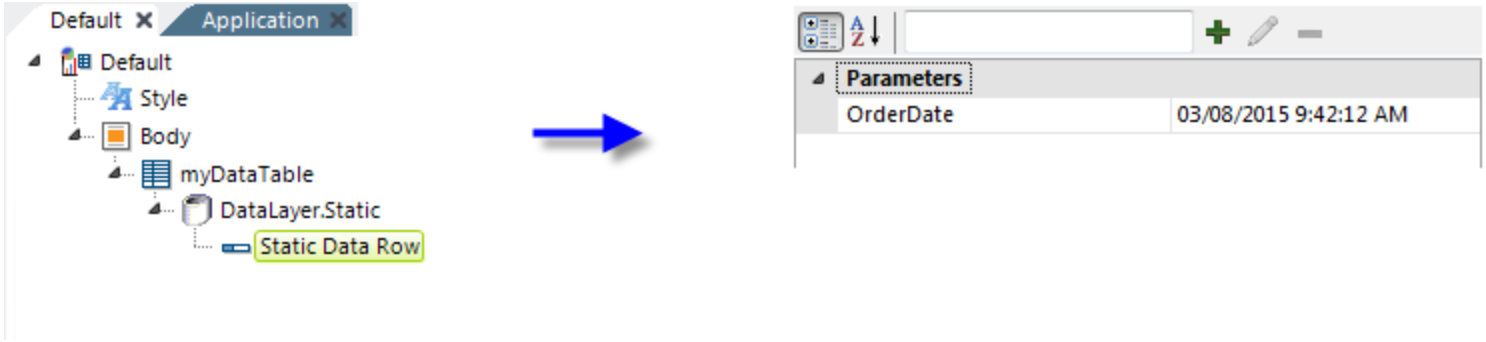
Monday	Tuesday	Wednesday	Thursday	Friday

Lundi	Mardi	Mercredi	Jeudi	Vendredi

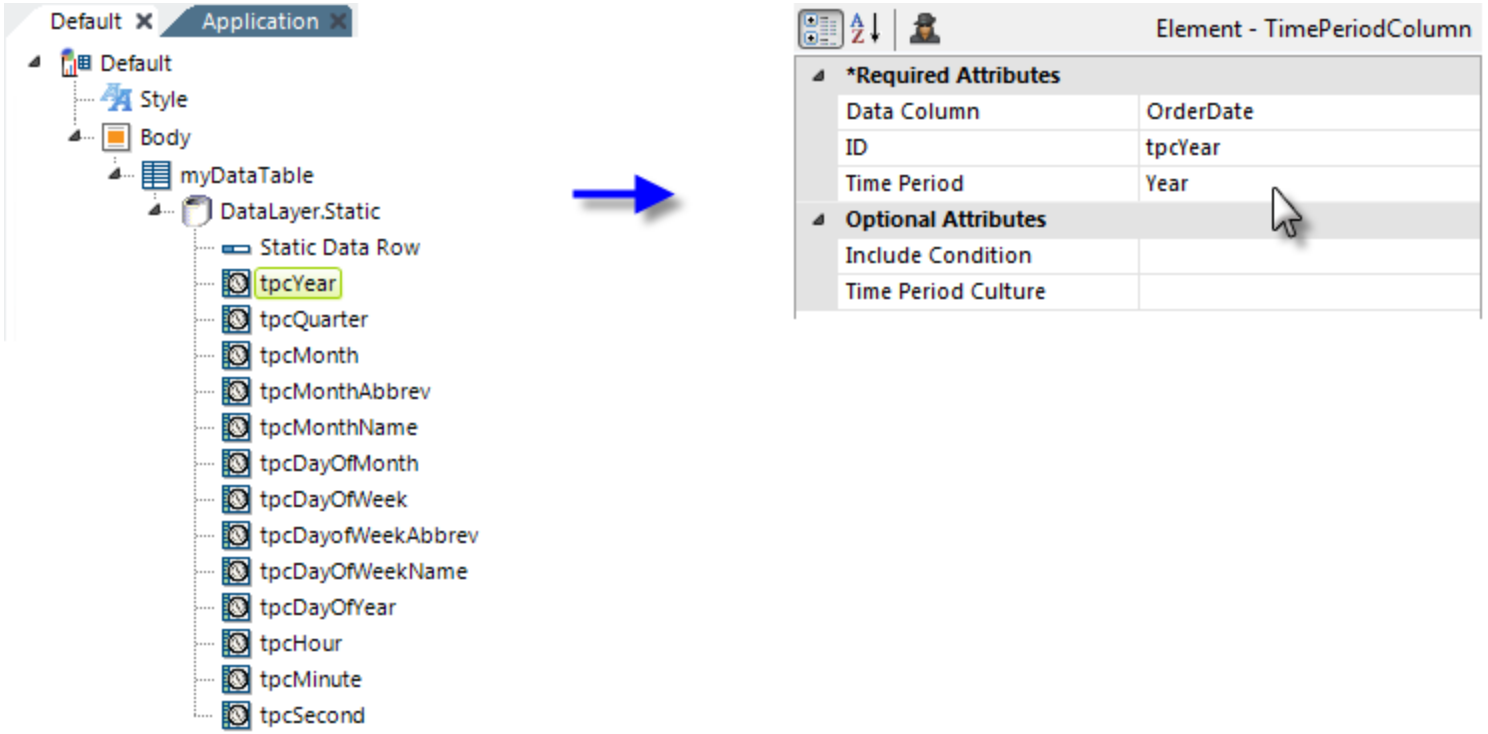
Lunes	Martes	Miércoles	Jueves	Viernes

Time Period Columns Simple Usage Example

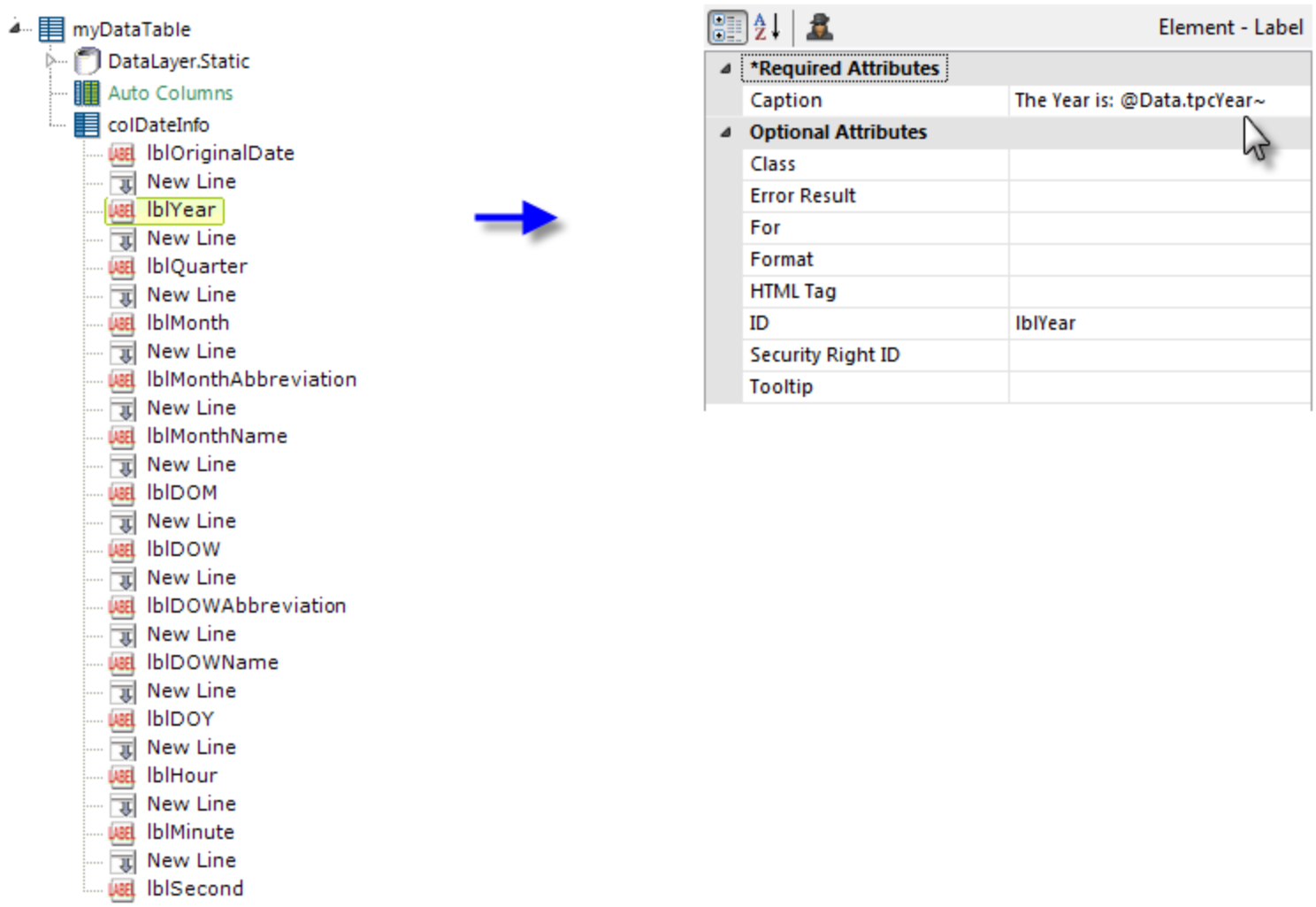
The following example illustrates how the Time Period Column works by displaying all of the possible date and time data parts the element can produce.



In the definition shown above, a **Data Table** and a **DataLayer.Static** element have been added. A single **Static Data Row**, with a single column ("OrderDate") whose value is a date and time, has been added to the datalayer.

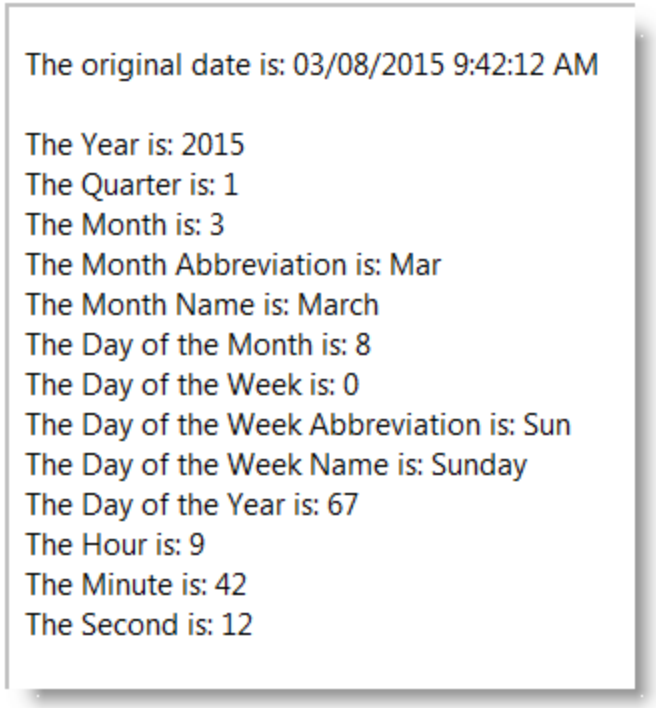


As shown above, a subset of the **Time Period Column** elements available are then added as children of the datalayer. The **Data Column** attribute identifies the datalayer column, in this case "OrderDate", from which the Time Period Column's data will be derived. The **ID** attribute provides an arbitrary name for the new column that will be added to the datalayer and will be used later with an @Data token to reference the derived data. The **Time Period** attribute determines which DateTime information to derive from the data column. The **Time Period Culture** attribute allows the element to derive data in a culture-aware fashion. These attributes are discussed in more detail in "Time Period Columns Attributes" on page 370.

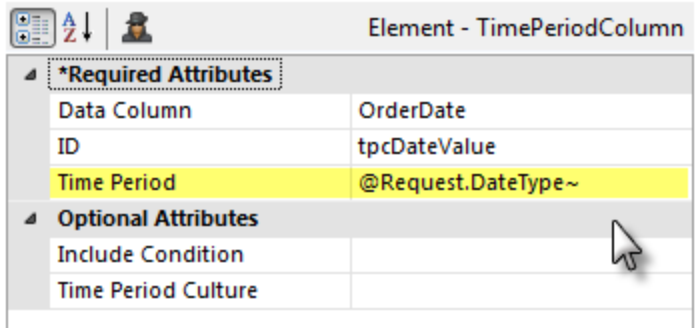


As shown above, a **Data Table Column** element has been added and **Label** elements placed beneath it to display the data in each of the Time Period Columns.

The assigned **ID** of a Time Period Column is used with an @Data token in the same manner as any actual data column. *Logi Info developers attempting to duplicate this example may care to use the Auto Columns element instead of adding a Data Table Column element and all the Label and New Line elements shown.*



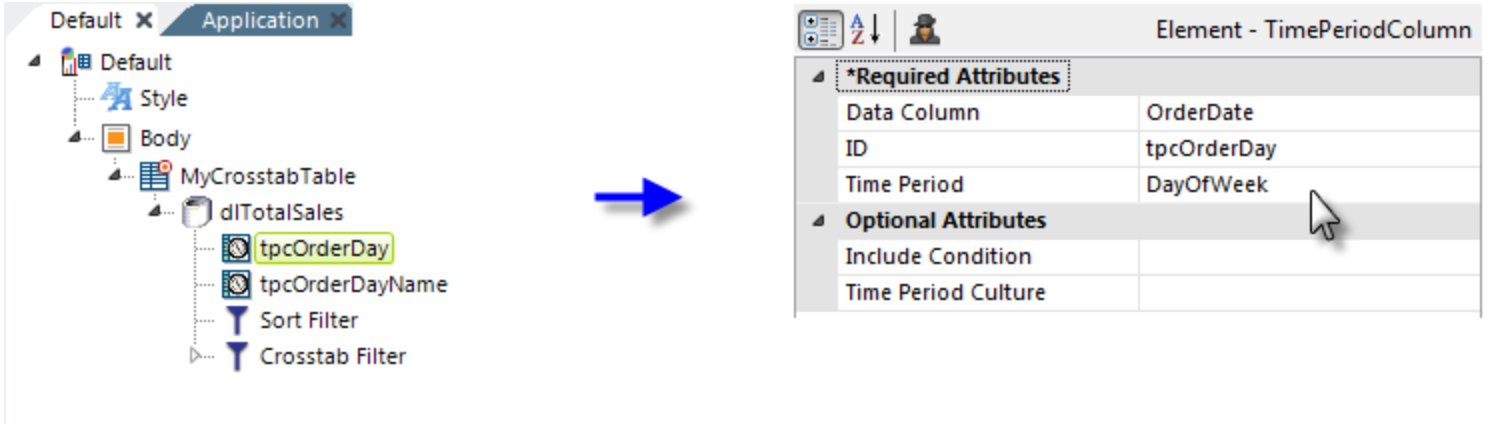
And the results are shown above.



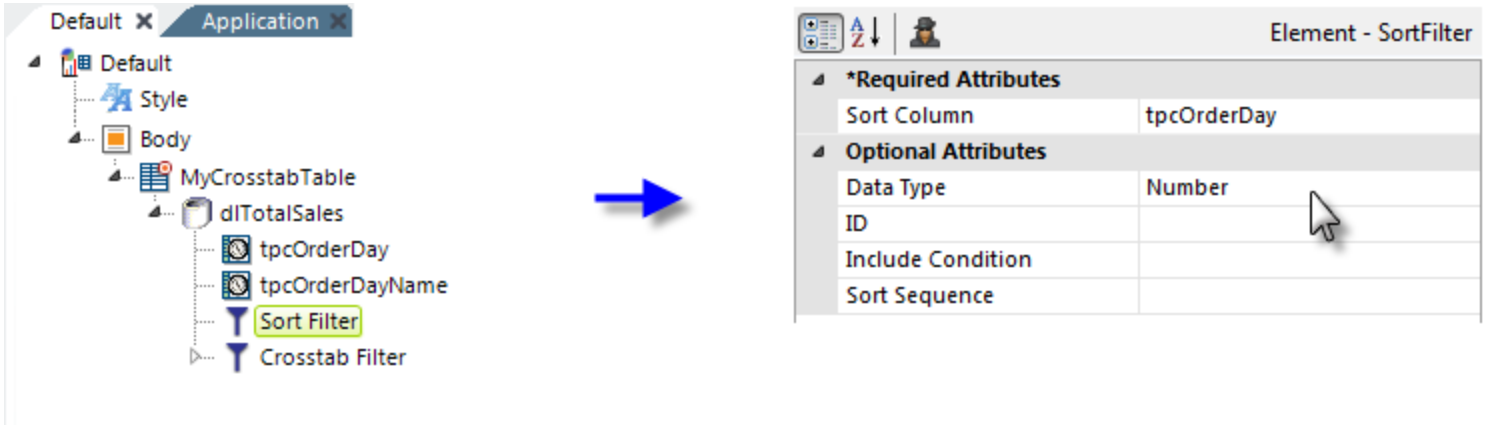
As shown above, the **Time Period** attribute value itself can be set using a token. Consider a situation, for example, where the user can choose options at runtime, from an Input Select List, such as Quarterly, Monthly, or Yearly. If the select list's **onChange** event is used to refresh the page, then the user's choice will be available as an @Request token and this token can be used to set the Time Period attribute value. This allows the content of the Time Period Column to be dynamic based on user input, which can be very useful in controlling data to be displayed.

Time Period Columns Advanced Usage Example

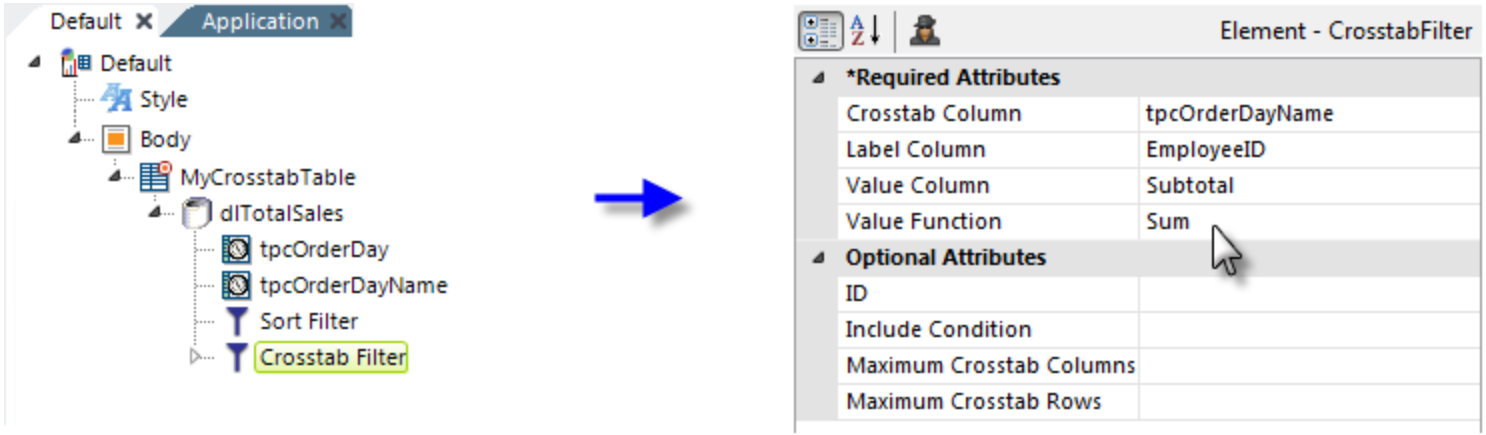
Time Period Columns can be really useful in providing **data** for use by **other** elements. The following example of a Crosstab Table uses them to provide data for the necessary sorting and crosstab filters.



In the example definition above, a **Crosstab Table** has been added and its datalayer's SQL query returns data including a column named OrderDate. Two **Time Period Column** elements have been added which create new columns in the datalayer based on OrderDate. The example above shows one of these ("OrderDay") set to derive the day of the week, a number, from OrderDate.



The datalayer's **Sort Filter** element can now use the new OrderDay column as its sort column, as shown above. This ensures the data for the report will be sorted in day-of-the-week order.



The datalayer's **Crosstab Filter** element is configured, as shown above, to use the new OrderDayName column as its crosstab column. This ensures the crosstab columns will be arranged one per day of the week.

ID	Employee Name	Sunday	Monday	Tuesday	Thursday	Friday	Saturday	Sum	Average	Count
1	Davolio, Nancy	$28,115.75	$35,280.42	$20,140.40	$45,431.56	$14,342.29	$43,966.96	$187,277.38	$31,212.90	120
2	Fuller, Andrew	$33,982.25	$30,390.14	$15,924.93	$34,877.67	$26,284.21	$21,310.58	$162,769.78	$27,128.30	93
3	Leverling, Janet	$36,607.38	$17,892.31	$4,908.35	$35,849.34	$52,564.10	$54,991.36	$202,812.84	$33,802.14	127
4	Peacock, Margaret	$56,192.13	$57,380.89	$4,401.16	$31,752.91	$53,374.31	$22,662.30	$225,763.70	$37,627.28	151

For the sake of brevity, the additional elements used to display the crosstab data are not shown here, but the first few rows of the resulting table are shown above. In conclusion, the **Time Period Column** element can create derived columns in a datalayer that are immediately available for use in other elements for purposes such as filtering and grouping.

The UnCrosstab Filter

The **UnCrosstab Filter** element is used with datalayer elements to "un-pivot" or "reverse pivot" rows of data into columns of data. It converts one row of multi-column data into multiple rows, each with a single data value.

The following topics discuss the UnCrosstab Filter element:

- Applying the UnCrosstab Filter
- Using Dynamic UnCrosstab Filters

About the UnCrosstab Filter

The **UnCrosstab Filter** element is available for use with all datalayer elements and is the functional opposite of the Crosstab Filter element. In operation, the element basically converts each row in the datalayer, each of which has multiple columns, into multiple rows, each of which has one value column.

Before filter is applied:

Customer ID	Company	Aniseed Syrup	Chartreuse verte	Escargots	Flotemysost	Lakkalikööri	Frankfurter	Raclette Courdavault	Rössle Sauerkraut	Spegesild
ALFKI	Alfreds Futterkiste	6	21	40	20	15	2	15	17	2

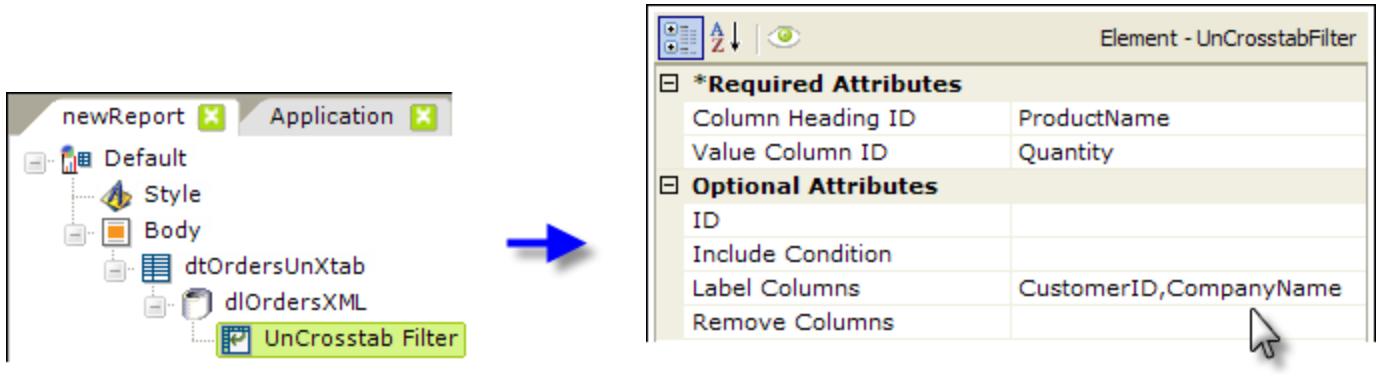
After filter is applied:

Customer ID	Company	Product	Quantity
ALFKI	Alfreds Futterkiste	Aniseed Syrup	6
ALFKI	Alfreds Futterkiste	Chartreuse verte	21
ALFKI	Alfreds Futterkiste	Escargots	40
ALFKI	Alfreds Futterkiste	Flotemysost	20
ALFKI	Alfreds Futterkiste	Lakkalikööri	15
ALFKI	Alfreds Futterkiste	Frankfurter	2
ALFKI	Alfreds Futterkiste	Raclette Courdavault	15
ALFKI	Alfreds Futterkiste	Rössle Sauerkraut	17
ALFKI	Alfreds Futterkiste	Spegesild	2

It's a little easier to understand the filter's actions when you can see the effect limited to one row, in a Data Table, as shown above. This "un-pivot" action normalizes the data on the fly and, among other applications, can be really useful when having to work with data imported from spreadsheets.

Applying the UnCrosstab Filter

The following example illustrates how the **UnCrosstab Filter** is used:



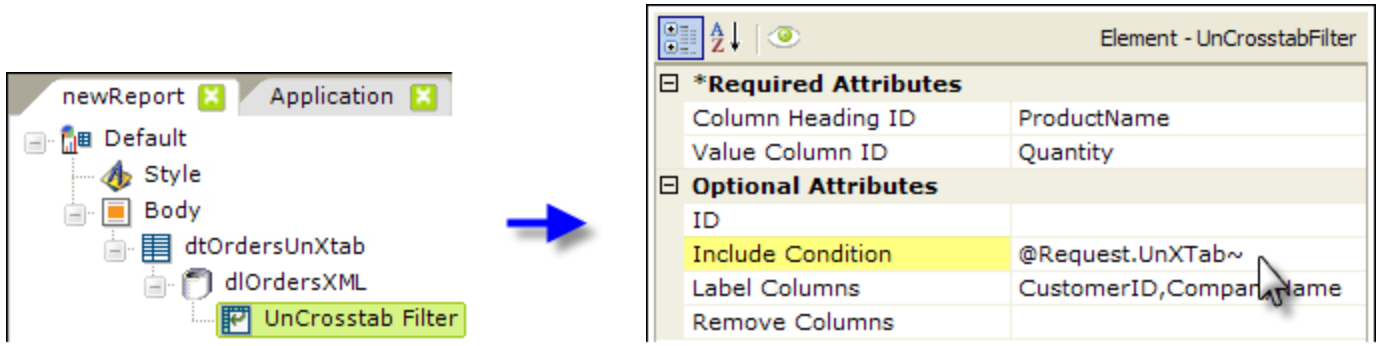
1. As shown above, an **UnCrosstab Filter** element is added as a child beneatha datalayer element.
2. Its **Column Heading ID** attribute value is set to the column used to provide the original pivoted column headings.
3. The **Value Column ID** attribute is set to name of a new column that will receive the data values.
4. The optional **Label Columns** attribute identifies one or more columns, in a comma-separated list, that are not pivoted.

5. The optional **Remove Columns** attribute identifies one or more columns, in a comma-separated list, that should not be included in the output.

In the complete example above, the sequence of events is that the datalayer reads in all the data from the XML data file, then the UnCrosstab Filter builds a substitute datalayer by examining each row in the source datalayer and adding rows to the substitute datalayer for each value column found.

Using Dynamic UnCrosstab Filters

The UnCrosstab Filter element also has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows developers to dynamically determine if the datalayer will be altered or not.

The Unit Conversion Column

The **Unit Conversion Column** element is used with datalayer elements to convert values from one standard measurement to another.

The following topics discuss the Unit Conversion Column element:

- Applying the Unit Conversion Column
- Using Multiple and Dynamic Unit Conversion Columns
- Customizing the Conversion Factors

About the Unit Conversion Column

The Unit Conversion Column element is available for use with all datalayer elements. In operation, the element basically performs a mathematic conversion on the specified data column value and places the result in a new column that's added to the datalayer. Here's an example showing the data before and after the Unit Conversion Column is applied, with settings for Temperature conversion:

City	Celsius
Stockholm	0
London	8
Berlin	12
Rome	18

Before

City	Celsius	Fahrenheit
Stockholm	0	32
London	8	46.4
Berlin	12	53.6
Rome	18	64.4

After

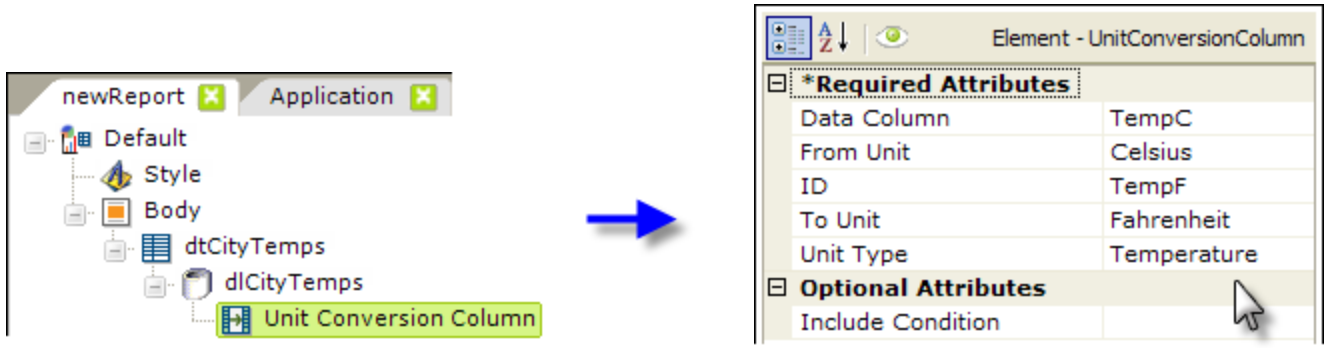
Conversion factors are provided for a variety of measurement categories, including:

• Area • Energy • Fuel Consumption • Length • Pressure	• Temperature • Velocity • Volume • Weight

The actual conversion factors are stored in a file and developers who wish to add their own custom factors can do so. See "Customizing the Conversion Factors" on page 391 for more information about customization.

Applying the Unit Conversion Column

The following example illustrates how the **Unit Conversion Column** is used:



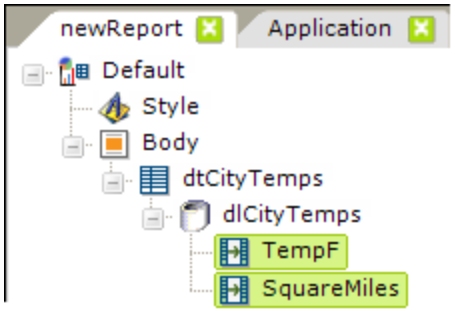
All of the attributes mentioned below are required:

1. As shown above, a **Unit Conversion Column** element is added as a child to a datalayer element.
2. Its **Data Column** attribute value is set to the name of the column in that datalayer whose value will be converted.
3. The **Unit Type** attribute is set to the standard category name for the data units. It's important to do this *before* setting the other attributes.
4. The **From Unit** attribute is set from the pull-down list of optionsto the unit of the data in the named Data Column.
5. The **ID** attribute sets the name of the new column that will be added to the datalayer, containing the converted value.
6. The **To Unit** attribute is set from the pull-down list of optionsto the unit of the data that will be output to the new column.

Using Multiple and Dynamic Unit Conversion Columns

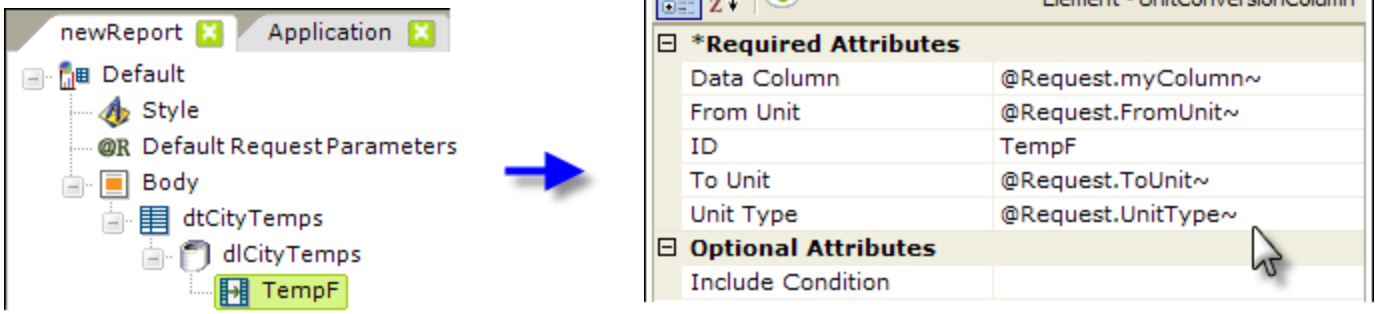
This topic introduces how to use multiple and dynamic Unit Conversion Columns.

1. Developers can use two or more **consecutive** Unit Conversion Column elements:



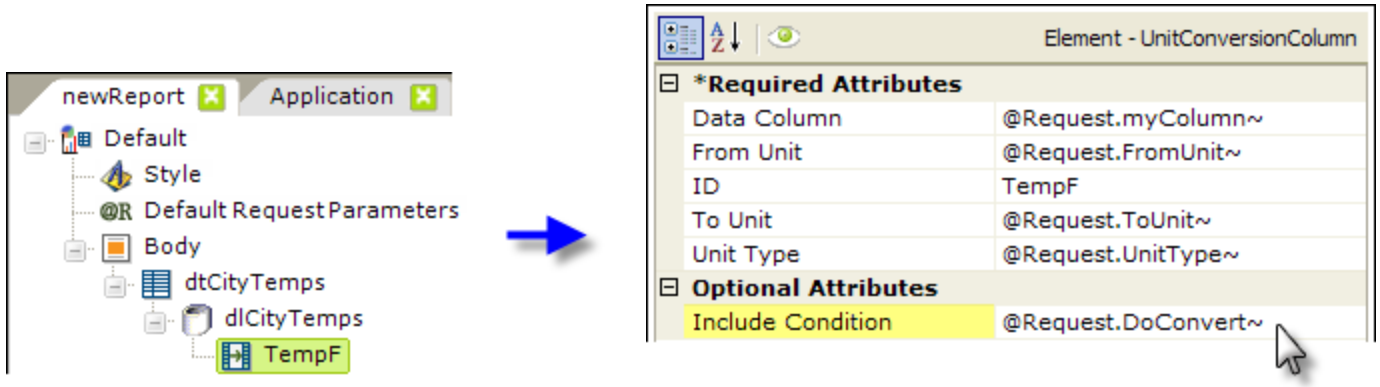
In this case, the conversions will be conducted **sequentially**, based on their arrangement from top to bottom in the element tree. So, after the first Unit Conversion Column is applied, the second Unit Conversion Column will be applied.

2. Developers can make Unit Conversion Column elements **dynamic** by using some tokens:



As shown above, all attributes other than the ID attribute, can use @Request, @Session, and @Constant tokens to make their values dynamic at runtime. Due to the order of processing operations, @Data tokens *cannot* be used here. Sometimes it is necessary to ensure that tokens have default values when running a report. When using @Request tokens, the **Default Request Params** element can be used to create default values for each **@Request** token.

3. The Unit Conversion Column element also has an **Include Condition** attribute:



If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows developers to dynamically determine if the datalayer will be filtered or not.

Customizing the Conversion Factors

The actual conversion factors are stored in the file:

```
<yourLogiApp>\rdTemplate\rdDataEngine\rdUnitConversion.xml
```

Developers who wish to add their own custom factors can do so by copying the file into the _SupportFiles folder and making changes to it there. The application will look in that folder first for the file. The XML data in the file consists ofmanyentries, including these three, which we'll use to explain the conversion factor requirements: `<Unit Name="Celsius" Scale="1" Shift="0" UnitType="Temperature" />`

`<Unit Name="Fahrenheit" Scale="1.8" Shift="32" UnitType="Temperature" />`

`<Unit Name="Kelvin" Scale="1" Shift="273.15" UnitType="Temperature" />`

- The **Name** XML attributedefines the name of the conversion, which will be used in the element's To Unit and From Unit attributes.
- The **UnitType** XML attributedefines the measurement category and groups related conversion factors together.

  Each unit type group begins with a "base unit" entry ("Celsius" in the example above) and all following conversion factors in the same group are calculated against it.

- The **Scale** XML attributeindicates the *multiplication* factor to be applied to the data column value during conversion, while the **Shift** XML attributeindicates the offset to be *added* to the result of the multiplication during conversion.

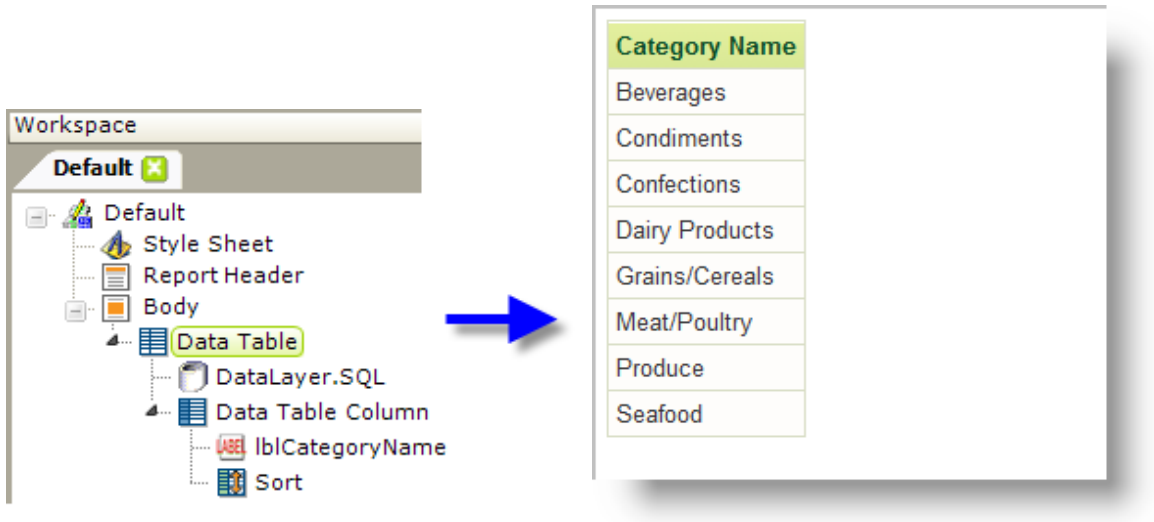
  For example, if we have a data column with a value of 8-degrees Celsius, the conversion applied for Fahrenheit is:

  ```
  ( Data Value * Scale Factor ) + Shift Factoror ( 8 * 1.8 ) + 32 = 46.4
  ```

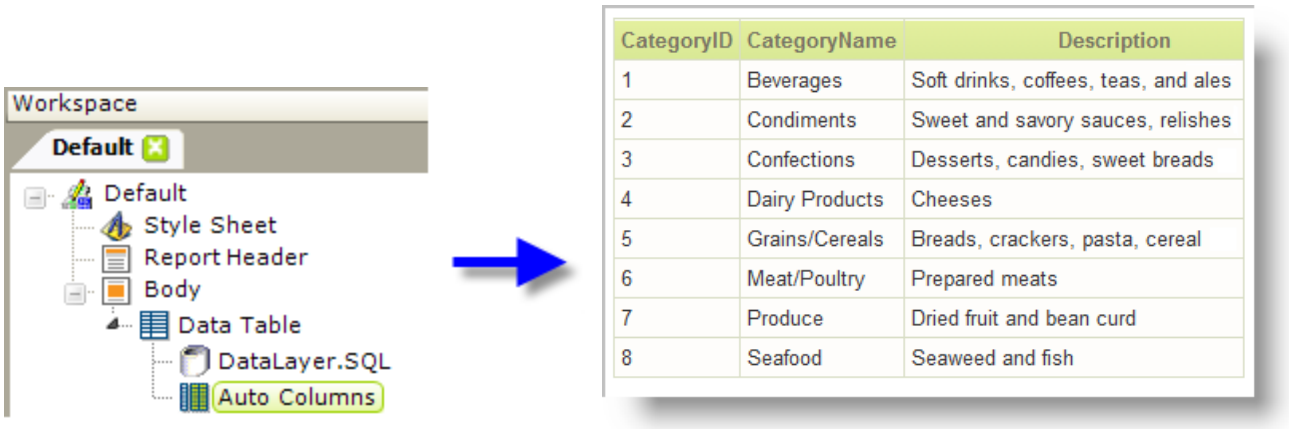
The use of tokens is not supported in the conversion factors file.

Auto Columns

The **Auto Columns** element provides an easy method of **viewing all datalayer columns** without having to know their names and without loading up a definition with Data Table Column elements. This topic discusses the use of the Auto Columns element.



The example above shows a report definition containing a table with **one column** and the report output.



CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, and ales
2	Condiments	Sweet and savory sauces, relishes
3	Confections	Desserts, candies, sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

However, when an **Auto Columns** element is used instead of (or with) Data Table Column elements, as shown above, the resulting report output will contain *all* of the columns. The column header text of these "automatic" columns will match the names of the columns in the datalayer. And if you combine Auto Columns with Data Table Column elements, the automatic columns will always appear to the *right* of any specified using Data Table Column elements.

The Auto Columns element has **no attributes** and there are **no data formatting** options.

Auto Columns reads the first row of datalayer data to identify the columns in the data set. If the first row doesn't contain *all* of the columns, for example if the Flattener element has been used on JSON data, then the results will seem incomplete.

How Is This Useful?

Where can this element be used to good effect? Though we don't discount the resourcefulness of developers in making use of this element in production, it's designed for use in **development** or **diagnosticsituations**, such as:

1. Including an Auto Columns element and remarking/unremarking it during development can provide a quick way to ensure that the datalayer is actually *receiving* the *expected data*.
2. The Auto Columns functionality is useful in situations where the *count* and *names of columns* returned to a datalayer vary dynamically based on a query or situation. For example, imagine that a web service your application uses suddenly changes the results that it returns; use of Auto Columns can help you discover the change.
3. Consider this query: `SELECT * FROM @Request.TableName~`. In this case, you may not be able to hard-code the columns in the result as elements in a report definition because the *table name varies* and so, possibly, will the column names.


Should you attempt to use Auto Columns in a production situation, this element is processed at *a different time* by the Logi engine than other elements, which can result in no data being shown. For example, if you try to use Auto Columns as a child of More Info Row, no data will be displayed when the More Info Row is expanded.

Action Elements

Action elements are an essential part of Logi reporting and the primary mechanism for providing HTML links in a definition. Different types of Action elements are available for developers to use in specific situations. Developers will find them easy to use; they save a lot of time when creating reports.

The following topics provide examples of how to use the Action Element:

- Example: Link to Another Report
- Example: Link to a URL
- Example: Link that Executes JavaScript
- Example: Export to PDF
- Example: Email Report as Attachment
- Example: Showing and Hiding Elements
- Example: With an Event Handler
- Example: With Event Handler for Popup Menu

About Action Elements

The following table provides a description of the available Action elements. The links connect to the Element Reference page or a relevant topic for each element.

Element	Description
Action.Add Bookmark	Allows users to add a new bookmark for the current report, saving the request parameters so the report can be re-run later with the same input values. For more information, see *Bookmarks*.
Action.Add Dash-	At runtime, allows user to add a new Dashboard panel, containing specified content from one report, in

Element	Description
board Panel	Dashboard in another report.
Action.Copy Bookmark	Allows bookmarks to be copied and added to a user's bookmark collection.
Action.Dial Phone	Launches a mobile device's phone application with a phone number. The user still needs to click a "Call" button to place the call. *Mobile Report definitions only.*
Action.Draft Email	Launches a mobile device's email application. The user still needs to click a "Send" button, and possibly enter Addressees, Subject and email Body text. *Mobile Report definitions only.*
Action.Draft Text Message	Launches a mobile device's SMS text messaging application with pre-filled phone number. The user still needs to enter the message text and click a "Send" button. *Mobile Report definitions only.*
Action.Drag Bookmark	Allows bookmarks to be dragged into the folders of the **Bookmark Organizer** element in the same Report definition. For more information, see Organizing Bookmarks.
Action.Edit Bookmark	Displays a pop-up text box that allows the user to change a bookmark's Description. For more information, see Creating a Bookmark Manager.
Action.Email Report	Sends a report as an email attachment. When the link is clicked, a dialog box is displayed so that To, From, Subject, and Message can be entered.
Action.Exit	Abandons the users session and takes him to another URL.

Element	Description
Action.Export CSV	Exports the report's Data Table data into a comma-delimited CSV file.
Action.Export Native Excel	Runs a report for export as a native Excel document.
Action.Export Native Word	Runs a report for export as a native Word document.
Action.Export PDF	Runs a report for export as a PDF document.
Action.Export Word or Excel	Runs a report so that it can be exported to Word or Excel.
Action.Export XML	Returns the XML generated by the target report's datalayers.
Action.Google Spreadsheet	Runs a report for export as a Google Spreadsheet document.
Action.Javascript	Runs JavaScript in the user's browser, which can also call JavaScript functions defined in script files with the Script File attribute and AdditionalScriptFiles element.
Action.Link	Together with Target.Link, runs a specific URL or executes embedded JavaScript.
Action.Map Location	Runs a Google Map query on a mobile device to show a map with one or more locations highlighted. *Mobile Report definitions only.*

Element	Description
Action.Map Marker Info	Used with a Google Map; displays an information panel when the user clicks on a map marker. *Logi Info only.*
Action.Popup Menu	Displays a pop-up menu offering various options to the user.
Action.Pre-Action Javascript	Runs client-side JavaScript *before* its parent Action element executes, enabling custom code to be run prior to the action occurring. The parent Action element will only be executed if custom JavaScript returns a value of *true*. For more information, see Scripting.
Action.Process	Executes a task in a Process definition. *Logi Info only.*
Action.Refresh Element	Refreshes one or more elements in the report, using AJAX technology so that the entire report page is not necessarily redrawn.
Action.Remove Bookmark	Allows the user to delete a bookmark. For more information, see Creating a Bookmark Manager.
Action.Report	Runs a report definition, in the same window, another window, or within an IncludeFrame.
Action.Run Bookmark	Runs a report while applying the parameters of a bookmark to it. For more information, see Bookmarks.
Action.Show Bookmark Sharing	Displays a pop-up panel for adding, listing, and removing users who those who may share a bookmark. For more information, see Sharing Bookmarks.
Action.Show Ele-	Shows or hides one or more elements on the current page, using Show Modes.

Element	Description
ment	
Action.Template	Runs a Logi Template definition.
Action.Widget	Loads a Logi widget.

Actions are usually attached to a parent element, such as a **Label** or **Image** element. The parent element controls when and how the action is triggered. For example, a Label element can be combined with an Action.Link element to create a hyperlink to a web site. Action elements make use of Target sub-elements when linking to another page or site. The Target element sets what page is to be loaded and how it will be displayed.


You can only use *one* Action element directly under a user input element, Label, or Image. Additional Action elements will be ignored. You can use multiple **Event Handlers**, each with its own Action element. See "Action Elements Example: With an Event Handler" on page 414 for more information.

The Action.Refresh Element

The **Action.Refresh Element** element uses AJAX technology to refresh specific elements or portions of a report page without reloading the entire page, which produces a smooth-looking update. However, developers should keep these guidelines in mind when using this element:

- The best practice is to refresh a **Division** or another container element (and thereby its contents) whenever possible.
- Don't attempt to refresh super-elements or Google Maps, which contain possibly-conflicting internal AJAX calls.

- Local Data datalayers are *not* re-run with all AJAX refresh requests. They are only re-run when the element being refreshed, using Action.Refresh Element, contains either a DataLayer.Linked element linked to the Local Data datalayer, or when it contains an @Local token.
- All "LoadDefinition" plug-in calls will be re-run with each refresh request.
- When refreshing a visualization that you then want to export and send via email, be sure to include the *container* (usually a Div) for the Action.Email Report and Export elements in the list of elements Action.Refresh Element will refresh.
- You can refresh multiple elements at once by listing their IDs in the **Element ID** attribute, separated by commas.
- You cannot refresh **Responsive Row** and **Responsive Column** elements directly by specifying their IDs in an Action.Refresh Element element. However, you can enclose the entire table in a Division and refresh it, or place the Responsive Column's child elements in a Division and refresh it.
- The Action.Refresh Element element's **Element ID** attribute supports tokens.
- The element's **Post Refresh Javascript** attribute allows you to call custom browser JavaScript *after* Action.Refresh Element has completed updating the page.

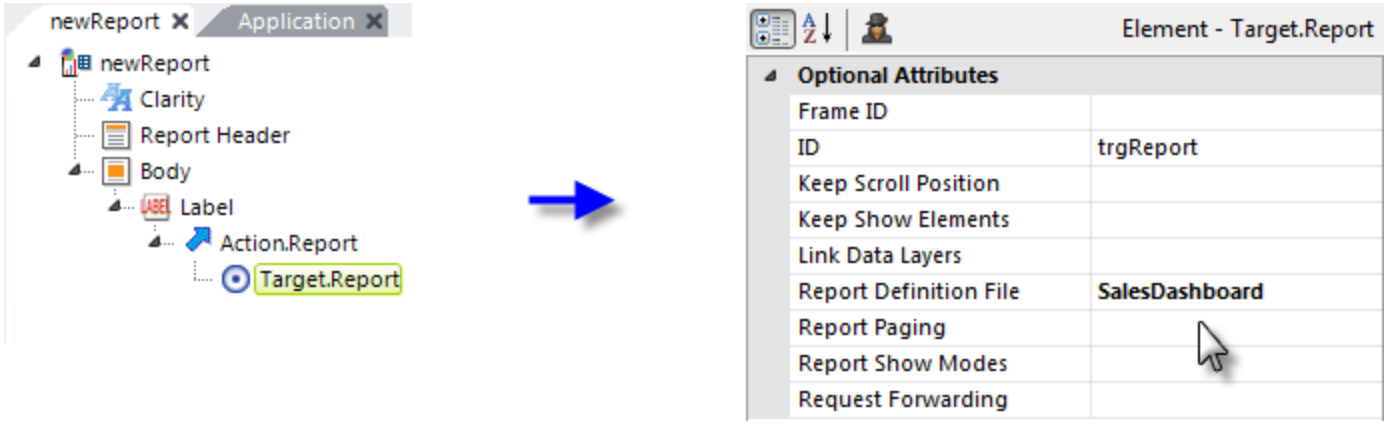
Using the Wait Panel

The **Wait Panel** element (which was called the Wait Page at one time) displays a modal "busy" icon when an action takes more than a few seconds, letting users know processing is in progress. Simply adding the Wait Panel as a child of the following Action elements makes it work:

- Action.Link
- Action.Process
- Action.Refresh Element
- Action.Run Bookmark

Action Elements Example: Link to Another Report

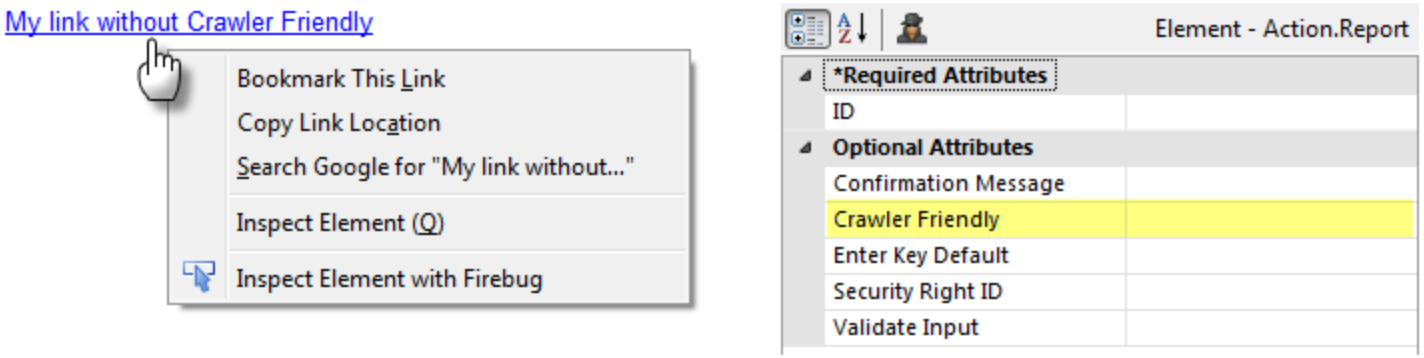
This example shows a simple link that launches another Logi report in the same application:

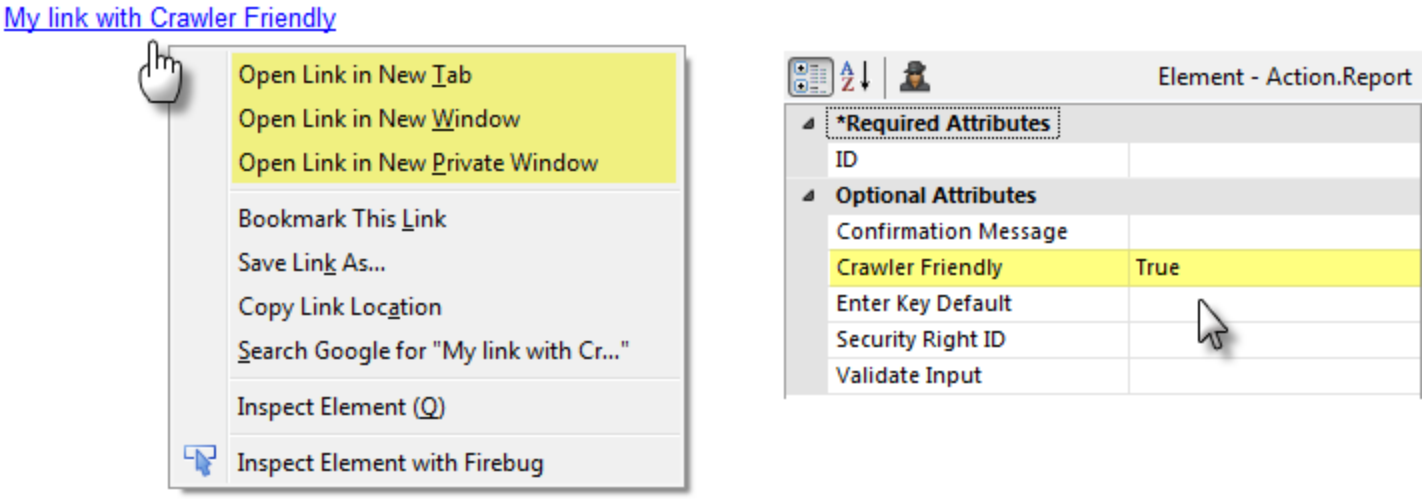


1. Add a **Label** element to your report and set its **Caption** attribute value as desired.
2. Beneath it, add an **Action.Report** element. Its default attribute settings are used for this example.
3. Beneath it, add a **Target.Report** element. Set its **Report Definition File** attribute to the definition file of the desired report.

   In order to reload the current report definition, DO NOT set this attribute value to *CurrentReport* if you want Input element values or Link Parameters to be passed, or the data to be refreshed. Enter the actual report definition name instead.

The Label caption will appear on the page as an HTML link; when clicked, the second report will be displayed. Action.Report can be used as a child of a number of elements, including Button, Image, Charts, and more.



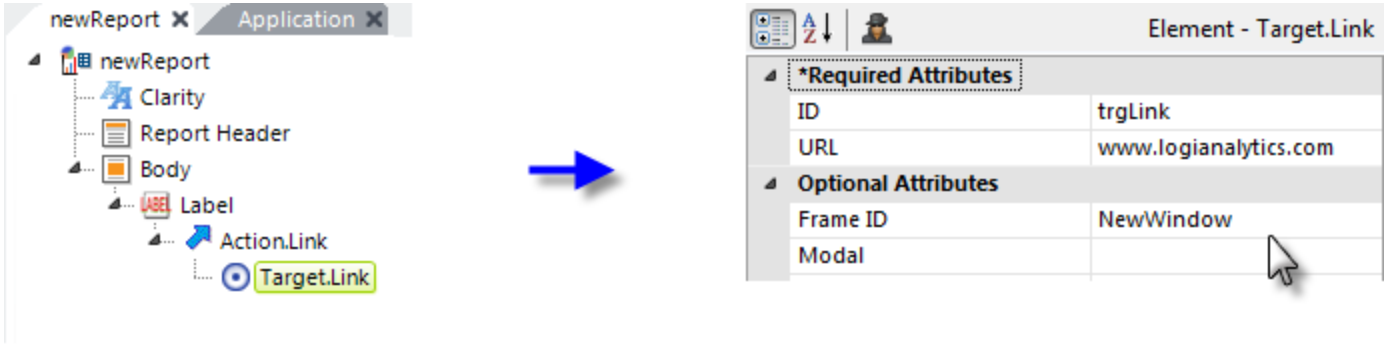


At runtime, links generated using Action.Report, by default, use internal JavaScript to link to other definitions. However, these links may go unnoticed by search robots and may cause unwanted browser behavior. For example, as shown above top, the default Action.Report settings produce the context menu shown when *right-clicked* in the browser. If you want users to be able to open the link target in a new browser tab or window, be sure to set the Action.Report element's **Crawler Friendly** attribute to *True*, as shown in the bottom example. This ensures that a standard HTML `<a>` tag set is generated for the link. Also use this value

when you want search robots to find the link. The appearance of the link can be controlled by the usual style classes associated with Anchors. For a discussion of methods of passing data to the next report when the link is clicked, see *Pass Information*.

Action Elements Example: Link to a URL

This example shows a simple link that opens another web page, in a new browser window:



1. Add a **Label** element to your report and set its **Caption** attribute value as desired.
2. Beneath it, add an **Action.Link** element. Its default attribute settings are used for this example.
3. Beneath it, add a **Target.Link** element. Set its **URL** attribute to the URL of the target web site or page (include the `"http://"` prefix).

   Other Target.Link attributes control aspects of the target, such as whether it's modal or will include scrollbars. The **Frame ID** attribute determines if the web page will open in the same browser window or tab (the default) or in other windows or tabs. If the latter, some of the behavior (window vs. tab) may be governed by your browser settings.

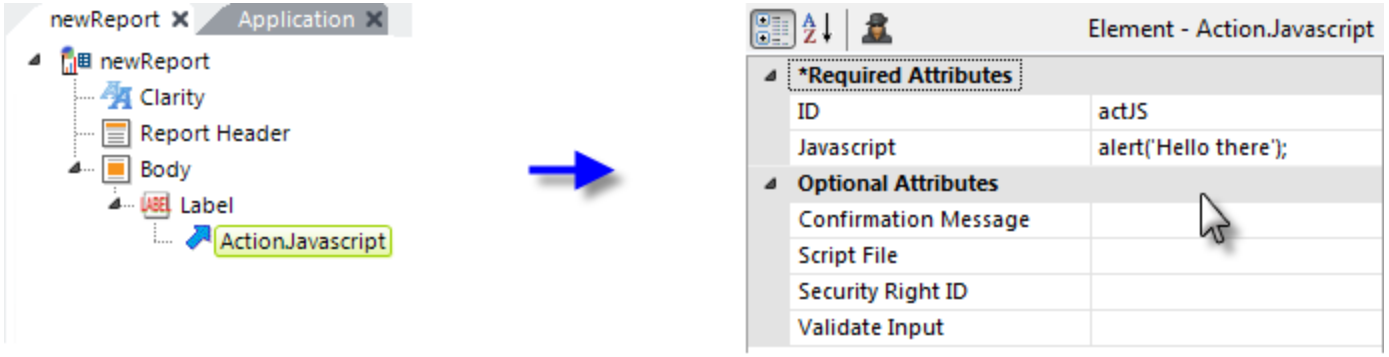
The Label caption will appear on the page as an HTML link; when clicked, the second web page or file will be displayed. Action.Link can be used as a child of a number of elements, including Button, Image, Charts, and more.

Opening a File

If the URL ends in a file name and extension, such as `http://devnet.logianalytics.com/downloads/StudioKeybdShortcuts.pdf`, the browser will attempt to download and open the file for display. In some cases, you may be prompted to either Save or Open the file. If the file type is unusual (for example, it does not have not a common extension like .pdf, .docx, .txt, etc.) you may need to add the file type to your Logi application's supported MIME Types in your web server. For IIS, this is accomplished using the IIS Manager application and is described here. MIME types tell the browser how to handle the file when it downloads it.

Action Elements Example: Link that Executes JavaScript

This example shows a simple link that executes some JavaScript:



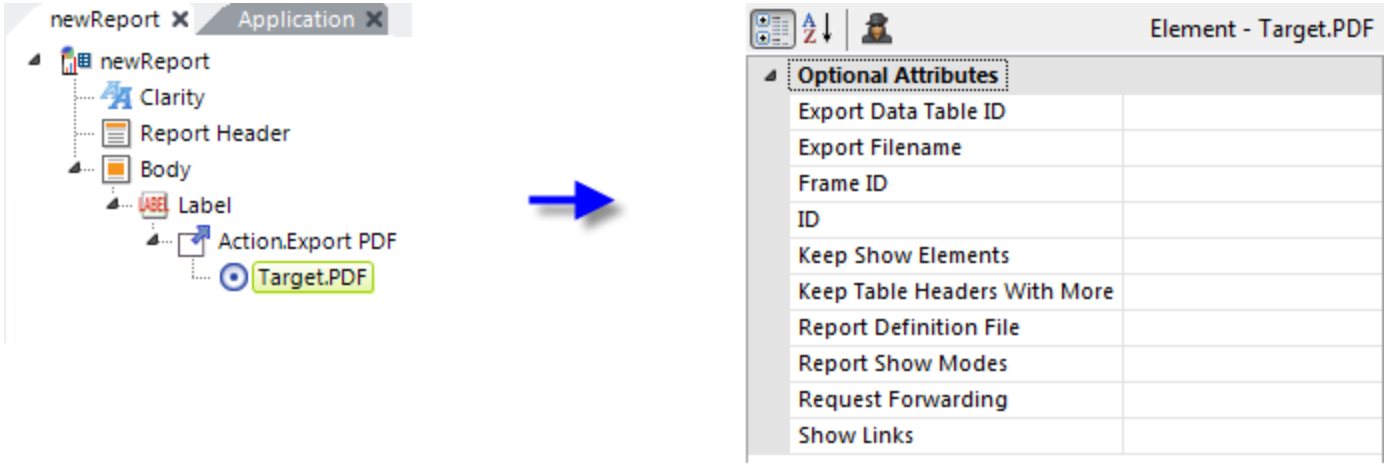
1. Add a **Label** element to your report and set its **Caption** attribute value as desired.
2. Beneath it, add an **Action.Javascript** element. Set its **JavaScript** attribute to the code you wish to run. You may enter multi-line JavaScript code, set off by curly brackets if necessary, and all the traditional functions are supported.

   Double-click the JavaScript attribute name to open the Attribute Zoom window for easier code editing. For more information, see *Using Logi 12 Studio*.

The Label caption will appear on the page as an HTML link; when clicked, the embedded JavaScript will be executed on the browser. Action.Link and Action.JavaScript may be used as the child of a number of elements, including Label, Button, Image, Charts, and more. The **Action.JavaScript** element provides the easiest way to enter JavaScript code. However, the same result can be achieved by using Action.Link and Target.Link elements. Enter the JavaScript code, prefaced with "javascript" followed by a colon, in the Target.Link element's **URL** attribute.

Action Elements Example: Export to PDF

This example shows a simple link that exports the current report to a PDF and is typical of the Action.Export-type elements:

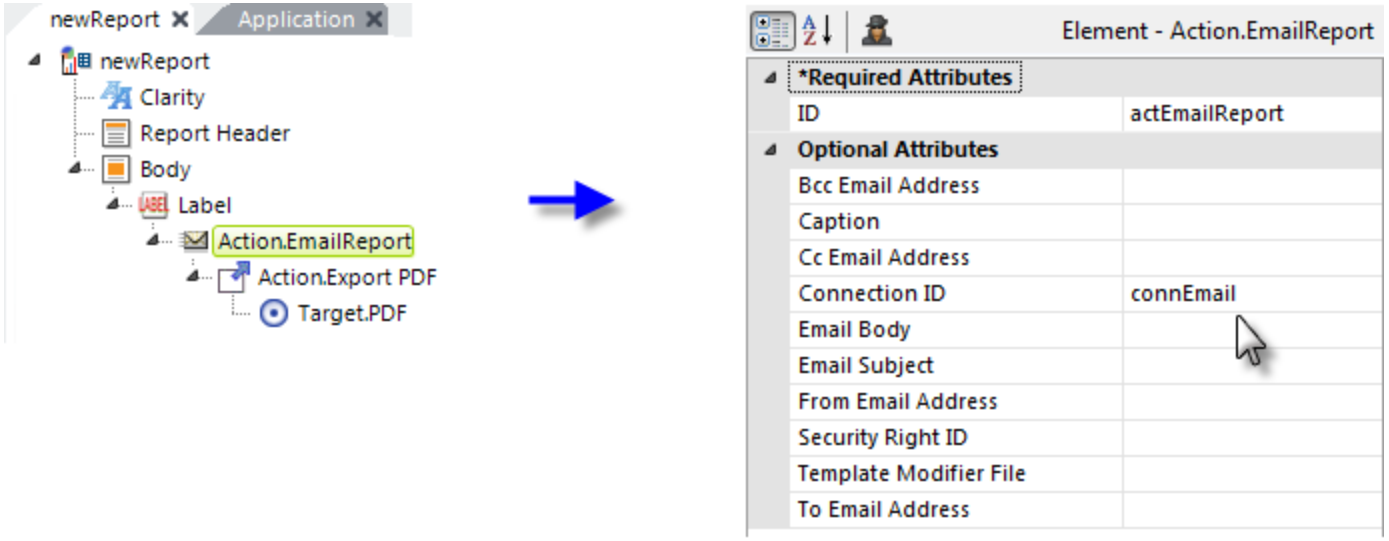


1. Add a **Label** element to your report and set its **Caption** attribute value as desired.
2. Beneath it, add an **Action.Export PDF** element. Its default attribute settings are used for this example.
3. Beneath it, add a **Target.PDF** element. By leaving its **Report Definition File** attribute blank, the default (the current report) will be exported.

Set the **Show Links** attribute to *True* if you want links exported from the report to be "clickable" in the PDF. The Label caption will appear on the page as an HTML link; when clicked, the report will be exported. Action.Export PDF can be used as a child of a number of elements, including Button, Image, Charts, and more. For more information about exporting to PDF, see our *Export to Adobe PDF*.

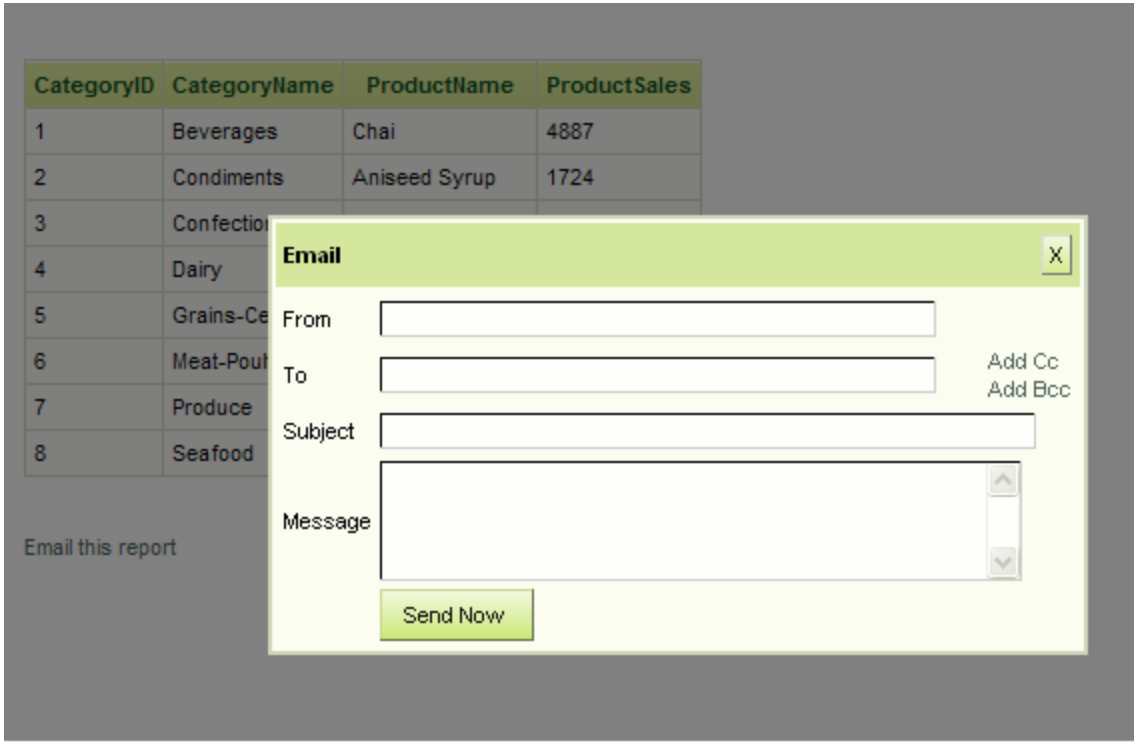
Action Elements Example: Email Report as Attachment

This example shows a link that uses **Action.Email Report** to send the current report as a PDF attachment.



1. Add a **Label** element to your report and set its **Caption** attribute value as desired.
2. Beneath it, add an **Action.Email Report** element. Set its **ID** and **Connection ID** attributes appropriately.
3. Beneath it, add **Action.PDF** and **Target.PDF** elements. By leaving Target.PDF's **Report Definition File** attribute blank, the default (the current report) will be attached. Other export formats are also supported.

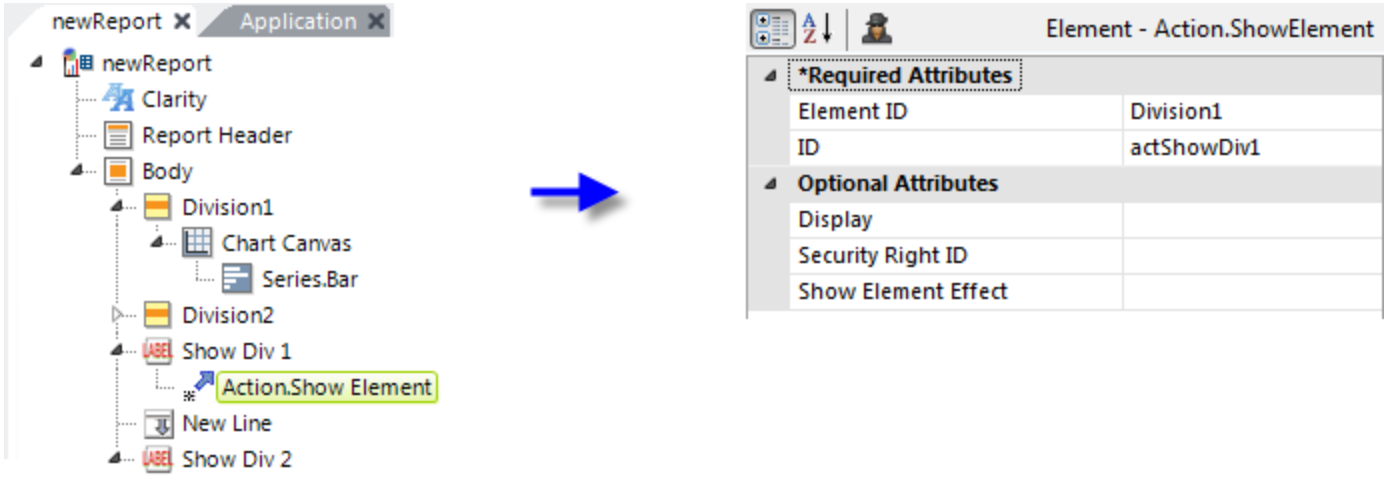
When the report is run and the link is clicked, a modal pop-up panel will be displayed:



Users can then enter email information. Multiple addresses can be entered in the "To" field by separating them with commas or semi-colons. If you provide values for other attributes, such as From Email Address, those values will appear in the pop-up panel as default values for the user input controls. When **Send Now** is clicked, the email(s) will sent out, with the exported report attached as a PDF. If it's supported by your browser, the Action.Email Report element will use an **HTML5** feature called "local storage" to automatically store the email addresses you enter, preserve them between sessions, and provide them back to you in a list for selection in subsequent sessions. If your browser doesn't support this feature, or if it's turned off in your browser options, obviously the element won't be able to provide the list of addresses for re-use.

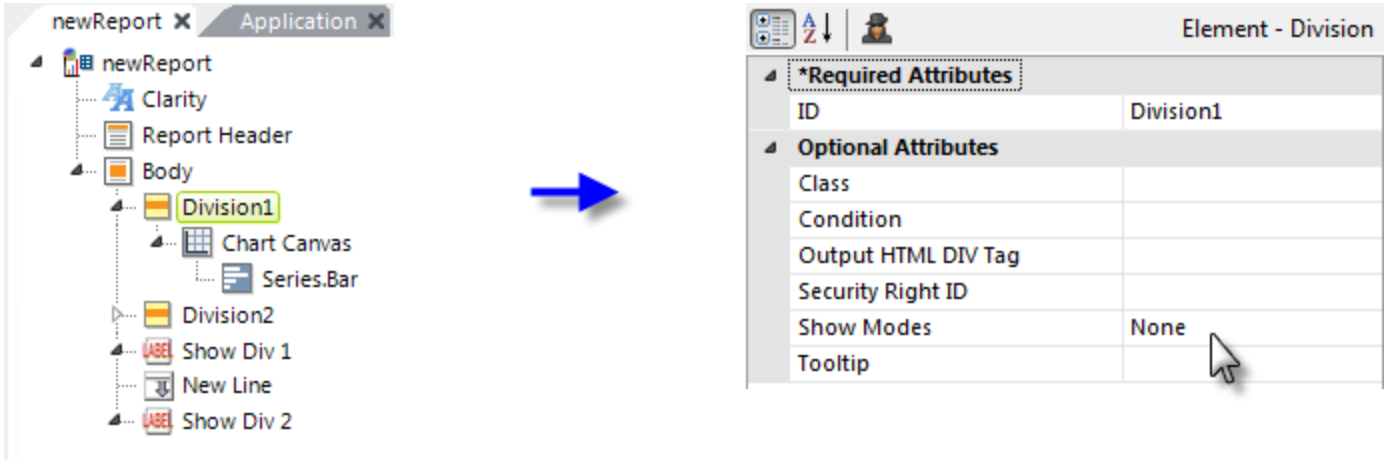
Action Elements Example: Showing and Hiding Elements

The **Action.Show Element** element allows you to dynamically change part of your web page when the user clicks on a link, image, or button, using *Show Modes*. This is very useful for revealing and hiding page sections, best defined by "container" elements such as HTML Rows or the **Division** element.



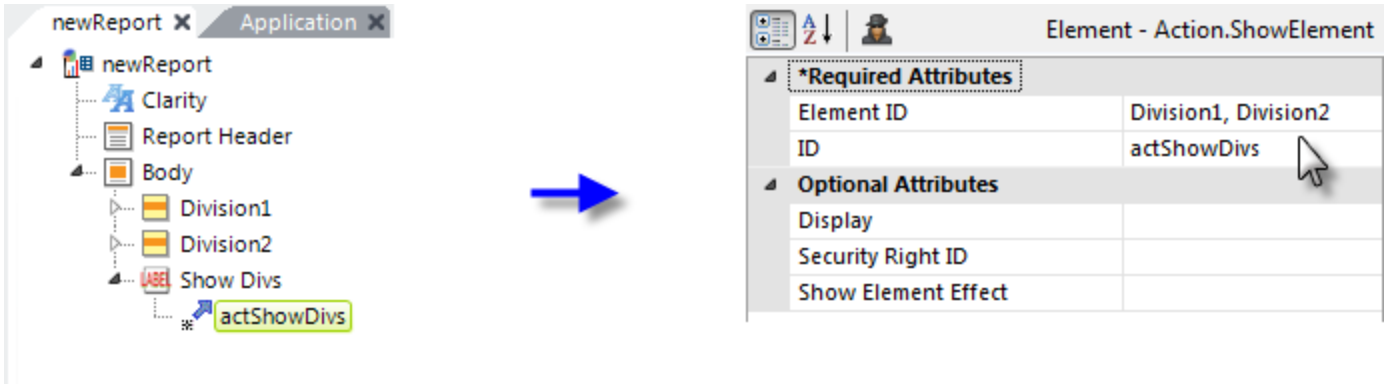
Here's how to use this element to show and hide other elements:

1. We begin by determining which elements we want to show and hide. In the example above, we've added two Division elements in order to control the visibility of their Chart Canvas child elements.
2. Label links, one for each Division, have been added with **Action.Show Element** elements beneath them. At runtime, these will initiate the showing and hiding of our divisions.
3. Next, set the Action.Show Element element's **Element ID** attribute values to the element IDs of the Divisions.
4. Leave the **Display** attribute blank; its default setting is *Toggle*, so alternate mouse clicks will show and then hide the Divisions.

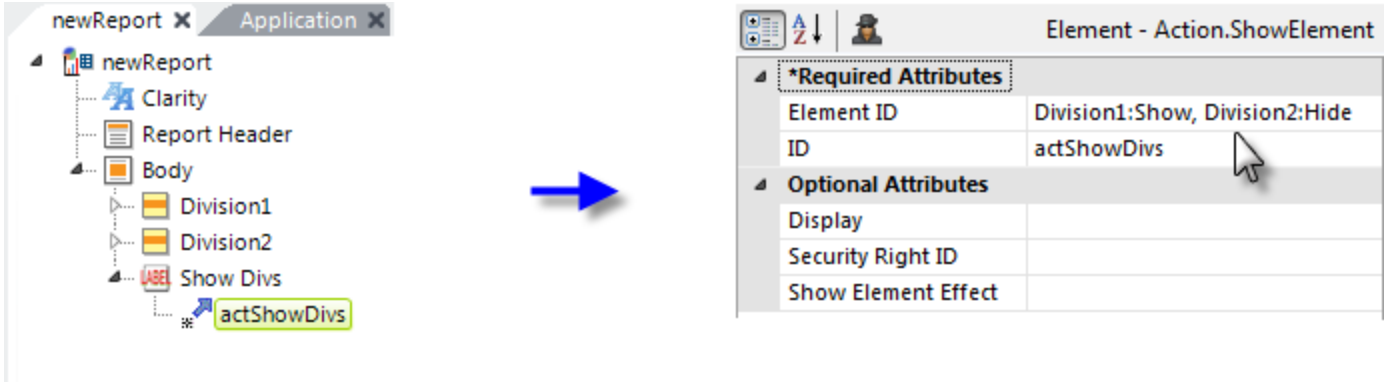


5. Finally, we need to set the **Show Modes** attribute for the elements being shown and hidden, as shown above. We want the division be hidden initially, so we set the attribute value to *None*.

This example will display two links when run. When either link is clicked, their respective division will be displayed. Another click, and they'll be hidden. Neither division is dependent on the other, so any combination of shown and hidden is possible. Here's another use case: one link that *alternates* which division is visible:



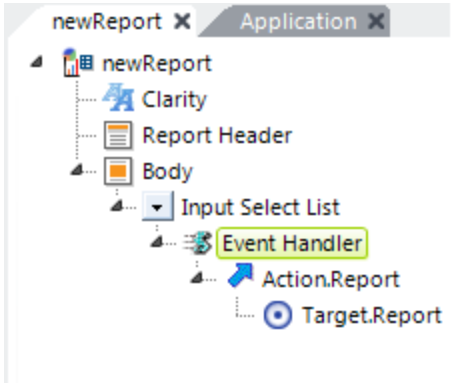
As shown above, this is accomplished by placing the IDs of *both* of the elements to be shown/hidden in the **Element ID** attribute of a single Action.Show Element element, separated by a comma. Change the **Show Modes** attribute value of one of the two Divisions to *All* so that it will initially be visible. Any number of elements can be included this way. Finally, it also possible to specify which action, Show or Hide, to apply to each Division independently:



As shown above, by adding *:Show* or *:Hide* after the element ID you can independently assign the desired action to each element. This is often useful in a menu-type situation. More examples of Show Modes in action are available in "Show Modes" on page 51.

Action Elements Example: With an Event Handler

Action elements can also be used with Event Handler elements, letting an event trigger an action.

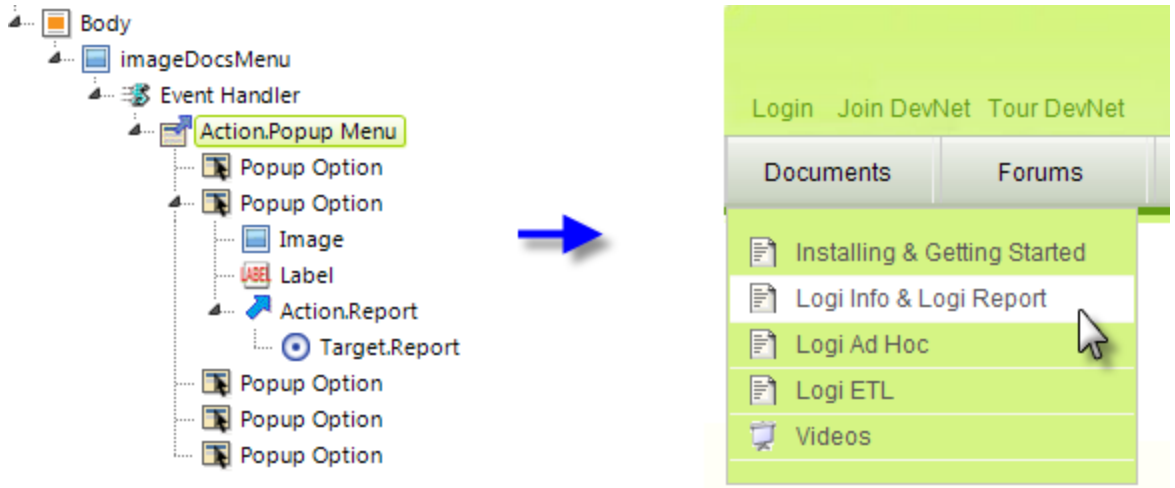


In the example shown above, an Input Select List element is the parent of an **Event Handler** element. Action and Target elements follow beneath. The Event Handler is set to fire when the DHTML *OnChange* event occurs, so the new report will be displayed when the selection in the list changes.

You can use *multiple* Event Handlers under a single parent element, each for a different event and each with its *own* Action element. For example, a link created by using a Label element can have an Event Handler for the *onMouseDown* event, and a second one for the *onClick* event. In this way, two Actions can be triggered when the link is clicked. The usefulness of this approach varies, however, based on the types of events the parent element supports and the order in which they're fired. For more specific information about using Event Handlers, see "Working with User Input Elements" on page 642.

Action Elements Example: With an Event Handler Popup Menu

The **Action.Popup Menu** element can be used with event handlers to cause a "pop-up" menu to appear when the mouse passes over a link or image.



The example shown above shows the elements used to produce the Documents popup menu (sometimes called a "pull-down" menu) on an early version of DevNet:

- The Event Handler element is set to fire on the *onMouseOver* event. When it fires, it executes the **Action.Popup Menu** element beneath it.
- A **Popup Option** element is included for each pop-up menu item to be displayed.
- Each Popup Option element needs, at least, a **Label** element and an **Action** element beneath it to cause something to happen when the pop-up menu item is clicked. If an **Image** element is included, it will also be displayed.

The Action.Popup Menu element has a **Popup Location** attribute that controls where the pop-up menu appears in relation to the Image or Label clicked to call it. The popup menu can be beneath it, as shown above, or beside it to the right.

Action.Show Element

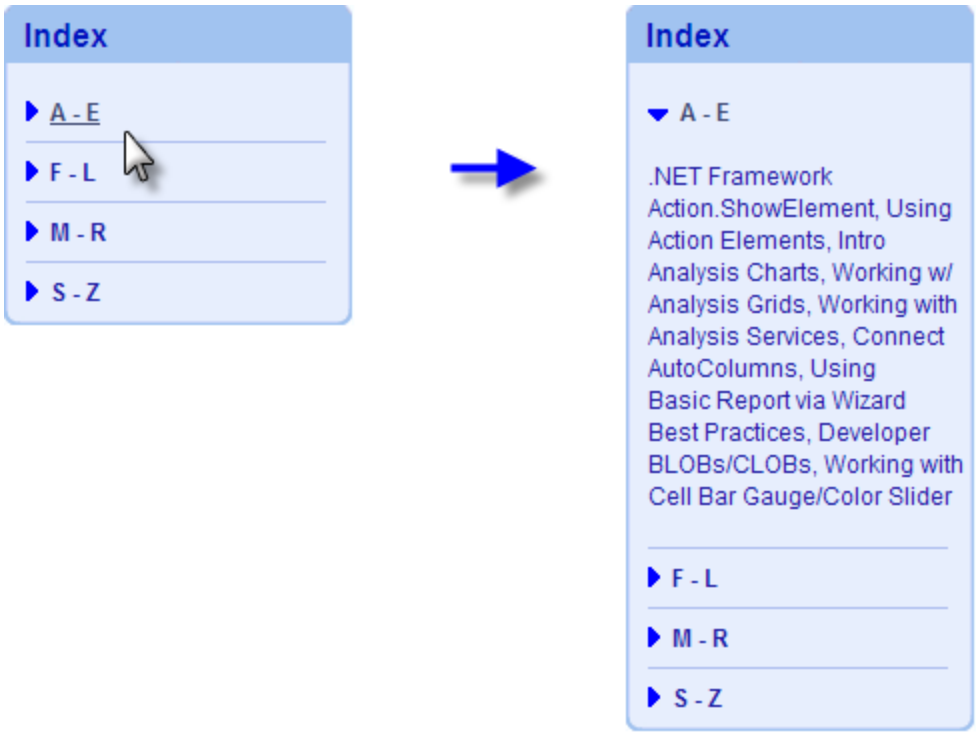
The Action.Show element is used in Logi reports to *show* and *hide* sections of a report or elements within it. Using DHTML techniques, this element lets you give the end-user dynamic control over page content, which is very useful.

The following topics provide examples for using the Action.Show element:

- Usage Example: Show/Hide Divisions
- Usage Example: Show/Hide a More Info Row

About Action.Show Element

Like most Action elements, Action.Show element is the child of a parent element such as **Label**, **Image**, or **Button** element, and is activated when its parent element is clicked.



In the example shown above, clicking the "A - E" link on the left results in the Index panel expanding to reveal links for the documents in that category. Clicking it again will collapse the panel back to its original state.

This is accomplished by placing the document links within a **Division** element; clicking the A-E link activates an Action.Show Element element that alternately shows or hides the division. The background images are applied using style and automatically expand or collapse to fill the space that the text requires.

The Action.Show Element element's **Element ID** attribute determines which element (the "target" element) will be affected when the Action.Show Element is activated. Action.Show Element works by overriding the **Show Modes** attributes of the target

element. Multiple target elements can be specified in a comma-separated list and you can independently assign the desired action (show or hide) to each element in the listby adding :Show or :Hide after the element ID.

 These terms are *case-sensitive*; you must use ":Show", for example, not ":show". In addition, if these explicit actions are specified, the usual "toggle" effect will not work. In other words, clicking on a link that calls Action.Show element with <ElementID>:Show will show the element, but another click on the same link will not hide it.

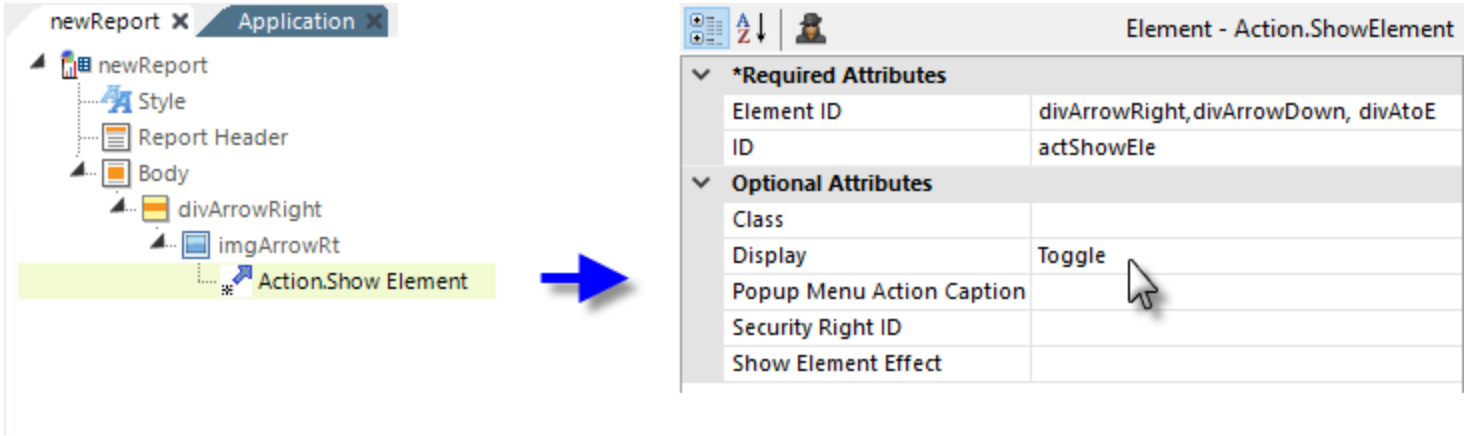
Action.Show Element's **Display** attribute determines whether a target element will be visible or invisible; there are three values that can be selected: *Show* (visible), *Hide* (invisible), or the default*Toggle* (opposite of current state).

The **Show Element Effect** attribute can be set to *FadeIn* and the target element will be shown with a brief transparency effect, so it appears to "fade-in".

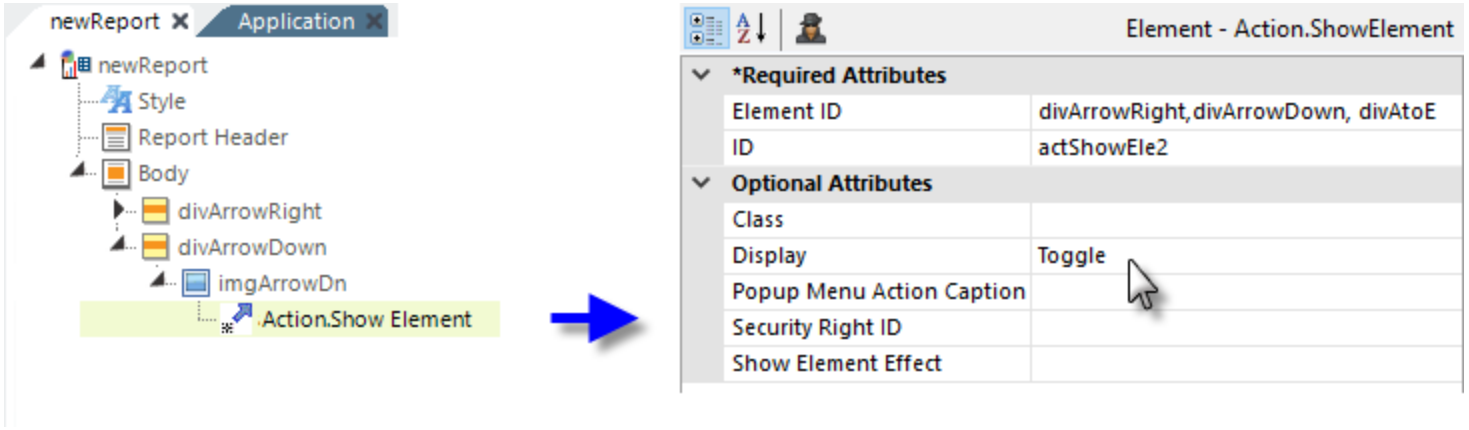
Unlike other actions, Action.Show Element does not have a specific **Target** child element. No links can be made to another report definition and only elements within the same definition can be controlled.

Action.Show Element Example: Show/Hide Divisions

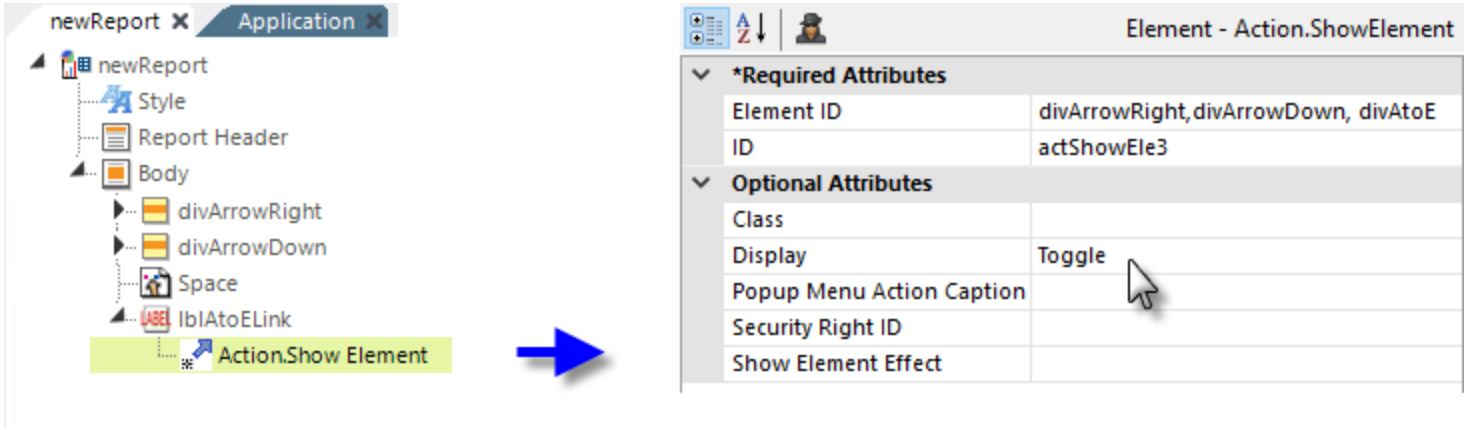
This example provides the functionality shown in "Action.Show Element" on page 417: showing and hiding links in the DevNet Developer Library Index. The example actually includes three links that can be clicked to activate the Action.Show Element element: a"right arrow" image, a"down arrow" image, and the A-E text link itself.



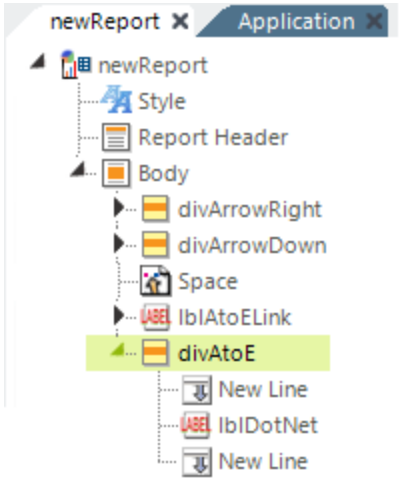
1. Add a **Division** element to contain the "right arrow", as shown above. Set its **Show Modes** attribute to *All.*
2. Beneath it, add the actual right arrow image and beneath it, an **Action.Show Element** element.
3. We'll have three divisions that are affected by the Action.Show Element element, so all of their element IDs are entered, as a comma-separated list, into the **Element ID** attribute, as shown above.
4. The **Display** attribute is set to *Toggle* so that we get alternating show/hide behavior.



5. Add a **Division** for the "down arrow" and repeat the previous steps, as shown above. Set the new division's Show Modes attribute to *None.* Be sure to give the new division an Element ID that is in the comma-separated list of element IDs for the Action.Show Element element.



6. Add **Space** and **Label** elements, as shown above.
7. The Label will become the A-E text link, so add an Action.Show Element element beneath it and set its attributes as shown. Be sure to specify the IDs of the three divisions in a comma-separated list in the Element ID attribute.



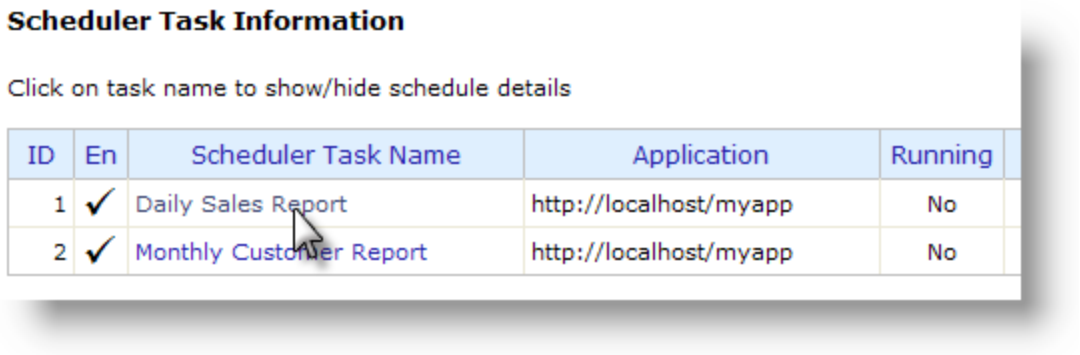
8. Finally, add the last **Division** element, as shown above, and beneath it add the detail text to be shown and hidden. Set the new division's **Show Modes** attribute to *None.* Be sure to give the new division an Element ID that is in the comma-separated list of element IDs for the Action.Show Element elements.

By setting the Show Modes attributes to *All* or *None*, you determine which divisions will be initially shown and which hidden. The Action.Show Element element works by overriding these Show Modes settings. The values *All* and *None* are just arbitrary text, so you can use whatever values make sense in the specific situation.

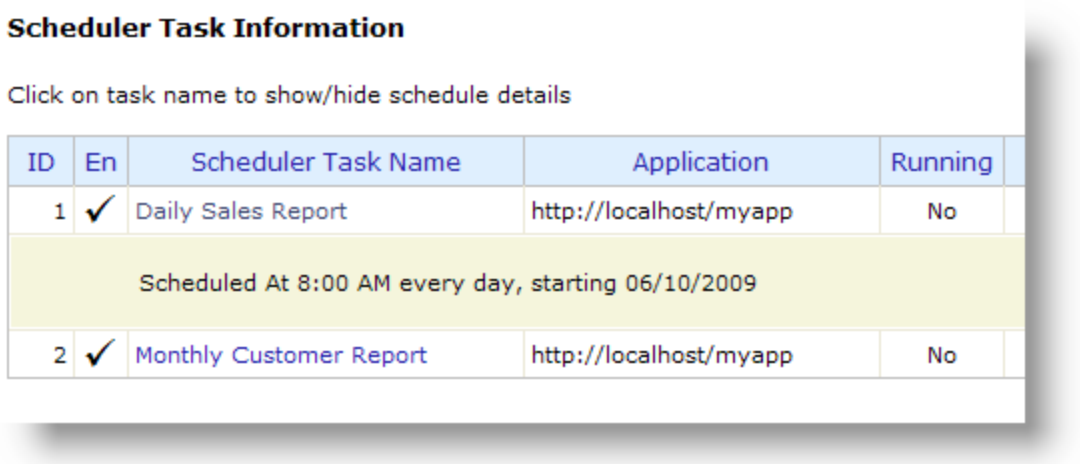
Preview the report and you'll see how the divisions are shown or hidden when the images or link are clicked.

Action.Show Element Example: Show/Hide a More Info Row

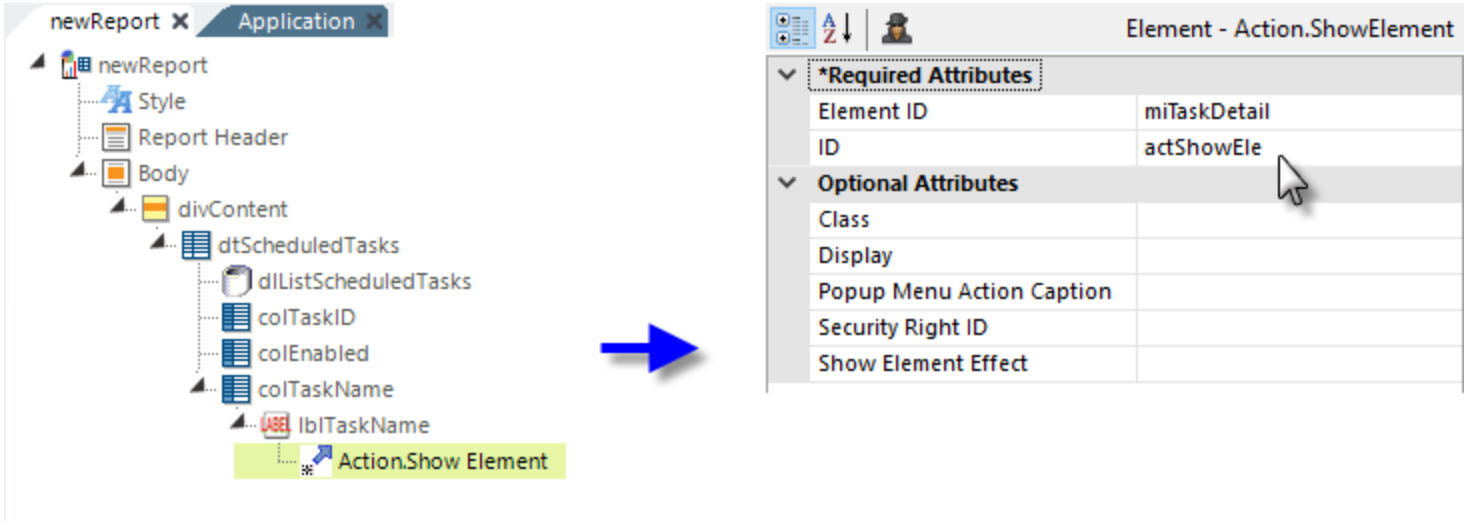
This example illustrates the use of the Action.Show element to show and hide a More Info Row within a Data Table. For more information, see *Data Table Rows*.



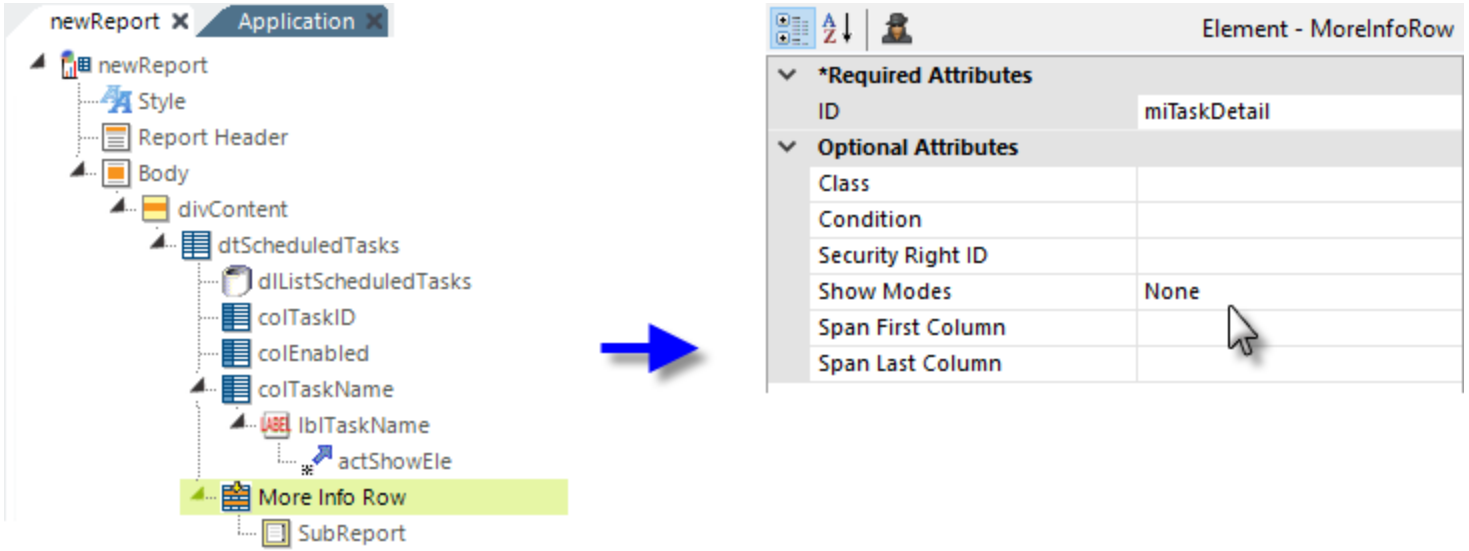
In the image shown above, the Logi Scheduler Console sample application displays two scheduled tasks. The Task Name is a link that can be clicked to show detail information,



as shown above. Clicking the Task Name link again hides the detail information. This is a handy way of making lots of information available in a Data Table when you have limited horizontal space on the page or a large number of data columns. Here's how this was done:



1. In the example report definition above, an **Action.Show** element has been added as the child of the **Label** element used to display the data in the table's Task Name column, making the data itself a link.
2. The new element's **Element ID** attribute value has been set to the name ("miTaskDetail"). This name will also be given to a More Info Row element added in the next step.
3. The **Display** attribute's default value is *Toggle*, so it has been left blank.



4. Next, a **More Info Row** element has been added to the table, as shown above.
5. Its **ID** has been set to match the ID used in the previous step ("miTaskDetail") and its **Show Modes** attribute has been set to an arbitrary text value, *None*.

The **More Info Row Column** element can be used to closely align More Info Row columns with the columns of its parent table. When this element is used, the More Info Row element's **Span First Column** and **Span Last Column** attributes are left *blank*, and the More Info Row Column element's **Column Span** attribute is used instead to control spanning of parent table columns.

The **More Info Row** element has a **Condition** attribute, so you can conditionally include the element, for greater control over when the element can be seen, primarily when the More Info Rows are displayed by default.

Now, when run, the report will display its Data Table and clicking the Task Name will insert a row of additional detail information right below the row that was clicked, as desired. A second click will hide the detail information.

Shared Elements

Shared elements provide an opportunity to create "reusable code", with its traditional benefits of greater efficiency and single instance maintenance, in Logi reports.

The following topics discuss shared elements:

- Creating a Shared Element Definition
- Using Shared Elements in a Report Definition
- Passing Parameters to Shared Elements
- Passing Elements to Shared Elements
- Shared Element Usage Tips

About Shared Elements

Developers often find that they're using the same elements, or groups of elements, repeatedly in *multiple* report definitions. Report headers and footers are good examples. However, rather than adding these elements or groups of elements repeatedly into numerous report definitions, and being faced with the challenge of maintaining them, developers can instead use **shared elements**.

A "shared element" is a *container* for one or more elements, identified by the shared element ID. It's referenced with a kind of placeholder element in a definition. At runtime, the Logi Server Engine replaces the placeholder with the actual elements from the shared element container, before the report HTML is generated.

Any changes made to a shared element "ripples through" to every definition that uses it, which makes maintenance easy. For example, you may wish to add a copyright message in your pages; by making it a shared element, you'll only have to change the year in one location each January.

A shared element also provides a level of abstraction similar to that found in object-oriented programming: a shared element can produce different results in different report definitions based on the data or variables fed to it. So it's not only reusable but its results and behavior can be dynamic based on external factors, including tokens, definition modifiers, stylesheets, etc. that reside in the definition that uses the shared element. This is a powerful feature that developers can leverage to make themselves more efficient and their reports more reliable.

The examples below are from the Template Modifier Files sample application. Please read this topic carefully as the words "shared" and "element" appear many times in many ways and can be confusing.

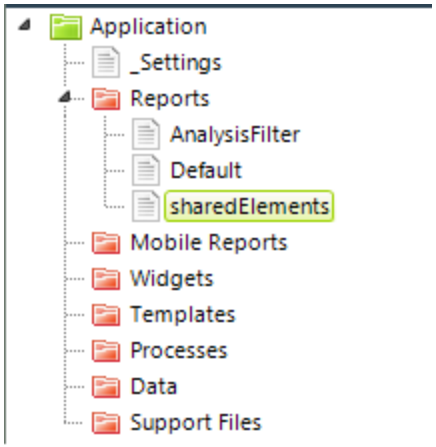
A **Master Report** is a special report definition that includes elements that are to be included in other reports. It's frequently used to provide a template-like frame, often defined with a header, a menu, and a footer, for integration with other reports. You may find this to be an useful alternative to building the same thing using shared elements. For more information, see *Master Reports*.

The **Shared Element Params** element allows you to pass values into a shared element. These values are applied before the shared element is inserted into the calling report definition, making the shared element itself dynamic. The **Passed Shared Element** element even allows you to pass and include groups of elements, from the calling definition, into a shared element. These elements are discussed in more detail in later sections.

The **Include Shared Element** element has a **Condition** attribute, allowing you to dynamically employ shared elements. The Condition attribute of shared elements does not support Default Request Parameter values, as the Default values are initiated after the shared elements have been processed. Request tokens can be used (as long as they are provided explicitly; however, default values won't be available when the conditions are evaluated.

Creating a Shared Element Definition

Shared elements can be defined in any report definition but the recommended practice is to create one or more *separate* report definitions in Logi Studio just to hold shared elements:



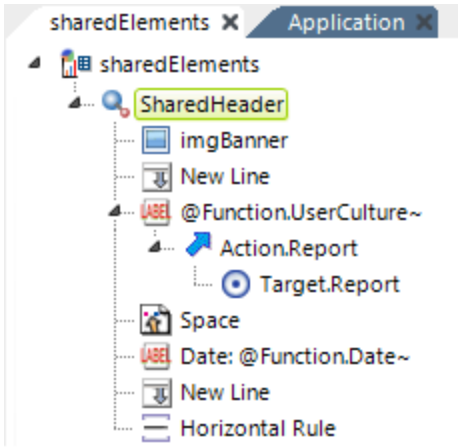
In the example shown above, a separate definition named "sharedElements" has been created to hold the shared elements. This approach has several benefits, including the possibility of copying the definition file itself to other Logi applications, extending the reusability concept and enforcing consistency.

If you're creating a shared elements definition for a Mobile Reporting application, be sure to place the definition within the **Mobile Reports** folder.



In the **sharedElements** definition, as shown above, a shared element definition is started by adding a **Shared Element** element. This is the basic container for one or more elements that you want to share. Be sure to give this element the required unique ID. This definition doesn't need any of the standard elements usually found in a report definition, such as Style, Report Header, Body, and Report Footer.

A definition file can hold many Shared Element elements, each containing one or more elements to be shared.

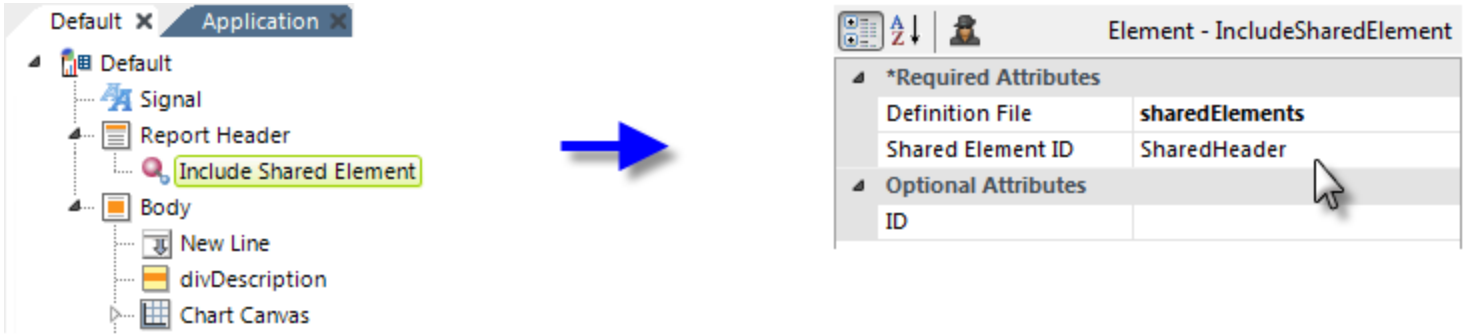


As shown above, a variety of elements can be added beneath the Shared Element element. These can include any element available for report definitions. In the example, elements have been added to create a common header for several reports.

Your shared elements are now available to any report definition in your application.

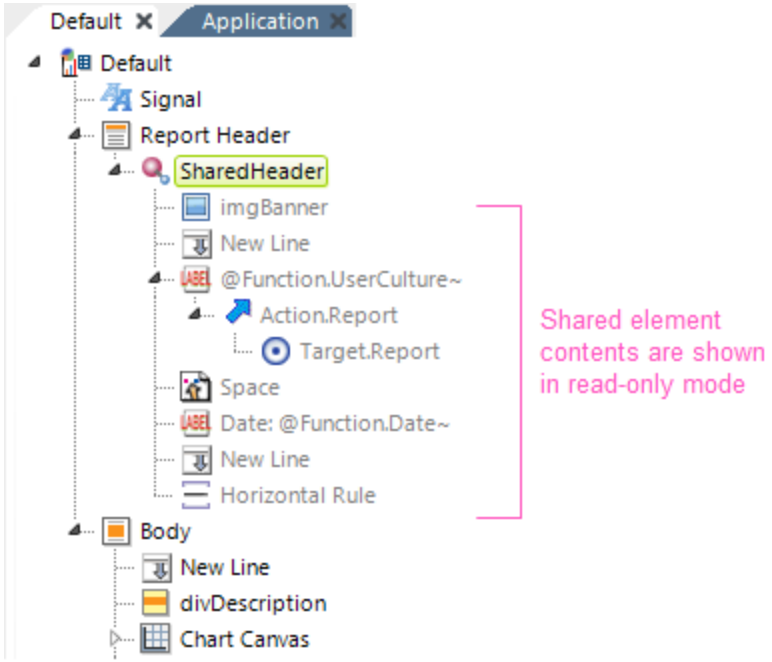
Using Shared Elements in a Report Definition

Using your shared elements in a report definition is very easy. Here's how to use a shared report header:



1. Add an **Include Shared Element** element to your report definition, as shown above, in this case beneath the Report Header element.
2. Set its **Definition File** attribute value by selecting the *sharedElements* definition from the drop-down list of definitions.
3. Set its **Shared Element ID** attribute value by selecting the *SharedHeader* definition from the drop-down list of shared elements in the previously selected definition.

Include Shared Element elements can be added anywhere in your report definition, and the same shared element can even be used multiple times in the same definition, ifnecessary.



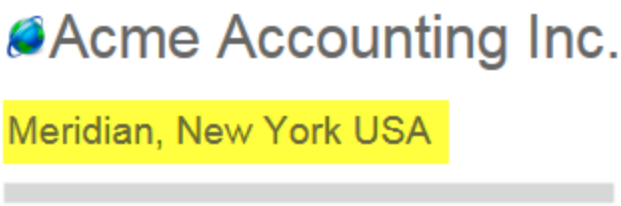
As soon as you select the shared element in the Attributes panel, its contents will be displayed, in read-only mode, in the Element Tree, as shown above. Any modifications to it must be made in the definition that contains the shared element.

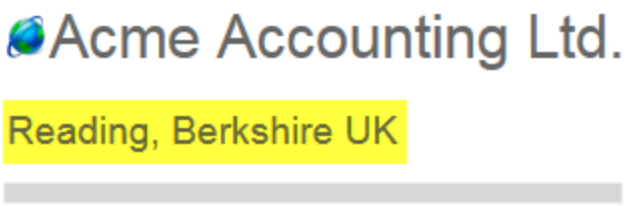
Now you can copy the configured Include Shared Element element into the Report Header of every report definition in an application. This will result in the same header appearing on every report.

Shared elements can also be used to include common but *non-visual* code in definitions, such as shim scripts, meta tags, and more.

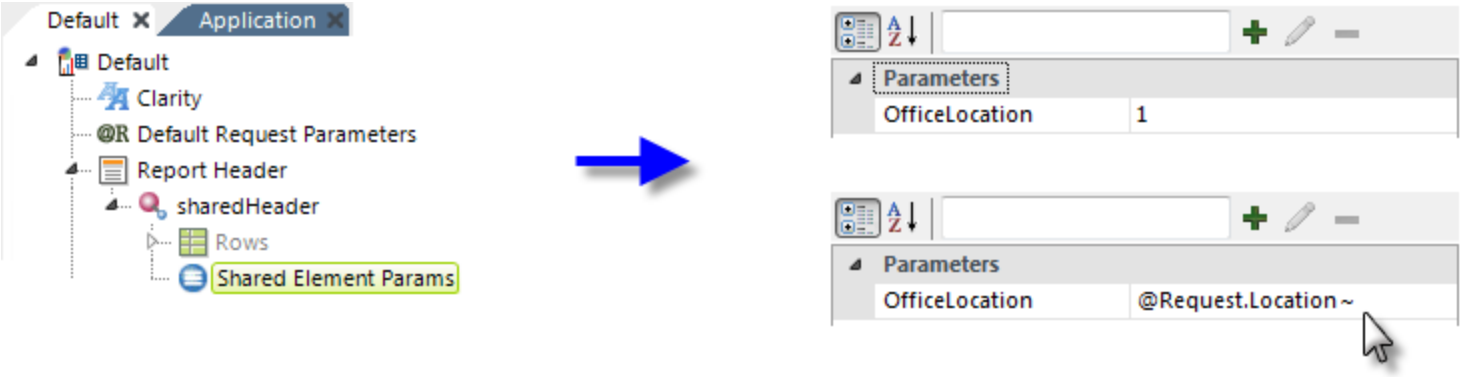
Passing Parameters to Shared Elements

Shared elements can be made dynamic by passing them parameters. The passed values are then applied to the shared elements *before* they're included in the report definition. Here's an example:



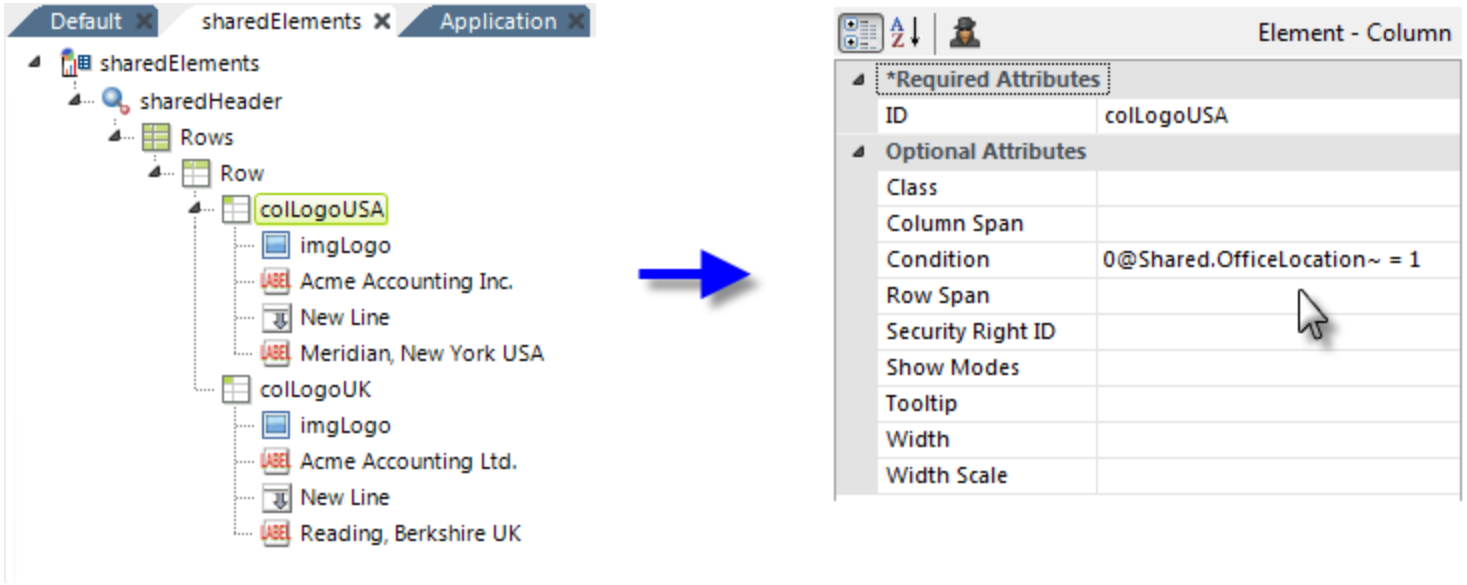


Imagine that we want to make use of shared report header code, but we want to change the *location* displayed with the logo (highlighted above), depending on some variable. Shared element parameters are perfect for this situation.



In the example code above, in our Default report definition, we added an **Include Shared Element** element ("sharedHeader") beneath the Report Header element. As usual, the code from the shared element appears in Read-Only mode beneath it ("Rows").

Next, we added a child **Shared Element Params** element and created a single parameter, "OfficeLocation", and gave it a value. The value can be literal or derived from tokens, as shown. This parameter will tell the shared code which location text to use in the header.



In the shared element definition, shown above, we can see how the passed parameter is used. The **@Shared** token provides access to the passed parameter. In our example, it's used in a Condition attribute to control which column is displayed in the header.

If tokens are used for parameter values, they are passed as intact tokens, *not* as the token *values*. The tokens will not be resolved in the shared element definition - instead, they'll be inserted with the shared elements into the calling report definition and will be resolved when the report definition is rendered.

```
<SharedElementParams
myParam1="Welcome to McLean"
```

```
myParam2=7900
myParamReq="@Request.myVal~"
myParamSes="@Session.myVal~"
myParamData="@Data.myColumnVal~"
myParamProc="@Procedure.myVal~"
myParamColName="myColumnName~"
/>
```

To further clarify how they're used, theXML source code forsome examples of Shared Element Parameters isshown above.

```
<SharedElement ID="mySharedElement" >
 <Label Caption="@Shared.myParam1~" />
<Label Caption="@Shared.myParam2~" />
<Label Caption="@Shared.myParamReq~" />
<Label Caption="@Shared.myParamSes~" />
<Label Caption="@Shared.myParamData~" />
<Label Caption="@Shared.myParamProc~"/>
<Label Caption="@Data.@Shared.myParamColName~~" />
</SharedElement>
```

The XML source code above shows a Shared Element element and the passed parameters being used in Label elements via @Shared tokens. This is similar to the way you'd use @Request tokens to access the values of Link Params in a Process Task.

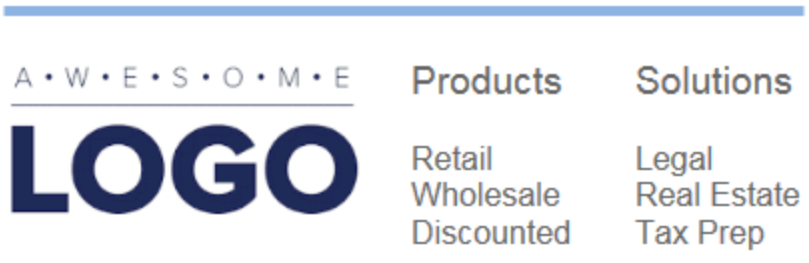
The last Label element above uses a nested token - when the calling report definition is rendered, the inner token (@Shared) will be resolved first, providing the literal column name, then outer token (@Data) will be resolved. For more information, see *"Nesting" Tokens*.

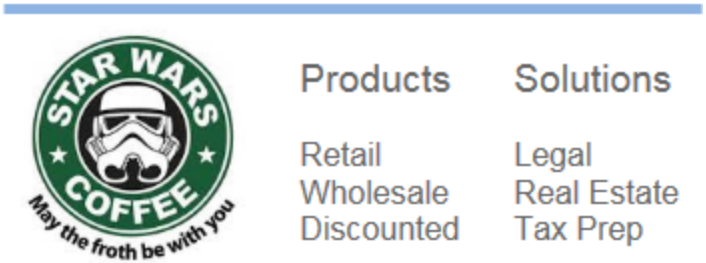
```
<Label Caption="Welcome to McLean"/>
<Label Caption="7900"/>
<Label Caption="@Request.myVal~"/>
<Label Caption="@Session.myVal~"/>
<Label Caption="@Data.myColumnVal~"/>
<Label Caption="@Procedure.myVal~"/>
<Label Caption="@Data.myColumnName~"/>
```

When the calling report definition is processed and the shared element is inserted into it, the source code for the inserted Label elements would look like the examples above. Note how the Caption attributes relate to the original shared element parameters.

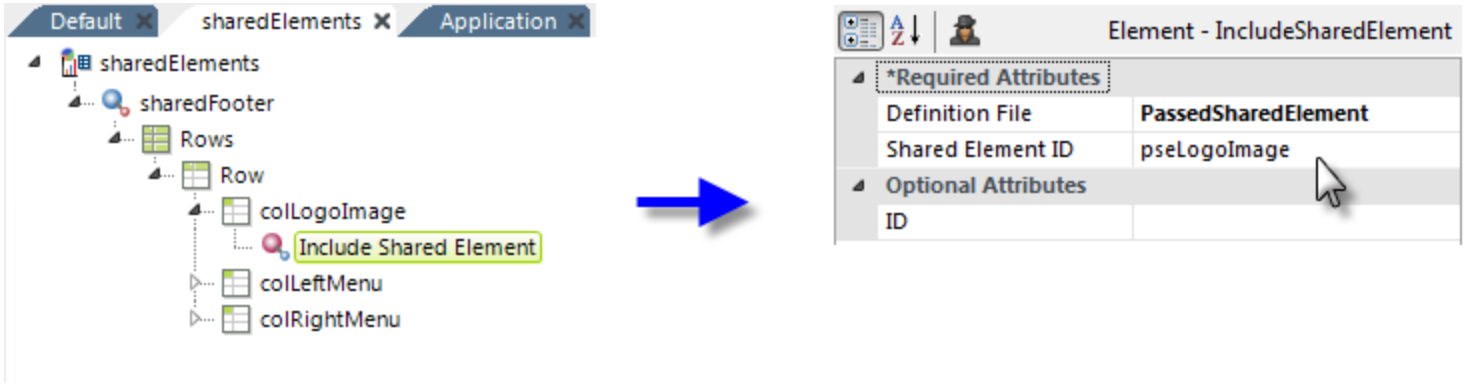
Passing Elements to Shared Elements

There may be situations where it's desirable to pass *entire elements* into a shared element. The passed element is then added to the shared element before it's applied to the calling report.

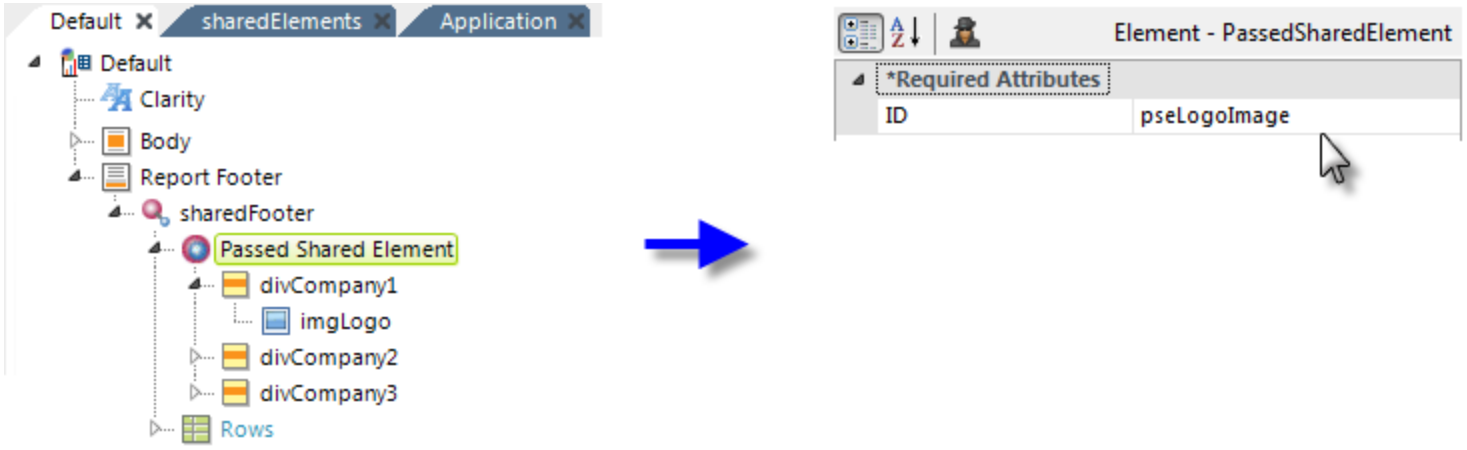




Imagine that you provide a report with a footer that includes a logo image and standard menu links, like the one shown above. Depending on the consumer, you need to show a different logo and the images are different sizes and types. Passing a configured Image element to the shared element is a good solution.



This time we'll start our example, shown above, in the sharedElements report definition and construct our footer. Where the logo image will appear, we added an **Include Shared Element** element. When configuring it, we used the special identifier "PassedSharedElement" for the Definition File attribute (it appears at the very top of the pull-down list of options), and the ID that we'll give later to the Passed Shared Element element in the calling definition.



In our Default definition, shown above, we added an Include Shared Element element in the footer. Beneath it, we added a **Passed Shared Element** element and set it ID attribute to match the one we used in the sharedElement definition. Then we added child elements beneath it, including the Image element we're passing.

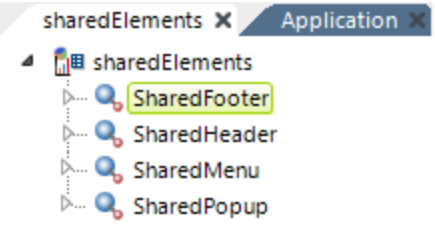
Using Division elements as containers for the image allows us to make the choice of image conditional, based on the Divs' Condition attributes.

This is a very powerful feature. Another interesting possible implementation would be to use a Passed Shared Element to pass a *datalayer* to a visualization, allowing it to work with data from different data sources.

Shared Element Usage Tips

The following is a list of helpful tips when using Shared Elements:

- The insertion of the shared XML source code in a shared element into a report at runtime is done *without* dynamic review of the "parent-child" element relationship rules, so it's possible to include shared elements where they don't belong. Therefore, as the developer it's your responsibility to ensure that the "element rules" are being followed. To do this, we recommend that you create shared elements by first, using all the elements to be shared right where they belong, directly in a report, then successfully testing them, then copying them out to the shared element definition, and finally replacing them with the Include Shared Element element.
- Multiple Shared Element elements can be placed in the same definition:



- **Stylesheets** apply to shared elements in several ways. If the shared elements do not include a style sheet of their own then the style sheet of the definition using the shared elements is applied. If a style sheet *is* included in one of the shared elements, then any style classes with the same name in both it and the report definition's style sheet will cascade.
- Both the **Definition File** and **Shared Element ID** attribute values of the Include Shared Element element can be specified using tokens.
- If a shared element definition file becomes very large, it may be useful and may improve performance to create multiple shared element definition files. Some developers group their shared elements into different definition files based on their function or purpose.

- **Action.ShowElement** cannot be used to show/hide an Include Shared Element element, although use of Division elements can achieve conditional inclusion of shared elements.

- Shared elements are inserted into a report definition when the report runs, so any Request parameters that are passed to the report can be used in the shared elements. Similarly, shared elements can work with *data* from a datalayer in a parent report definition, but they must be in scope; i.e. inserted beneath the chart or table that's the parent of the datalayer. The exception to this is data from LocalData, which can be used throughout the report and shared elements.
- You may also include shared elements in Process definitions.

Repeat Elements

The **Repeat Elements** element allows data-driven replication of elements, making Logi report definitions dynamic and able to automatically accommodate data schema changes.

The following topics discuss the use of the Repeat Elements element:

- Example: Building a Comma-Separated String of Values
- Example: Dynamic Data Table Columns
- Example: Dynamic Tables with Nested Repeat Elements
- Example: Dynamic Analysis Grid Columns

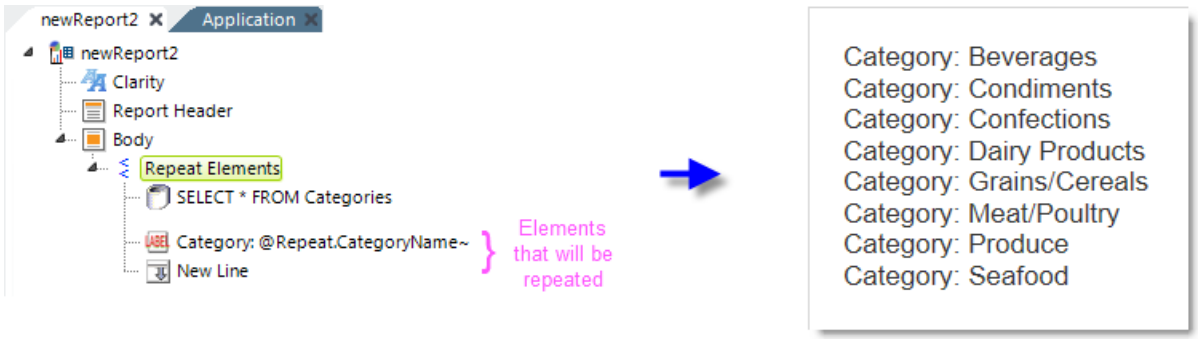
About the Repeat Elements Element

It's not unusual for Logi developers to encounter situations in which there's an uncertainty about the data they're working with. That uncertainty makes it hard to know, during development, how many Logi elements are necessary to present the data in a report definition. This may be due to a changing database schema, or to expanding data, or other reasons.

For example, imagine a scenario in which you're creating a monthly report that displays a table for each project proposal your company receives during the month. However, the number of proposals varies widely each month. How can you create a report with a Data Table for each project if you don't know how many of them there will be (short of editing the definition each month, which is hardly efficient)?

Another example is a situation in which you want to significantly *reduce* the *number* of elements in, and shorten, a report definition, just to make it easier to work with.

The **Repeat Elements** element is the right tool for these and similar jobs! It's associated with a datalayer and iterates through each row, telling the Logi Server Engine at runtime to create a complete set of its child elements for each row.



In the definition example above, left, a **Repeat Elements** element is used. It has its own datalayer and two other child elements: a **Label** and a **New Line**. All of the child elements, except the datalayer, will be replicated - one set of child elements for *each* datalayer row - by the Logi Server Engine at runtime. The result is shown above, right.

A special token, @Repeat, is used to represent the data. In the example above, the **Caption** of the Label element is configured as:

```
Category: @Repeat.CategoryName~
```

As usual, the second part of the token is the name of a column in the datalayer.

Yes, you could have just as easily gotten the same result using a regular Data Table. However, the value of the Repeat Elements element will become more obvious as we go along. The Repeat Elements element can be used *anywhere* in a report definition, and it can take almost every other element as a child element (except as noted below), so you have a lot of flexibility in using it. In the following sections, you'll see examples of a variety of use-cases for this element.

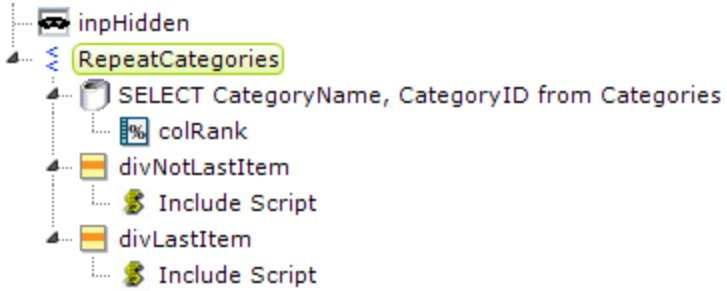
Some Token & Element Restrictions

 Repeat Elements are one of the very *first* things processed when a page goes through the Logi Server Engine and, therefore, some elements and values that are processed *later* are not available for use with Repeat Elements, including:

- @Local tokens
- @Session tokens for variables that are set in the definition using the **Set Session Variables** element
- @Request tokens for variables that are set in the definition using the **Default Request Parameters** element
- Do not use **DataLayer.Linked** to provide data to Repeat Elements
- Do not include Repeat Elements in a **Shared Element**; it will not be processed

Repeat Elements Example: Building a Comma-Separated String of Values

Have you ever needed to build a comma-separated string of values from data? With the Repeat Elements element, it's easy!



In the definition fragment shown above, a Repeat Elements element is used to iterate through the data and an **Input Hidden** element is used as the repository of the comma-separated string of values.

The traditional difficulty when building comma-separated strings is avoiding having a trailing comma after the last item in the list. To do this, we need to know when we're processing the last data row. This is where the **Rank Column** element ("colRank") comes into play. It ranks the datalayer rows, based on the Category ID in this example, in *reverse* (set its **Reverse Rank** attribute to *True*), which means the *last* data row will have a rank of *1*.

With that ranking information in hand, we can use two **Division** elements, with their **Condition** attributes set to test the ranking, to control the JavaScript that builds the comma-separated string. Here's the Condition attribute for the `divNotLastItem` element above:

```
@Repeat.colRank~ <> 1
```

and the other division's Condition tests to see if the rank does equal 1. The two JavaScript scripts used to create the string either include a trailing comma, or not. Here's the script *with* the comma:

```
var myHidden = document.getElementById('inpHidden');
myHidden.value += '@Repeat.CategoryName~,';
```

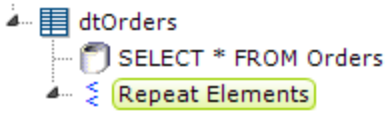
and the other script is exactly the same but omits that highlighted comma.

After the page is submitted, the completed comma-separated string of values will be available to the next definition as `@Request.inpHidden~`.

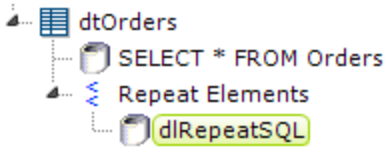
Ranking may not work as a test for the last datalayer row for all scenarios, but there are other ways to do it, including using a Sequence Column and a Aggregate Column (set to COUNT the rows) and comparing the two.

Repeat Elements Example: Dynamic Data Table Columns

Suppose you have a requirement to display a Data Table that may have a varying number of columns. You could use **Auto Columns** to display the table columns but you want more control over formatting and appearance. This is a perfect opportunity to use Repeat Elements, but assumes that your datasource can accept queries about the data schema. Here's how:



1. Start by adding a **Data Table** element and a datalayer to your definition. Then, as a child of the Data Table, add a **Repeat Elements** element, as shown above.



2. Next, add a datalayer beneath the Repeat Elements element. This datalayer is going to query the datasource, in this example a SQL server, for information about the table schema. You will need to know the specific syntax required to do this, which can vary widely from one datasource to another. This query is for Microsoft SQL Server 2012:

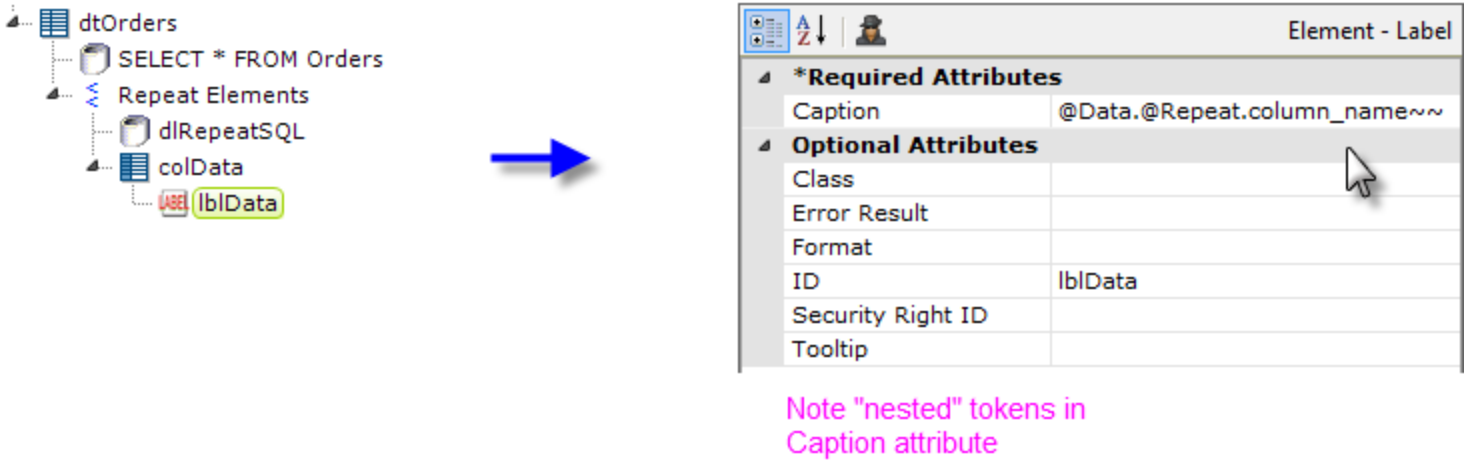
```
SELECT column_name,
CASE WHEN data_type= 'nchar' THEN 'Text'
WHEN data_type = 'int' THEN 'Number'
 WHEN data_type = 'money' THEN 'Number'
WHEN data_type = 'datetime' THEN 'Date'
```

```
END As column_type
FROM information_schema.columns
WHERE table_name = 'Orders'
AND ordinal_position < 9 -- limits number of columns returned
ORDER BY ordinal_position
```

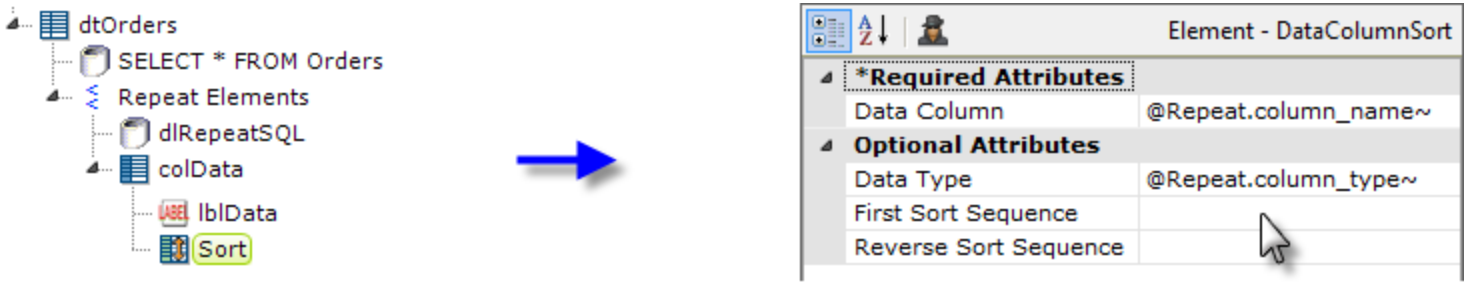
It's pretty straightforward, except for the CASE statement. We're going to add **Sort** elements to our Data Table columns later and, to do that, we need to know the *data type* for each column. The CASE statement translates the data type name in the database's system schema table to one of the values used in the Sort element's Data Type attribute. The actual data types available to you for use in the CASE statement will depend on your datasource.



3. Now we can add the **Data Table Column** element, as shown above. Its **Column Header** attribute makes use of the `@Repeat` token and the column name we retrieved in the query.



4. To display data in the column, we add a **Label** element. Its **Caption** uses something you may not have seen before: "nested" tokens. Basically, this is a token within a token; the inner token gets evaluated first, then the outer token. If this was a mathematical expression, it might look like: `@Data.(@Repeat.column_name~)~` and the token in the parenthesis gets evaluated first.



5. And, finally, we add the **Sort** element and set its attributes, using `@Repeat` tokens, as shown above.

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight
10248	VINET	5	1996-07-04T00:00:00-04:00	1996-08-01T00:00:00-04:00	1996-07-16T00:00:00-04:00	3	32.3800
10249	TOMSP	6	1996-07-05T00:00:00-04:00	1996-08-16T00:00:00-04:00	1996-07-10T00:00:00-04:00	1	11.6100
10250	HANAR	4	1996-07-08T00:00:00-04:00	1996-08-05T00:00:00-04:00	1996-07-12T00:00:00-04:00	2	65.8300
10251	VICTE	3	1996-07-08T00:00:00-04:00	1996-08-05T00:00:00-04:00	1996-07-15T00:00:00-04:00	1	41.3400
10252	SUPRD	4	1996-07-09T00:00:00-04:00	1996-08-06T00:00:00-04:00	1996-07-11T00:00:00-04:00	2	51.3000

When the report is run, we get something like the table shown above. Obviously, additional formatting can be done as well. If the schema of the table changes, the report will handle it without any additional development.

Special Crosstab Table Considerations

If you want to create dynamic **Crosstab Tables** using Repeat Elements and the techniques shown above, there are some special considerations:

When working with Crosstab Tables, you may decide to use **Extra Crosstab Value Column** elements, summaries, or individual column IDs, all of which use special column names consisting of an "rd" reserved word, a dash, and a column ID, such as `rdCrosstabValue-OrderTotal`.

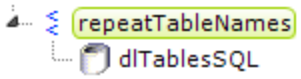
However, these special column names won't work in nested tokens. For example, this nested token *won't* work:

`@Data.rdCrosstabValue-@Repeat.column_name~~`

In order to use these special column names correctly in nested tokens, you need to concatenate the "rdCrosstab" reserved word and the dash with the column ID value in the column name data in the Repeat Element element's datalayer. You can do this in your SQL query, or with a Calculated Column element. The `@Data.@Repeat.column_name~~` nested token will then work correctly.

Repeat Elements Example: Dynamic Tables with Nested Repeat Elements

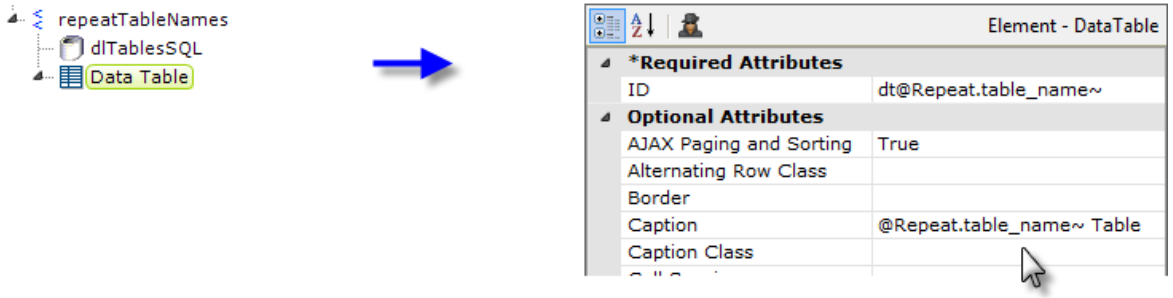
Let's expand on the previous example - this time we don't even have to know the table names. We can use nested **Repeat Elements** elements to gather all the information to we need to display the data for whatever tables are available. Let's also do a little more sophisticated formatting and aligning this time, and some summarizing.



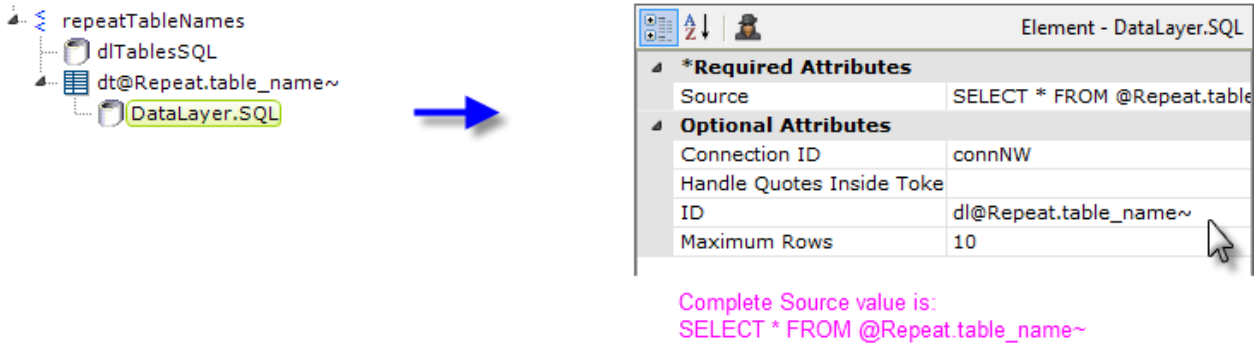
1. This time we'll start by adding a **Repeat Elements** element to our definition, with a child datalayer, as shown above. Once again, the syntax for querying the schema will vary by datasource and this example is for Microsoft SQL Server 2012. The SQL query, which limits us to three tables for this example, is:

```
SELECT table_name FROM information_schema.tables
WHERE table_name IN ('Customers', 'Employees', 'Orders')
ORDER BY table_name
```

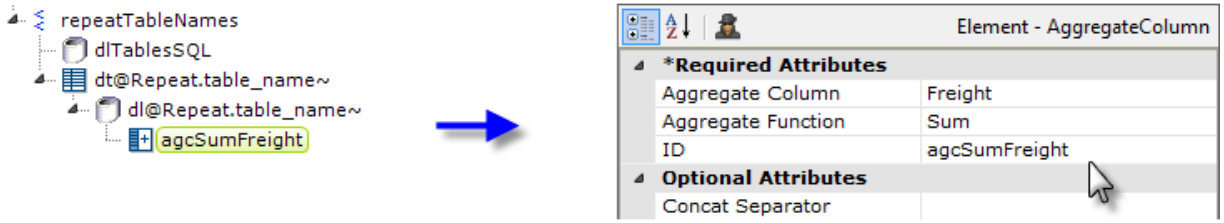
Think of this as the "outer loop" of the code, which allows us to iterate through each of the tables.



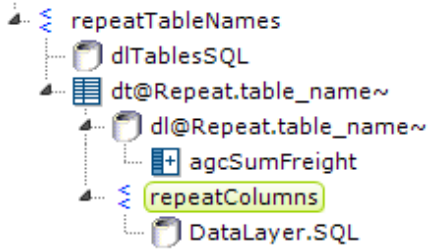
2. Now we'll add a **Data Table** element, and use @Repeat tokens from the datalayer to give each table its required unique ID and a caption, as shown above.



3. Add a **datalayer** beneath the Data Table, as shown above, and set its attributes as shown above, right. This query will retrieve that data for each table.



4. We want to summarize freight costs for the Orders table, so add an **Aggregate Column** element beneath the datalayer and set its attributes as shown above.



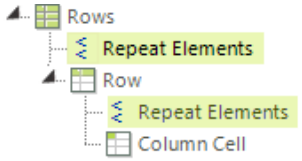
5. Now things begin to get interesting: add *another***Repeat Elements** element beneath the Data Table, as shown above, with its own datalayer. This is the "inner loop" of the code, which iterates the table columns. The SQL query for this datalayer is:

```
SELECT column_name, data_type,
CASE WHEN data_type = 'nchar' THEN ''
WHEN data_type = 'int' THEN '######'
WHEN data_type = 'datetime' THEN 'Short Date'
 WHEN data_type = 'money' THEN 'Currency'
END As column_format,
CASE WHEN data_type = 'nvarchar' THEN 'ThemeAlignLeft'
WHEN data_type = 'nchar' THEN 'ThemeAlignCenter'
WHEN data_type = 'int' THEN 'ThemeAlignRight'
WHEN data_type = 'datetime' THEN 'ThemeAlignCenter'
WHEN data_type = 'money' THEN 'ThemeAlignRight'
END As column_align
FROM information_schema.columns
```

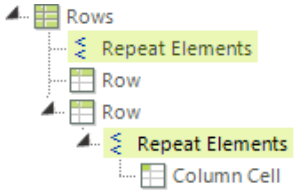
```
WHERE table_name = '@Repeat.repeatTableNames.table_name~'
AND ordinal_position < 9 -- limits number of columns returned
ORDER BY ordinal_position
```

As shown above, the query uses CASE statements to set formatting and alignment values based on the data type the column and, in the WHERE clause, uses an @Repeat token from the *first* Repeat Elements element (the "outer loop") to limit the result set to a specific table.

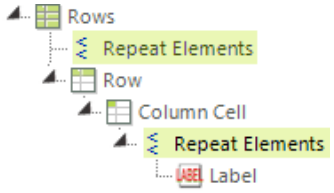
You can identify columns from the first Repeat Elements element more specifically using a token that includes its element ID: `@Repeat.repeatTableNames.table_name~`.



Valid Arrangement

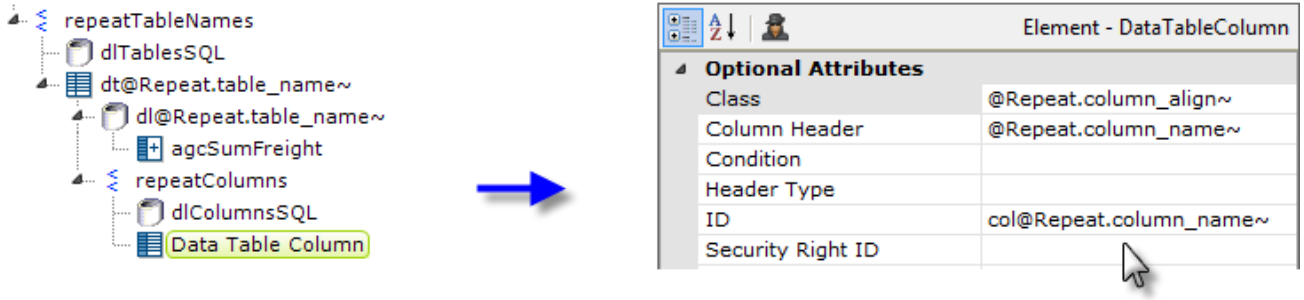


Valid Arrangement

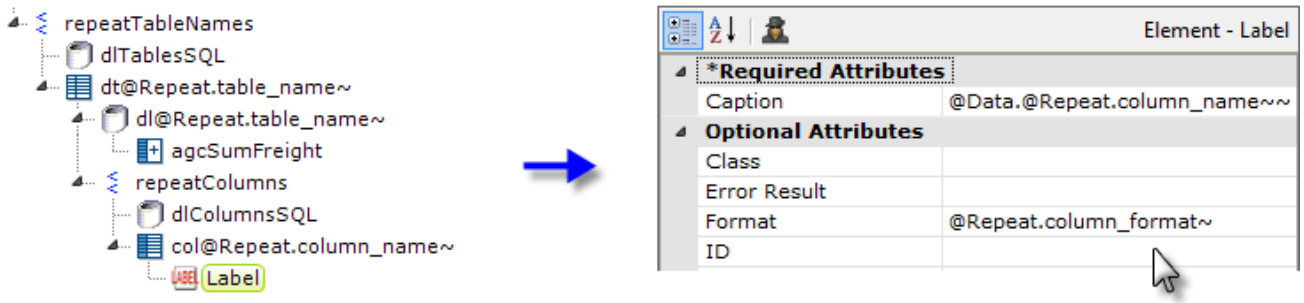


INVALID Arrangement

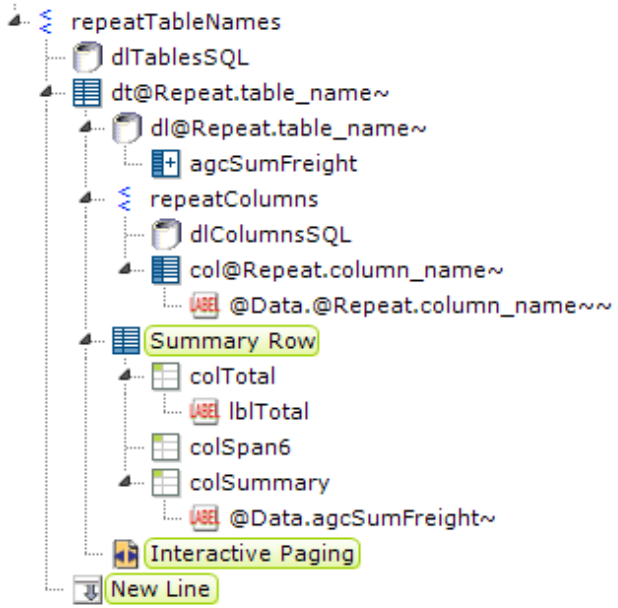
When using nested Repeat Elements elements in any type of table, you cannot have more than one table hierarchy (Rows, Row, Column Cells) level between them. As shown above, the arrangement of rows and columns in the left and middle examples is correct. The right example is *invalid* and the second Repeat Elements element will not be processed.



6. At last we can add a **Data Table Column** element under the second Repeat Elements element, as shown above, and set its attributes using @Repeat tokens.



7. And then we can add a **Label** element to display the data, using nested tokens as before, in its attributes as shown. We'll also use an @Repeat token to set the format.



8. Finally, we can finish up by adding **Summary Row** (and its child elements) and **Interactive Paging** child elements beneath the Data Table and a **New Line** element beneath the first Repeat Elements element, as shown above. To ensure that the summary row only appears for the Orders table, set the Condition attribute of its three **Column** child elements to `"@Repeat.table_name~" = "Orders"`.

Customers Table

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	PostalCode
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	12209
ANATR	Ana Trujillo Emparedados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.	05021
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.	05023
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London	WA1 1DP
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå	S-958 22
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim	68306
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg	67000
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid	28023

Page 1 of 2

Employees Table

EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate	HireDate	Address
1	Davolio	Nancy	Sales Representative	Ms.	12/8/1948	5/1/1992	507 - 20th Ave. E. Apt. 2A
2	Fuller	Andrew	Vice President, Sales	Dr.	2/19/1952	8/14/1992	908 W. Capital Way
3	Leverling	Janet	Sales Representative	Ms.	8/30/1963	4/1/1992	722 Moss Bay Blvd.
4	Peacock	Margaret	Sales Representative	Mrs.	9/19/1937	5/3/1993	4110 Old Redmond Rd.
5	Buchanan	Steven	Sales Manager	Mr.	3/4/1955	10/17/1993	14 Garrett Hill
6	Suyama	Michael	Sales Representative	Mr.	7/2/1963	10/17/1993	Coventry House Miner Rd.
7	King	Robert	Sales Representative	Mr.	5/29/1960	1/2/1994	Edgeham Hollow Winchester Way
8	Callahan	Laura	Inside Sales Coordinator	Ms.	1/9/1958	3/5/1994	4726 - 11th Ave. N.E.

Page 1 of 2

Orders Table

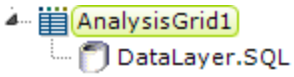
OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight
10248	VINET	5	7/4/1996	8/1/1996	7/16/1996	3	$32.38
10249	TOMSP	6	7/5/1996	8/16/1996	7/10/1996	1	$11.61
10250	HANAR	4	7/8/1996	8/5/1996	7/12/1996	2	$65.83
10251	VICTE	3	7/8/1996	8/5/1996	7/15/1996	1	$41.34
10252	SUPRD	4	7/9/1996	8/6/1996	7/11/1996	2	$51.30
10253	HANAR	3	7/10/1996	7/24/1996	7/16/1996	2	$58.17
10254	CHOPS	5	7/11/1996	8/8/1996	7/23/1996	2	$22.98
10255	RICSU	9	7/12/1996	8/9/1996	7/15/1996	3	$148.33
Total:							

Page 1 of 2

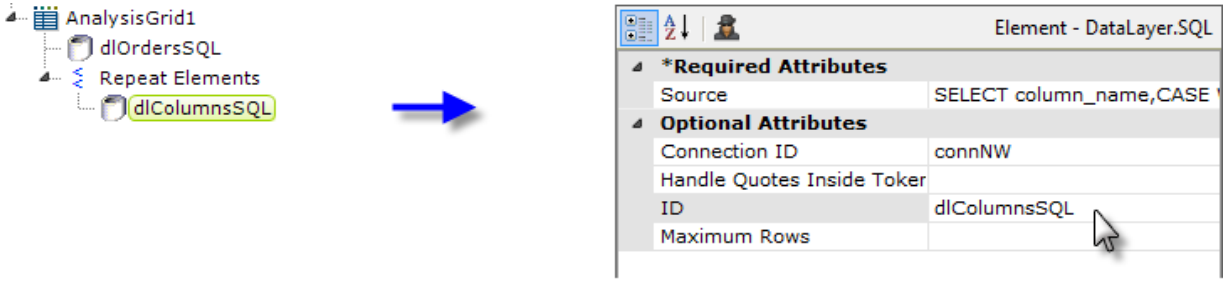
When the report is run, three tables should be displayed, as shown above.

Repeat Elements Example: Dynamic Analysis Grid Columns

Repeat Elements can also be used with super-elements, such as the **Analysis Grid**. This allows you to present dynamic Analysis Grids and let the user select the datasource at run time.



1. We'll start by adding an **Analysis Grid** element, and its child datalayer, to the definition, as shown above. The datalayer query will retrieve all of the data from the table: `SELECT * FROM Orders`

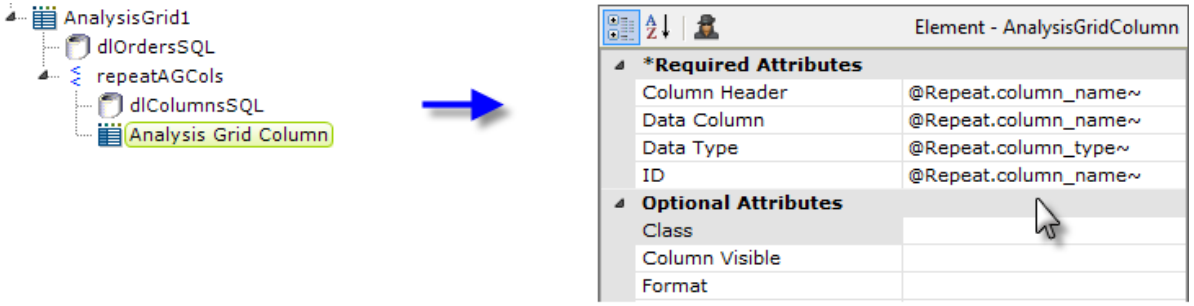


2. Next, we add **Repeat Elements** element as a child of the Analysis Grid, with its own child datalayer, as shown above. This datalayer will retrieve the column information from the datasource. Once again, the syntax for querying the schema will vary by datasource and this example is for Microsoft SQL Server 2012. The SQL query is:

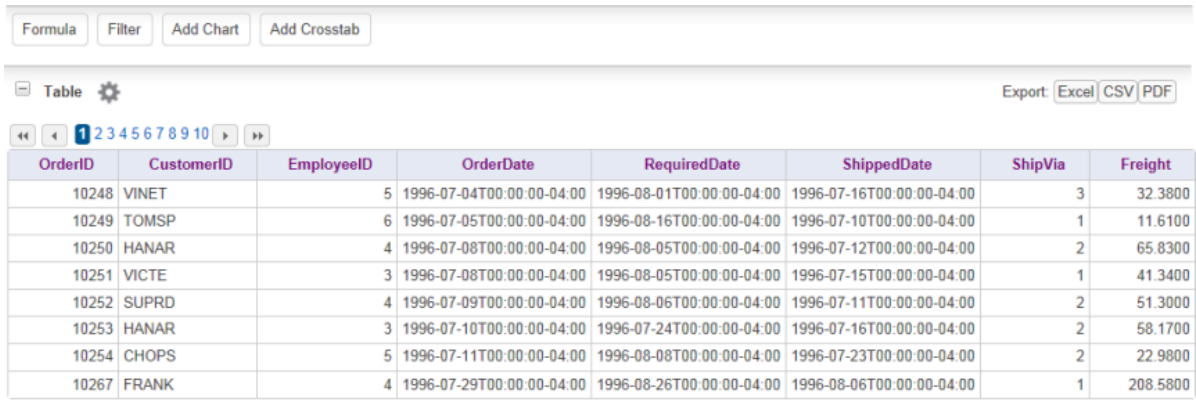
```
SELECT column_name,
CASE WHEN data_type = 'nchar' THEN 'Text'
WHEN data_type = 'int' THEN 'Number'
 WHEN data_type = 'money' THEN 'Number'
```

```
WHEN data_type = 'datetime' THEN 'Date'
END As column_type
FROM information_schema.columns
WHERE table_name = 'Orders'
AND ordinal_position < 9 -- limits number of columns returned
ORDER BY ordinal_position
```

It's pretty straightforward, except for the CASE statement. We need to specify the column data types for the Analysis Grid columns and, to do that, we need to know the data type for each column. The CASE statement translates the data type name in the system schema table to one of the values used in the **Analysis Grid Column** element's **Data Type** attribute. Once again, the actual data types available to you for use in the CASE statement will depend on your datasource.



3. Finally, we add an **Analysis Grid Column** element, as shown above, and set is attributes using @Repeat tokens.



OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight
10248	VINET	5	1996-07-04T00:00:00-04:00	1996-08-01T00:00:00-04:00	1996-07-16T00:00:00-04:00	3	32.3800
10249	TOMSP	6	1996-07-05T00:00:00-04:00	1996-08-16T00:00:00-04:00	1996-07-10T00:00:00-04:00	1	11.6100
10250	HANAR	4	1996-07-08T00:00:00-04:00	1996-08-05T00:00:00-04:00	1996-07-12T00:00:00-04:00	2	65.8300
10251	VICTE	3	1996-07-08T00:00:00-04:00	1996-08-05T00:00:00-04:00	1996-07-15T00:00:00-04:00	1	41.3400
10252	SUPRD	4	1996-07-09T00:00:00-04:00	1996-08-06T00:00:00-04:00	1996-07-11T00:00:00-04:00	2	51.3000
10253	HANAR	3	1996-07-10T00:00:00-04:00	1996-07-24T00:00:00-04:00	1996-07-16T00:00:00-04:00	2	58.1700
10254	CHOPS	5	1996-07-11T00:00:00-04:00	1996-08-08T00:00:00-04:00	1996-07-23T00:00:00-04:00	2	22.9800
10267	FRANK	4	1996-07-29T00:00:00-04:00	1996-08-26T00:00:00-04:00	1996-08-06T00:00:00-04:00	1	208.5800

When the report is run, an Analysis Grid will be displayed that dynamically includes the available data columns. Data can be formatted and aligned using techniques from the previous example.

Responsive Design Elements

Responsive Design elements provide a dynamic layout and visibility mechanisms for Logi reports on a variety of devices.

The following topics discuss use of the Responsive Design elements:

- Using Responsive Rows and Columns
- Using Responsive Visibility

About Responsive Design

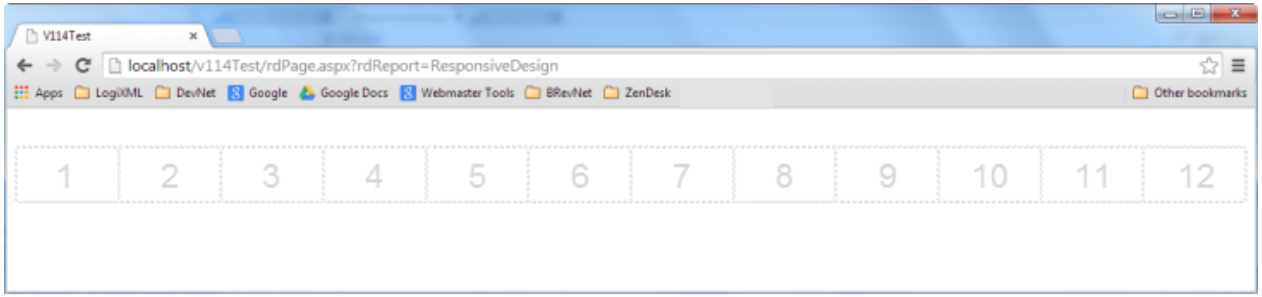
Responsive web design has the goal of creating pages that provide an optimal viewing experience, including easy reading and navigation with a minimum of resizing, panning, and scrolling, across a wide range of devices (from mobile phones to desktop computer monitors).

In a Logi application, this is accomplished using Responsive Design elements. They dynamically adapt the report layout to the viewing environment by using fluid, proportion-based grids and controlled visibility. When these elements are used, viewport sizes are automatically determined using CSS3 media queries and are standardized in Logi apps as the following:

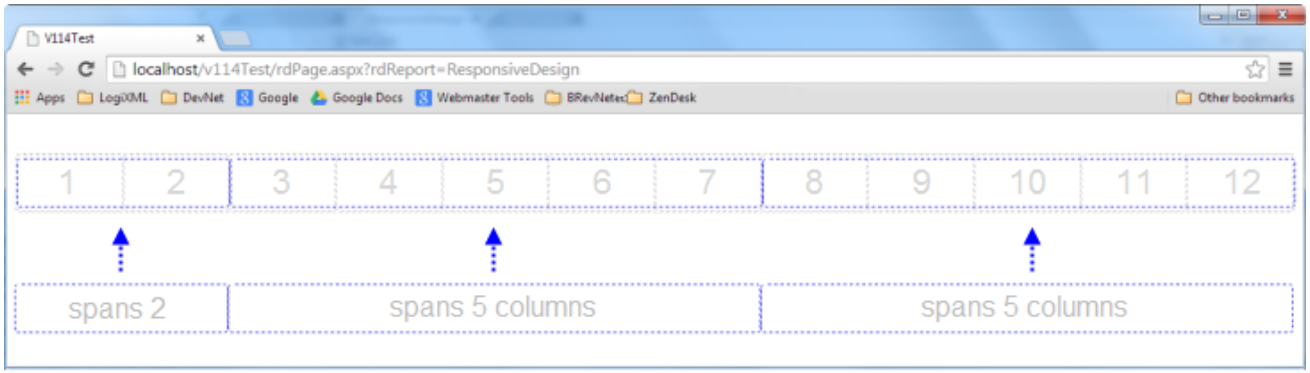
Device	Width	Attribute Name
Extra small devices: phones	< 768 pixels	Extra Small Screen
Small devices: tablets	≥ 768 pixels	Small Screen
Medium devices: desktops	≥ 992 pixels	Medium Screen
Large devices: desktops	≥ 1200 pixels	Large Screen

Using Responsive Rows and Columns

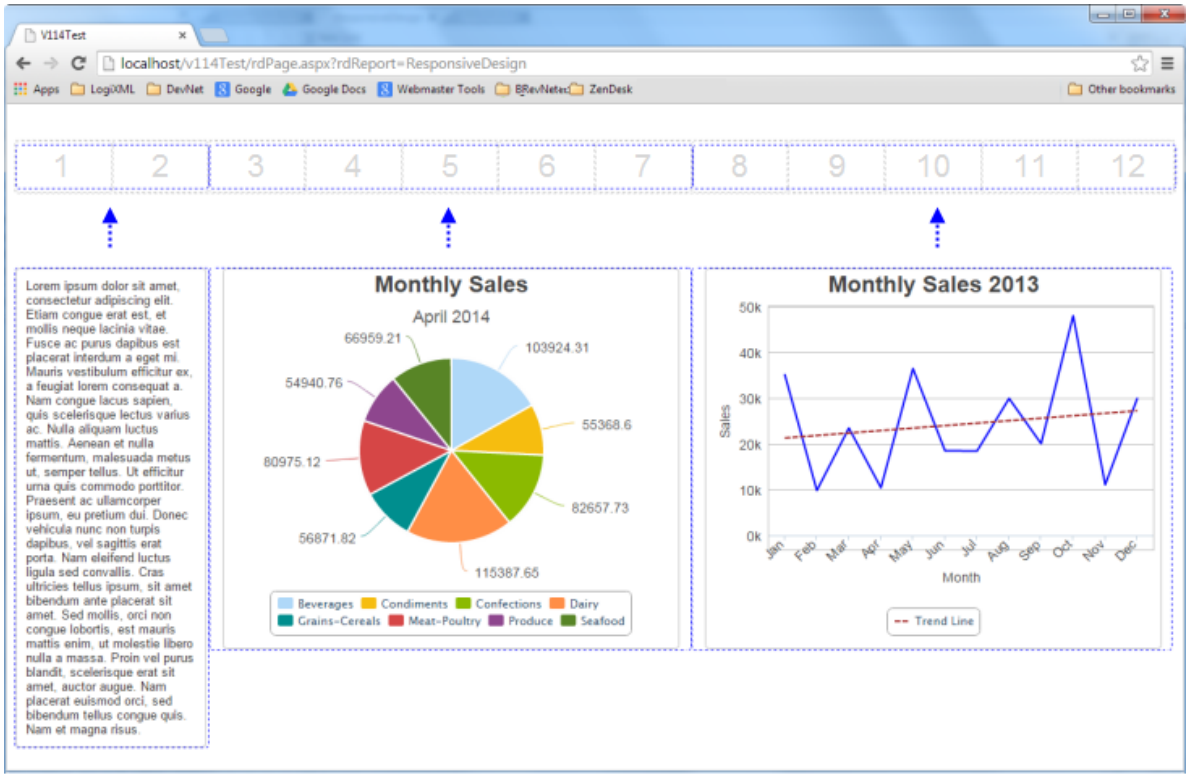
Logi applications have long had Rows, Row, and Column Cell elements available for construction of an HTML table object as a container for content. The **Response Row** and **Responsive Column** elements provide similar functionality but also dynamically resize to accommodate viewport sizes.



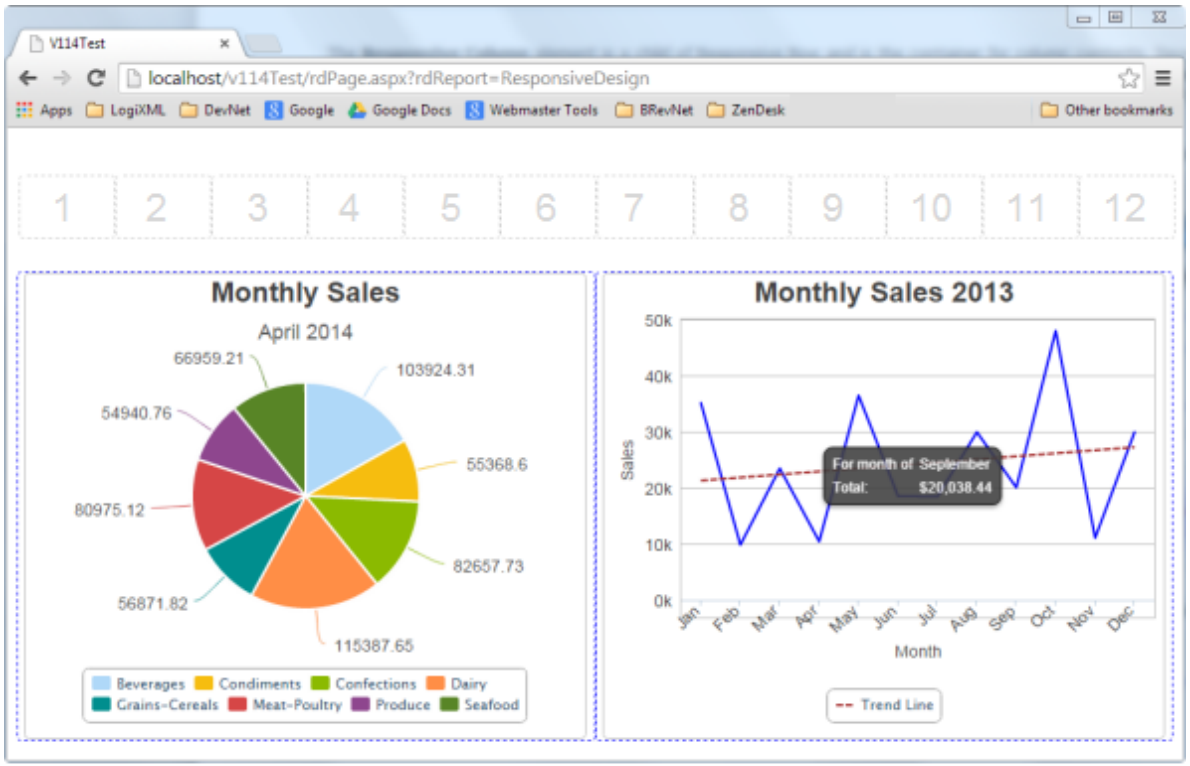
The **Responsive Row** element adds a "fluid grid" to the report page, as shown above, which fills 100% of the width of its parent element. The grid consists of 12 columns. This is similar to grids found in the Bootstrap and Foundation frameworks.



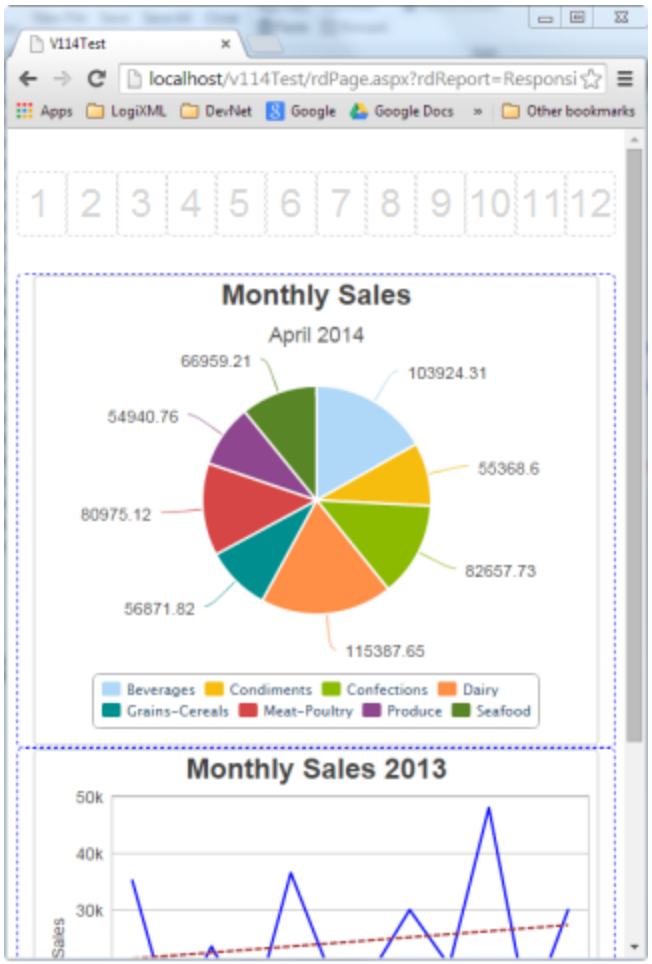
The **Responsive Column** element is a child of Responsive Row and is the container for column contents. Developers can configure how many grid columns each Responsive Column will span for each viewport size (or whether it will be displayed at all). In the example shown above, there are three Responsive Columns and, for the Large Screen viewport, they span two, five, and five Responsive Row grid columns. Normally these two elements are transparent, without borders or background color, though they can be styled.



In the example above, we see how content fits into the Responsive Column elements and how they span the grid in a Large Screen viewport.



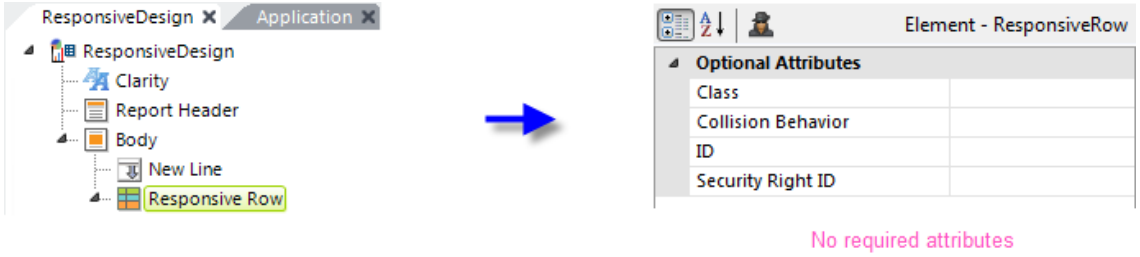
Notice what happens when the viewport width is changed to a Medium Screen size: the Responsive Row grid columns get proportionally smaller, as shown above. Responsive Column elements can be configured by the developer to react when the viewport window gets smaller, too. In this case, the first column is no longer visible at all and the other two now span six Responsive Row grid columns each.



And, finally, as the viewports gets even smaller, the Responsive Column elements can be configured to span *all* the Responsive Row grid columns, as shown above, producing a "stacked" appearance.

Implementation

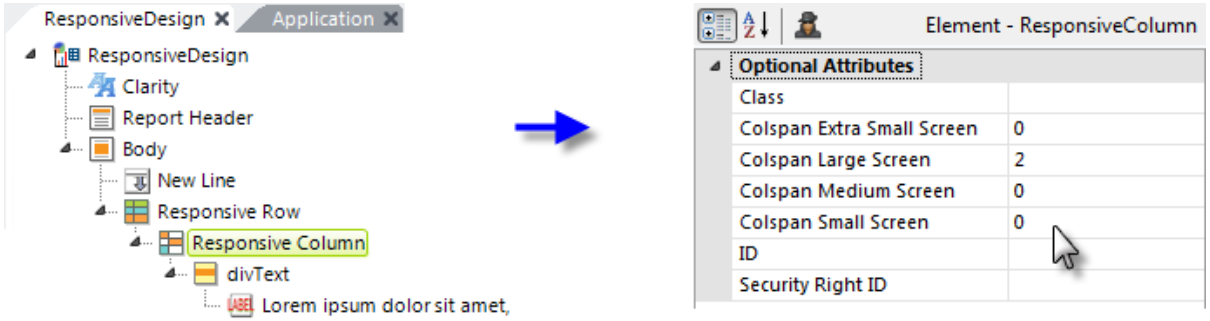
Here's an example showing how the elements discussed above are used:



The **Responsive Row** element can be the child of a variety of other elements. In the example above, it's been added directly beneath the Body element. It has no required attributes.

However, take note its **Collision Behavior** attribute. Responsive Row content works best when it dynamically shrinks and grows to fit the column width. Normally, report developers should give Responsive Column elements enough width so that the content fits. But there are times when, as the viewport gets smaller, content just can't shrink enough to fit. For example, a Data Table can only become so small or an image may have a fixed width. The Collision Behavior attribute specifies what to do when Responsive Columns "collide" in different viewport sizes because their contents can't shrink appropriately. Attribute value options include

- *HorizontalScrollbar* - (the default) displays a horizontal scroll-bar and some of the content is out of view.
- *Wrap* - moves the Responsive Column down to the next row.
- *Overlap* - makes the contents overlap each other.

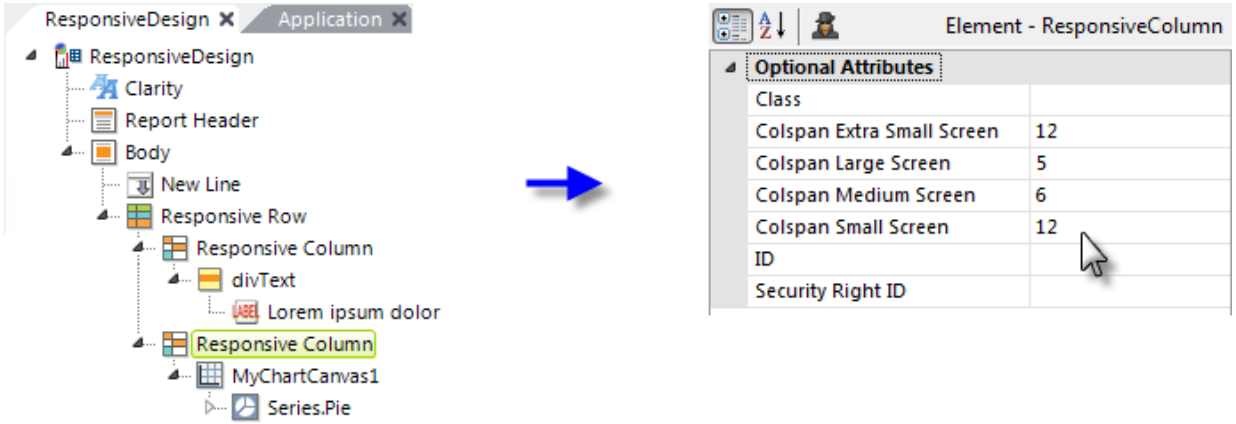


Next, you need to add one or more **Responsive Column** elements to the definition. It has no required attributes.

Its **Colspan** attributes are used to specify how many Responsive Row grid columns each Responsive Column should span, for each viewport size. With a value of *12,* the Responsive Column will span the entire grid, while a value of *0* will cause it to be hidden. In the example above, this column will only be visible for Large Screens, where it will span two grid columns.

If you specify a value for a larger viewport size, make sure you also specify values for smaller viewport sizes or you may get a surprise when a smaller viewport is used.

If more than 12 Responsive Column elements are used, they'll be wrapped into another row below the first 12 elements when displayed.



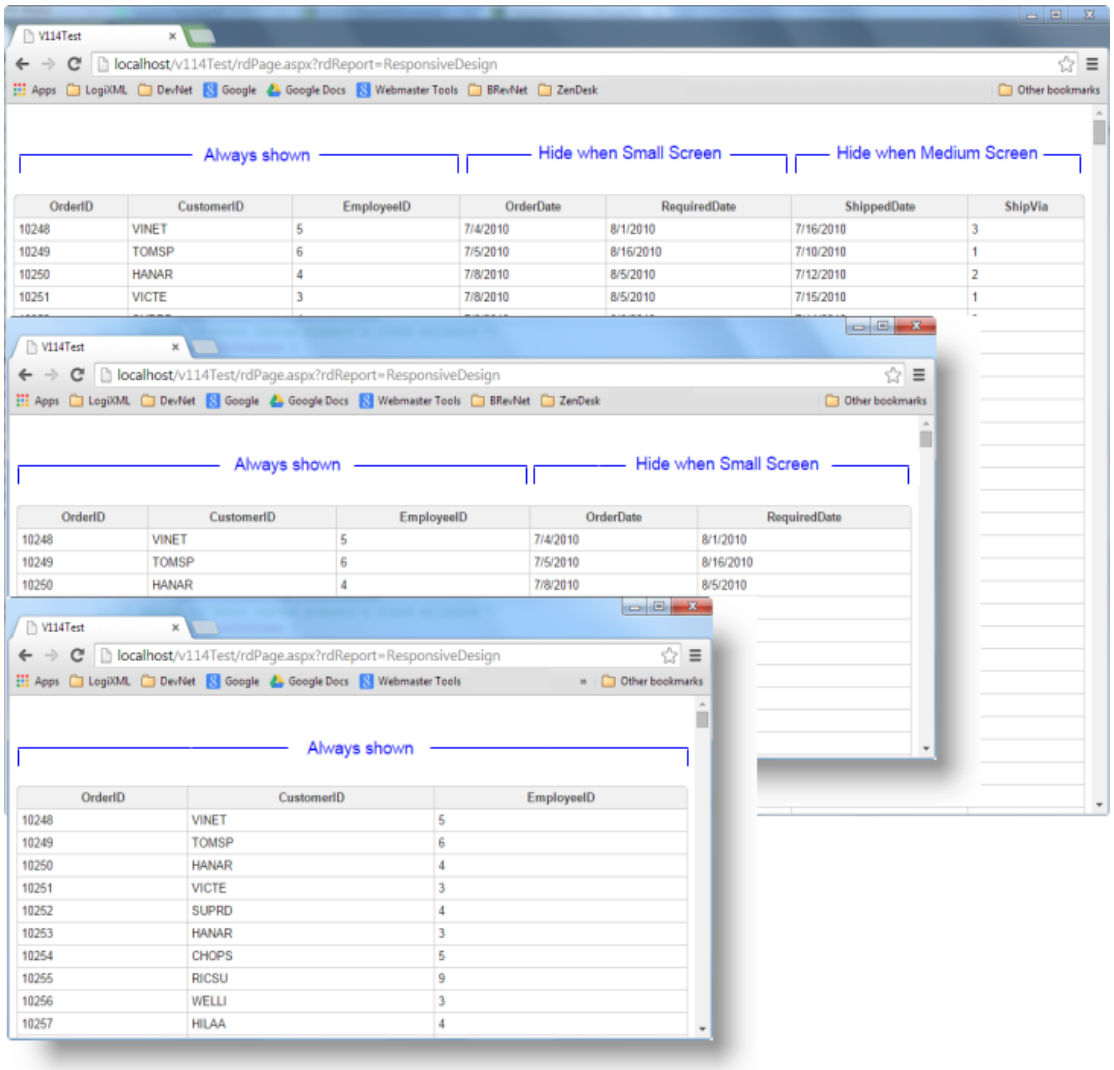
Another Responsive Column element example is shown above. Its Colspan attributes are set to produce the behavior seen in the examples at the beginning of this topic.

You cannot refresh **Responsive Row** and **Responsive Column** elements directly by specifying their IDs in an **Action.Refresh Element** element. However, you can enclose the entire table in a Division and refresh it, or place the Responsive Column's child elements in a Division and refresh it.

Adding "HTML Attribute Params" to your Responsive Rows enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Using Responsive Visibility

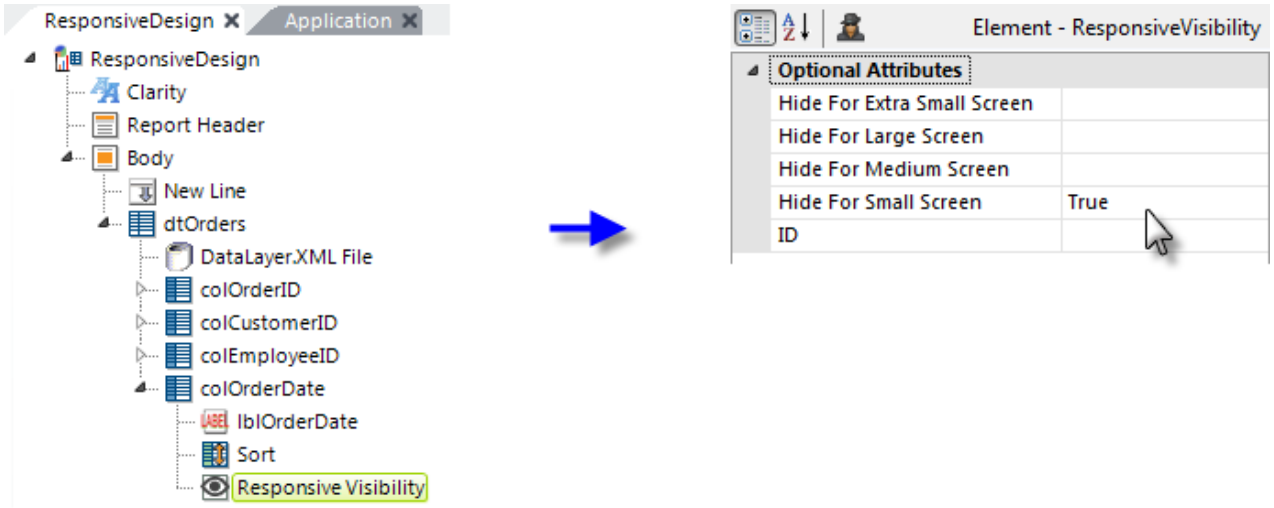
The **Responsive Visibility** element can be the child of a variety of other elements and, as its name implies, it controls their visibility. Developers can use its attributes to specify which viewport sizes cause the parent element to be hidden from view.



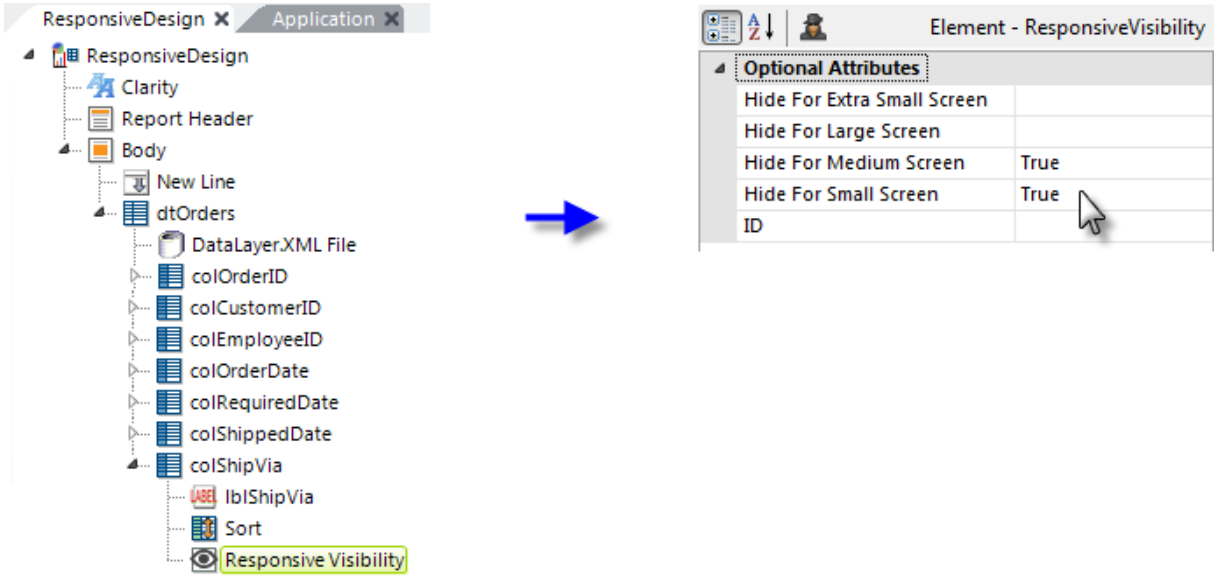
In the example above, Data Table Columns elements have been given child Responsive Visibility elements and, as the browser window gets progressively smaller, additional Data Table columns are hidden.

Implementation

Here's an example showing how the element discussed above is used:



In the example shown above, a **Responsive Visibility** element has been added as a child of a Data Layer Column element. Its attributes are set as shown. The column will not be displayed for Small Screen size viewports.



Now we've added more Data Table columns and another **Responsive Visibility** element.

If you decide to hide an element at a larger viewport size, you will most likely also want to hide it at smaller sizes, otherwise it will disappear then reappear for progressively smaller viewports.

Forecast Methodologies

Forecasting is the process of making statements about events whose actual outcomes (typically) have not yet been observed. A commonplace example might be estimation for some variable of interest at some specified future date. *Prediction* is a similar, but more general term.

The following topics provide information about the **Forecasting** elements that can be used in Logi Info applications:

- Regression
- Prorating
- Time Series Decomposition

 This topic does not apply to Logi Predict.

About Forecasting

The source data used in forecasting may be:

- **Causal**: A dataset ordered by an independent variable, where an independent variable and a dependent variable are numbers. The independent variable may or may not have strong interval, such as 1, 5, 14, 23, 72. For example, the number of unique web site visitors and a count of the clicks on each ad.
- **Time Series**: A natural temporal ordering with a strong interval, where an independent variable (X-axis) is of DateTime type and a dependent variable (Y-axis) is a number. For example, Product Sales by Month.

Which Method is Right?

There is no single "right" forecasting method and the number of usable methods is huge. For example, Wikipedia displays the following list of methods for Time Series forecasting alone:

- Moving Average
- Weighted Moving Average
- Exponential Smoothing
- Autoregressive Moving Average
- Autoregressive Integrated Moving Average (Box-Jenkins)
- Extrapolation
- Linear Prediction
- Trend Estimation
- Growth Curve
- Regression Analysis

Some of these methods are simple to understand, but others are not. For example, the Moving Average method, by itself, is not that useful for forecasting, because its forecasts will have tendency to show just the "average" line. However, the Autoregressive Integrated Moving Average method is difficult for end-users to understand, as it requires the definition a number of constants, including Autoregression Lag, Integration Parameter, and Moving Average Parameter, and there is no "right" way to determine these parameters automatically.

Logi Forecasting elements use three methods of implementing forecasting calculations, which utilize a mix of the methods listed earlier:

- Regression (for Casual data)
- Prorating (for Time Series data)

- Time Series Decomposition (for Time Series data)

Our implementation of these methods is discussed in the following sections.

Regression

Linear prediction is a mathematical operation in which the future values of a dependent variable are estimated as a function of previous samples. The **Forecast.Regression** element implements:

- **Simplelinear regression**, where values are based on a trend line
- **Autoregression**, which predicts an output based on previous outputs, using the "Burg" method
- **Non-linear regression**, which displays the relationship between dependent and independent variables using a curvilinear function and may provide more accuracy. Available curvilinear functions include:
    - Exponential
    - Logarithmic
    - Polynomial2
    - Polynomial3
    - Polynomial4
    - Polynomial5
    - Power

Input Data Requirements

Data should conform to the following requirements:

- Dependent data column data should be Numeric data type
- Dependent data column data should not contain NULL values
- Independent data column may have any data type (if data type is not Numeric, independent data column will be replaced with an integer enumeration (1,2,3,...RowCount).
- Dataset should be in ascending order by independent data column value

- Forecast Length attribute should be less than original row count (if user defines Forecast Length as more than row count, Forecast Length value will automatically be truncated to 20% of RowCount)
- Some of the regression methods require a minimum number of rows (for example, Autoregression requires at least one more row than the value of the AutoRegressive Order attribute, and Polynomial3 requires at least four rows)

Results

As a result of the forecast operation, two new columns will be added to the datalayer. The names of these two columns will be drawn from the element's attributes:

- Forecast Indicator Column ID: this column will contain a boolean flag, set to *True* if the row contains a forecast value
- Forecast Value Column ID: this column will contain the forecast value for each row of the original dataset

The following table shows the effect on the datalayer of a forecast operation:

X-axis	Y-axis	Forecast Indicator	Forecast Value
1	**220**	False	220
2	**230**	False	229
3	**240**	False	241
4	**250**	False	253
5	-	True	260
6	-	True	270

Prorating

The **Forecast.Current Time Period** element produces its forecast by analyzing a value from the last row in the dataset and "prorating" it into the future, through completion of a specific time period.

Input Data Requirements

Data should conform to the following requirements:

- Dependent data column data should have Numeric data type
- Dependent data column data should not contain NULL values
- Independent data column should have DateTime or Date data type
- Dataset should be in ascending order by independent data column value

Method Implementation

An example, assuming Data Column value = "300", Time Period = "Month", and Current DateTime = "01/10/2014":

1. Determine start of date period (current month start): 01/01/2014
2. Determine the end of date period (current month end): 01/31/201
3. Calculate number of seconds between date period start and current date: 01/10/2014 - 01/01/2014 = 86,400 (seconds per day) * 10 days = 864,000 seconds
4. Calculate data value for one second: 300 / 86,4000 = ~0.00034
5. Calculate number of seconds between current date and date period end: 01/31/2014 - 01/10/2014 = 1,814,400 seconds
6. Determine predicted value by multiplying number of seconds between current date and date period end, and data value for one second: 0.00034 * 1,814,400 = ~616
7. Save predicted value to column identified by Forecast Difference Column ID attribute.

8. Save sum of predicted value and actual data value (300 + ~616) to column identified by Forecast Value Column ID attribute.

Results

As a result of the forecast operation, three new columns will be added to the datalayer. The names of these columns will be drawn from the element's attributes:

- Forecast Difference Column ID: this column will contain the value of the difference between the starting data value and each prorated value
- Forecast Indicator Column ID: this column will contain a boolean flag, set to *True* if the row contains a forecast value
- Forecast Value Column ID: this column will contain the forecast value for each row of the original dataset (if this value is left blank, the forecast values will be added to the value of the dependent Data Column)

Time Series Decomposition

The decomposition of time series data is a statistical method that deconstructs it into *notional* components. Several decomposition methods exist and the Logi **Forecast.Time Period Decomposition** element uses "Decomposition Based on Rates of Change".

This is an important technique for all types of time series analysis, especially for seasonal adjustment. It seeks to construct, from an observed time series, a number of component series (that could be used to reconstruct the original by additions or multiplications) where each of these has a certain characteristic or type of behaviour. For example, Monthly or Quarterly economic time series are usually decomposed into:

- a Trend Component *T* that reflects the long term progression of the series
- a Cyclical Component *C* that describes repeated but non-periodic fluctuations, possibly caused by the economic cycle
- a Seasonal Component *S* that reflects seasonality (seasonal variation)
- an Irregular Component *I* that describes random, irregular influences (or "noise"). Compared to the other components it represents the residuals of the time series.

The equation used in the method is: $Y = T * S * C * I$

Input Data Requirements

Data should conform to the following requirements:

- Dependent data column data should have Numeric data type
- Dependent data column data should not contain NULL values
- Independent data column should have DateTime or Date data type
- Dataset should be in ascending order by independent data column value

- Forecast Length attribute should be less than original row count (if user defines Forecast Length as more than row count, Forecast Length value will automatically be truncated to 20% of RowCount)
- Source rows count should be not less than season count

Data Validation

It's necessary to check the dataset for inconsistencies before running the forecasting algorithm. "Empty spaces" inside time series are not allowed, a validation process checks the dataset in advance. If an iteration includes more than three "empty spaces", such as "Jan, Feb, July, Aug...", validation returns an error. Otherwise, validations fills missing time periods with the average of the values for the previous and next period. For example, the following dataset, which is missing data for February:

Iteration	Source Data
January	10
March	30
April	40

becomes:

Iteration	Source Data	Validated Data
January	10	10
February		20

Iteration	Source Data	Validated Data
March	30	30
April	40	40

Time Series Iteration

Time Series Decomposition requires defined date iteration, which is necessary for the determination of the seasonal component. These date iterations are used:

Iteration	Season Count	Season Determination
Hour	24	Hour
Day	7	Day of Week
Week	5	Week of Month
Month	12	Month
Quarter	4	Quarter
Year	1	No season

Method Implementation

1. "Deseasonalizing" the data: Remove short-term fluctuations from the data, so that the longer-term trend and cycle components can be more clearly identified. These short-term fluctuations include both seasonal patterns and irregular variations. They can be removed by calculating an appropriate Moving Average (MA) for the series, which should contain the same number of periods as there are in the seasonality that we want to identify. If the season count is an even number, the moving averages are not really centered in the middle of the seasons. We calculate a Centered Moving Average (CMA) and it represents the deseasonalized data.
2. Determine Seasonical Factor (SF), which is the ratio of the actual value to the deseasonalized value.
3. Create Season Index (SI) for each season, which is the normalized mean of the seasonal factors for the selected season.
4. Estimate Long Term Trend from the deseasonalized data using linear regression.
5. The cyclical component of a time series is the extended wavelike movement about the long-term trend. It is measured by the Cycle Factor (CF), which is the ratio of the Centered Moving Average to the Long Term Trend.
6. Determine forecast for Cyclical component by using AutoRegression with Order equal to the Count of seasons (use 4 if no season).
7. Calculate forecast for time series.

Forecast Calculations Example

The following table displays the calculation used for a forecast, assuming Quarterly data for a 1-year forecast:

	Period	Data	MA	CMA	Trend By CMA	CF	SF	SI	Prediction
1	02/01/65	181.50	-	-	215.15	-	-	-	-
2	05/01/65	296.70	-	-	215.05	-	-	-	-
3	08/01/65	266.20	240.95	240.79	214.95	1.12	1.11	1.10	264.44
4	11/01/65	219.40	240.63	235.89	214.84	1.10	0.93	0.88	207.00
5	02/01/66	180.20	231.15	222.61	214.74	1.04	0.81	0.79	176.83
6	05/01/66	258.80	214.08	204.36	214.64	0.95	1.27	1.24	253.01
7	08/01/66	197.90	194.65	190.51	214.53	0.89	1.04	1.10	209.22
8	11/01/66	141.70	186.38	185.86	214.43	0.87	0.76	0.88	163.10
9	02/01/67	147.10	185.35	191.14	214.33	0.89	0.77	0.79	151.83
10	05/01/67	254.70	196.93	203.94	214.22	0.95	1.25	1.24	252.49
11	08/01/67	244.20	210.95	215.05	214.12	1.00	1.14	1.10	236.17
12	11/01/67	197.80	219.15	220.59	214.02	1.03	0.90	0.88	193.57
13	02/01/68	179.90	222.03	222.63	213.91	1.04	0.81	0.79	176.84
14	05/01/68	266.20	223.23	224.03	213.81	1.05	1.19	1.24	277.36
15	08/01/68	249.00	224.83	223.73	213.71	1.05	1.11	1.10	245.70
16	11/01/68	204.20	222.63	221.73	213.61	1.04	0.92	0.88	194.57
17	02/01/69	171.10	220.83	216.51	213.50	1.01	0.79	0.79	171.99
18	05/01/69	259.00	212.20	207.41	213.40	0.97	1.25	1.24	256.79
19	08/01/69	214.50	202.63	-	213.30	**0.93**		1.10	218.49
20	11/01/69	165.90	-	-	213.19	**0.92**		0.88	171.91
21	**02/01/70**				**213.09**	**0.93**		0.79	**158.13**
22	**05/01/70**				**212.99**	**0.97**		1.24	**254.52**
23	**08/01/70**				**212.88**	**1.00**		1.10	**232.64**
24	**11/01/70**				**212.78**	**1.01**		0.88	**189.52**

Results

As a result of the forecast operation, two new columns will be added to the datalayer. The names of these two columns will be drawn from the element's attributes:

- Forecast Indicator Column ID: this column will contain a boolean flag, set to *True* if the row contains a forecast value
- Forecast Value Column ID: this column will contain the forecast value for each row of the original dataset

Logi OLAP

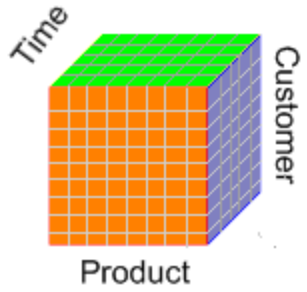
Logi Info includes an OLAP-related family of elements, including the OLAP Grid super-element, that allows users to "slice and dice" OLAP cube data at runtime.

The following topics provide instructions for using these elements:

- Logi OLAP Elements
- Connecting to Data
- Creating a Basic OLAP Grid
- Other OLAP Grid-related Elements
- Using the OLAP Grid
- Using the OLAP Table
- Using OLAP Elements with Logi XOLAP
- Re-using the MDX Query
- Using Two-Dimensional Data
- Adding OLAP Grid Visualizations to Dashboards
- Customizing OLAP Grid Appearance

Logi OLAP has been deprecated; the related elements are still supported, for now, and will work but they're no longer available in Logi Studio.
Information about connecting to data is available in *Connect to MS SQL Server Analysis Services*.

About Logi OLAP



On-Line Analytical Processing (OLAP) is a technology for quickly providing answers to analytical queries that are dimensional in nature and can be viewed from multiple perspectives. Typical applications of OLAP include analysis for sales, marketing, business process

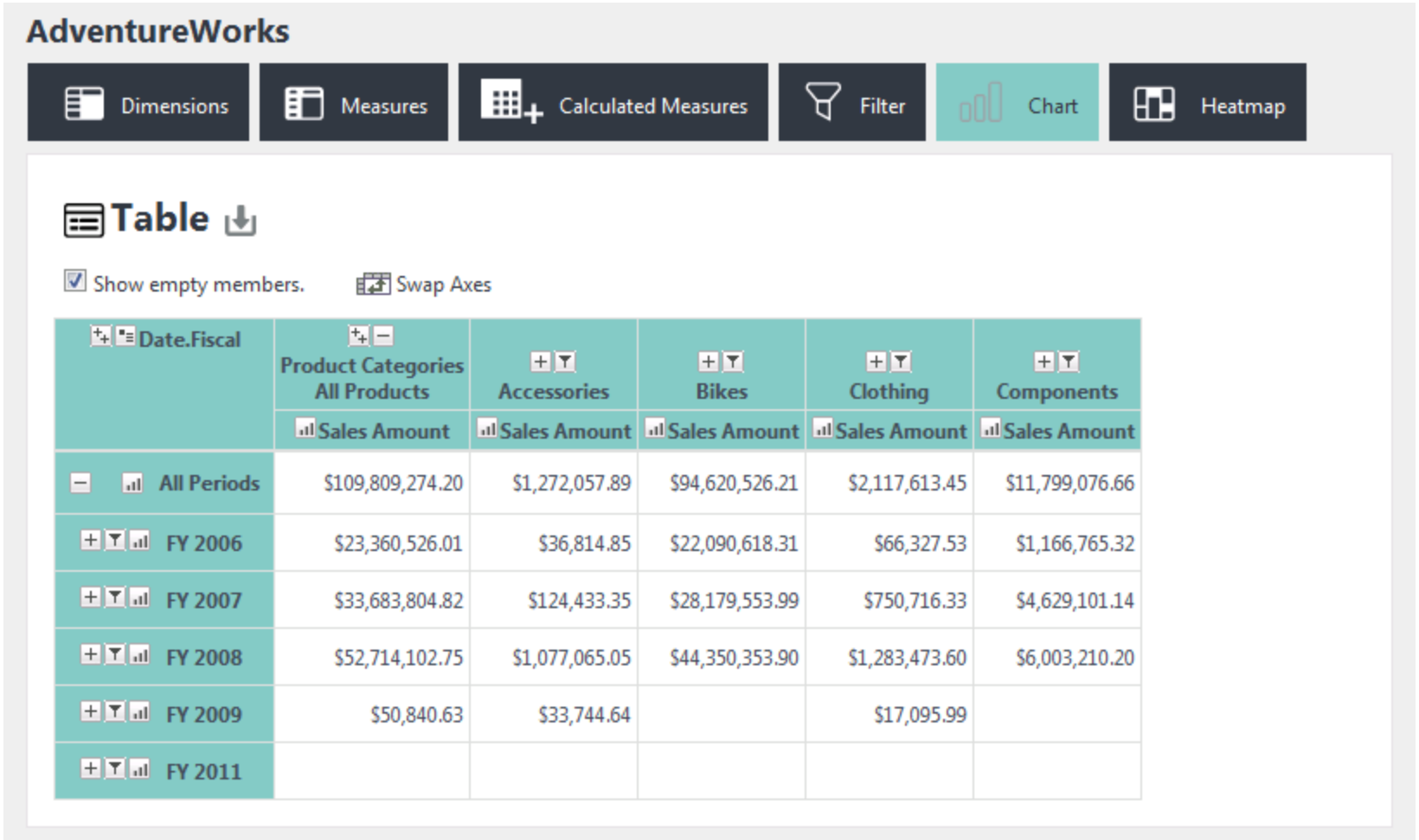
management, financial reporting, and similar areas. Some organizations deal with massive amounts of transactional data and, in order to optimize their data for analysis, keep pre-aggregated data in multi-dimensional OLAP "cubes" to get faster answers to critical business questions.

Logi Analytics' zero-footprint **OLAP** elements make using OLAP analysis easy, by allowing your users to quickly make sense of corporate data so they can get immediate answers to their "why" questions. Logi Info's OLAP elements provide front-end tools that can be used to query, analyze, and display data from OLAP cubes created with Microsoft SQL Server Analysis Services (SSAS).

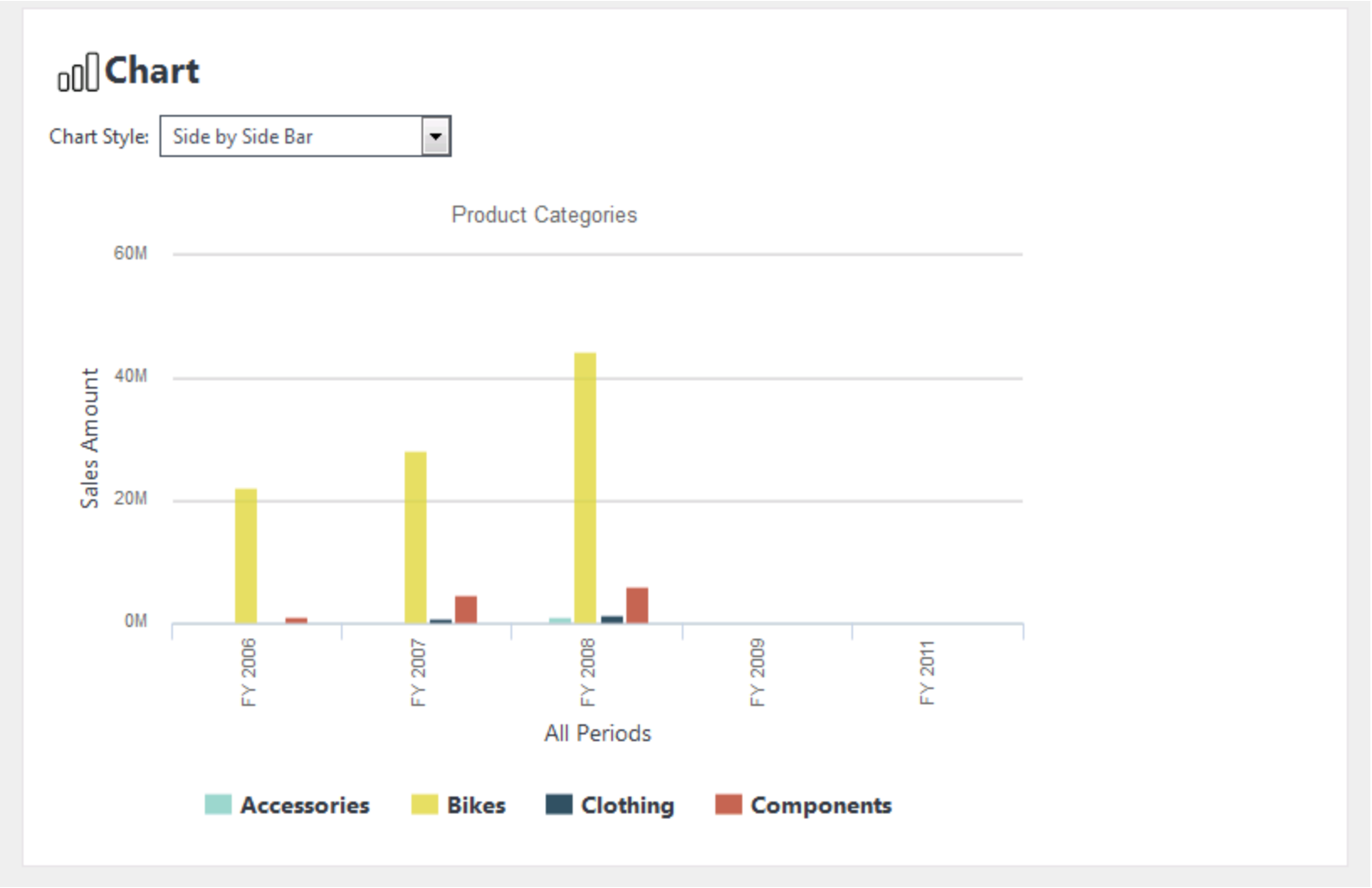
Logi OLAP only works with Microsoft SQL Server Analysis Services; it *does not* support other data cubes.

Logi OLAP - Elements

Logi Info's family of OLAP-related elements provides both an **OLAP Grid** super-element, a tool with a built-in user interface that allows users to filter, analyze, and visualize data, and an **OLAP Table** element similar to a standard Data Table. The following examples focus on the OLAP Grid; the OLAP Table is discussed in "Logi OLAP - Using the OLAP Table" on page 514.



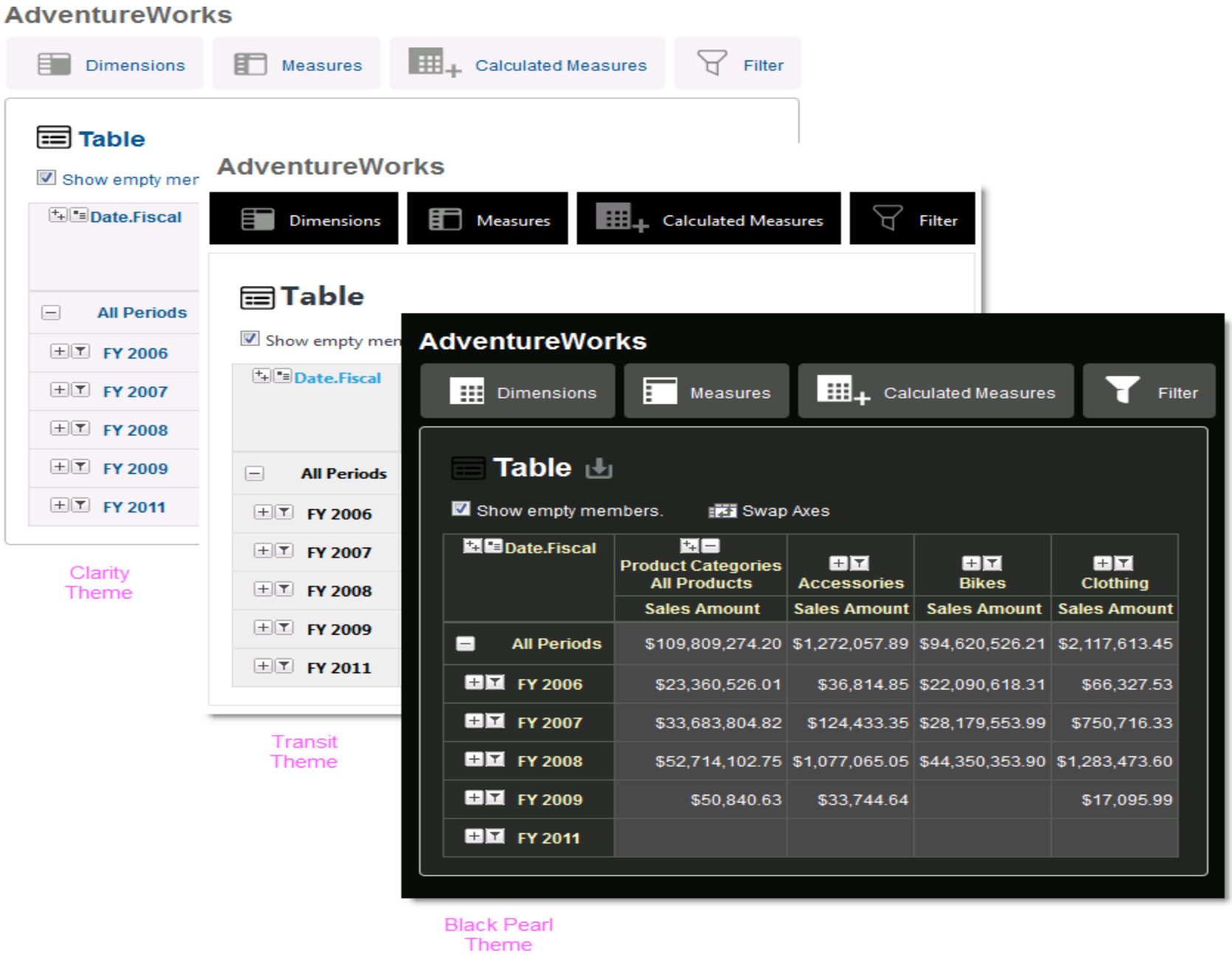
Date.Fiscal	Product Categories All Products	Accessories	Bikes	Clothing	Components
	Sales Amount	Sales Amount	Sales Amount	Sales Amount	Sales Amount
All Periods	$109,809,274.20	$1,272,057.89	$94,620,526.21	$2,117,613.45	$11,799,076.66
FY 2006	$23,360,526.01	$36,814.85	$22,090,618.31	$66,327.53	$1,166,765.32
FY 2007	$33,683,804.82	$124,433.35	$28,179,553.99	$750,716.33	$4,629,101.14
FY 2008	$52,714,102.75	$1,077,065.05	$44,350,353.90	$1,283,473.60	$6,003,210.20
FY 2009	$50,840.63	$33,744.64		$17,095.99	
FY 2011					



An example of the OLAP Grid in use is shown above. The buttons across the top show and hide feature panels, including: Dimensions, Measures, Calculated Measures, Filter, Chart, and Heatmap. By default, the OLAP Grid includes all of the buttons shown, but it can be configured, or restricted by security, to include fewer buttons and panels, if desired. At runtime, users can:

- Select Dimensions for the left and top axes
- Select Measures
- Select Filters for slicing the data
- Drill-down on Dimension members
- Create charts
- Drill-through to underlying data

With all these possibilities, it may be important for the user to be able to save these settings between sessions. Info developers can create a Process Task that uses **Procedure.SaveOlapGrid** to do this. A saved OLAP Grid can be reloaded with the `rdOgLoadSaved` parameter, for example: *rdOgLoadSaved = myFilename.xml* Users' settings are automatically maintained during the session but can be cleared by calling the OLAP Grid element's report with the `rdOgReset` parameter, for example: *rdOgReset = True* The example shown above uses the standard **Signal** theme to style the grid. Here are some of the other standard themes in action:


AdventureWorks
Dimensions
Measures
Calculated Measures
Filter
Table
Show empty mer
Date.Fiscal
All Periods
FY 2006
FY 2007
FY 2008
FY 2009
FY 2011
Clarity Theme
AdventureWorks
Dimensions
Measures
Calculated Measures
Filter
Table
Show empty mem
Date.Fiscal
All Periods
FY 2006
FY 2007
FY 2008
FY 2009
FY 2011
Transit Theme
AdventureWorks
Dimensions
Measures
Calculated Measures
Filter
Table
Show empty members.
Swap Axes
Date.Fiscal
Product Categories All Products
Accessories
Bikes
Clothing
Sales Amount
All Periods $109,809,274.20 $1,272,057.89 $94,620,526.21 $2,117,613.45
FY 2006 $23,360,526.01 $36,814.85 $22,090,618.31 $66,327.53
FY 2007 $33,683,804.82 $124,433.35 $28,179,553.99 $750,716.33
FY 2008 $52,714,102.75 $1,077,065.05 $44,350,353.90 $1,283,473.60
FY 2009 $50,840.63 $33,744.64 $17,095.99
FY 2011
Black Pearl Theme

As you can see, the user interface looks quite different depending on the theme that's applied by the developer. It is *very important* that the value of the **ID** attribute for the OLAP Grid element be *unique* throughout the entire application. This is because Logi Info saves the last state of the OLAP Grid in a user's session information based on the grid's ID. Consequently, if two different report definitions contain an OLAP Grid with the same ID value, the application will display the same grid for both reports.

Caveats and Limitations

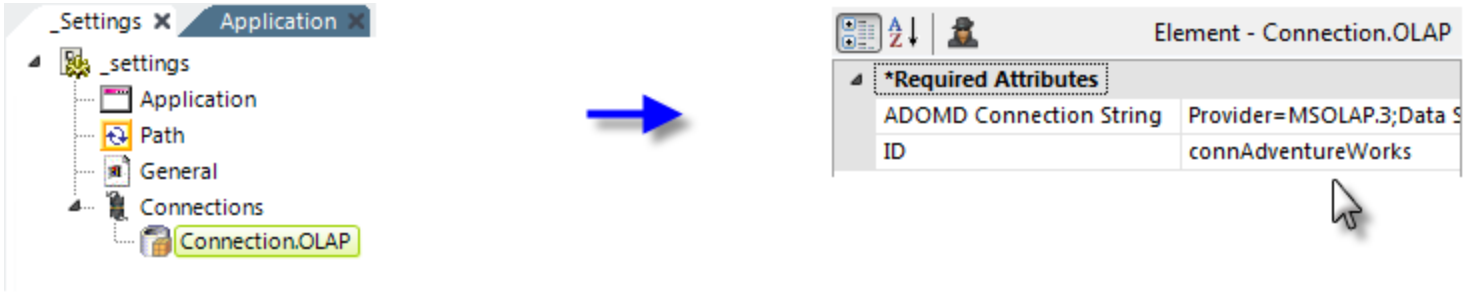


- Chart Canvas charts are used by default by the OLAP Grid in all *new applications*. Older Logi applications that are *upgraded* to v12 will use the classic static charts for their OLAP Grids. To force upgraded apps to use Chart Canvas charts, add the constant `rdFavorChartCanvas = True` to your _Settings definition.
- The maximum number of rows that can be displayed in the OLAP Grid's Data Table is *2,000*.
- We *do not* recommend that you use any Logi OLAP elements within an embedded sub-report.

Logi OLAP - Connecting to Data

These are the basic steps for creating an application using an OLAP Grid and connecting to the data:

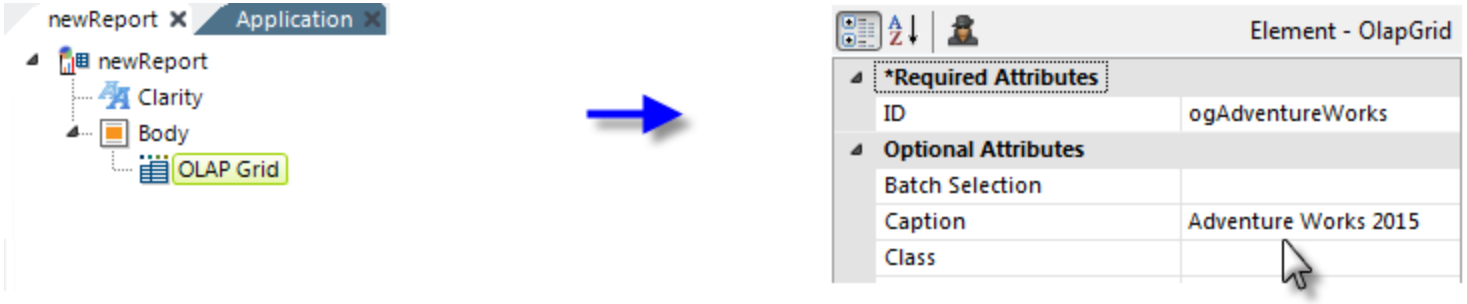
1. In Logi Studio, create a new application as usual and register it with IIS.



2. In the _Settings definition, add a **Connection.OLAP** element, as shown above, and set its attributes. The **ADOMD Connection String** attribute value can be a bit more complicated than the usual database connection string and there are other data provider requirements as well. For more information, see *Connect to MS SQL Server Analysis Services*.

Logi OLAP - Creating a Basic OLAP Grid

Now that you have a connection to Analysis Services configured, you can add the elements that comprise your OLAP Grid:



In your report definition, add an **OLAP Grid** element beneath the Body element, as shown above.

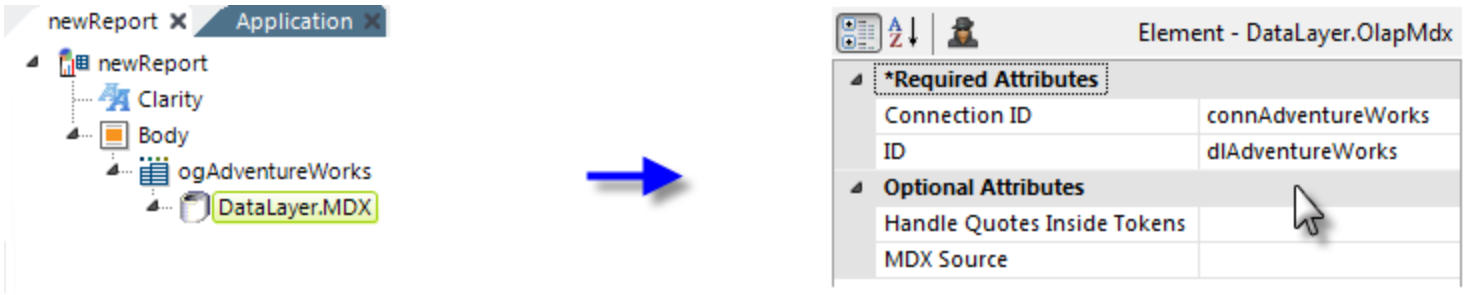
The OLAP Grid attributes are:

Attribute	Description
ID	(Required) Specifies an element ID, unique with the definition.
Batch Selection	Specifies whether UI changes made by the user will be applied immediately (*False*) or only when the user clicks an "OK" button (*True*). Deferring until the button is clicked may speed up the user experience by reducing the number of retrieval requests sent to the server.
Caption	Specifies a title that will appear above the grid.
Class	Specifies Cascading Style Sheet classes to be used by the element. Classes will also be used by all child ele-

Attribute	Description
	ments that don't have their own classes set. A class set here may be overridden by a built-in theme.
Drilldown All	Specifies if each dimension in the table will show an extra "Drilldown All" button (*True*). Clicking this button causes a drilldown on all items below the dimension that have the "+" drilldown icon. The default value is *False*.
Hide Excel Export	Specifies if the Export: Excel button will be hidden from view. The default value is *False*.
Level Indent	Specifies the amount of indentation, in pixels, used when displaying levels of a hierarchy in a table. The default is *2 pixels*.
OLAP Cell Colors	Specifies whether or not to use foreground and background colors, defined in SSAS, for Calculated Members. The default value is *False*.
Pick Member Properties	Specifies whether the user will be able to pick Member Properties to be displayed in columns next to the Dimension column. Member Properties are selected by clicking an icon shown next to the Dimension name in the table. Selected Member Property columns only appear when there is at least one member that has the property and Member Properties can only be selected for Dimensions added to the *left* side of the table. The default value is *False*.
Saved OLAP Grid Folder	OLAP Grid configuration options selected by the user at runtime can be saved for reuse in a file, using the Procedure.Save Olap Grid element. Saved options can be reloaded by passing the "saved" filename in the rdOgLoadSaved request parameter. This attribute specifies the fully-qualified file system path to the folder, beneath the application root folder, that contains the saved files. Function tokens may be used to facilitate

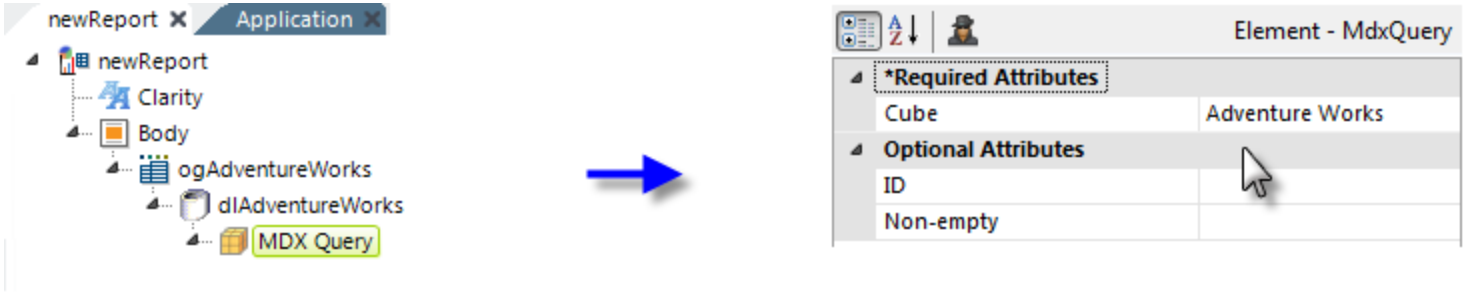
Attribute	Description
	this, for example: `@Function.AppPhysicalPath~\SavedOlapGrids`
Security Right ID	When specified, access to this element can be controlled with Logi Security. Specify the ID of a Right defined in the application's _Settings Security element. Only users with matching rights will be able to see the element.
Show Dimension Attributes	Specifies whether the table will show Dimension Attributes in addition to Dimensions and Dimension Hierarchies. Set to *True* to also show Dimension Attributes. The default value is *False*.
Template Modifier File	Specifies the name, with file extension, of a *Template Modifier Files* to be applied to the OLAP Grid at runtime. This is an XML file that can be used to change the underlying template used to create the OLAP Grid. Template Modifier files can be in any folder accessible to the application; if a folder isn't specified, the _SupportFiles folder is assumed.
Width	Specifies the width of the displayed OLAP Grid.
Width Scale	Specifies the width units of the Width attribute value: either *px* for pixels or % for percent.

A minimal configuration, like the one shown in the example above is all that's needed to display a usable grid. Once the OLAP Grid element is configured, you're ready to add its child elements:



Beneath the OLAP Grid element, add a **DataLayer.MDX** element and set its attributes as shown above. The **Connection ID** attribute value should be the ID of the Connection.OLAP element you configured at the start of this process.

The Attribute Panel identifies the element as a DataLayer.OlapMdx element.



Beneath the datalayer, add an **MDX Query** element and set its attributes as shown above. Your OLAP Grid is now functional. Click the Preview tab at the bottom of the Workspace to view it.

Logi OLAP - Other OLAP Grid-related Elements

The topic describes the other elements available for use with the OLAP Grid.

MDX Query Child Elements

In "Logi OLAP - Creating a Basic OLAP Grid" on page 495, we saw that the **MDX Query** element was used to automatically build the OLAP MDX query. These child elements can be used with it to customize the query:

Element	Description
Mdx Axis Dimension	Specifies the Dimensions that will be shown across the top or left side. Because the OLAP Grid allows user selection of Dimensions, only "default" Dimensions need to be specified. There must be at least one Dimension for the left axis to see any OLAP results. You may specify a Hierarchy in the DimensionName attribute. For example: `[MyDimension].[MyHierarchy]` You may have any number of Mdx Axis Dimension elements for either Axis.
Mdx Drill-Down (child of Mdx Axis Dimension)	Specifies a single member of the dimension to be drilled into/ expanded. Multiple MdxDrilldown elements may be used. The Value attribute must be a member of the parent element's dimension. Separate each level with the period character. When a member has spaces in the name, it must be enclosed in brackets.

Element	Description
	Begin the Value with the dimension name. Example for the first level of the Customers dimension: `[Customers].[All Customers]` When there are multiple dimensions on a single axis, the members of each axis must be separated by the asterisk character. Example with Customers: `[Customers].[All Customers].[USA].[CA]*[Education Level].[All Education Level].[Graduate Degree]*[Gender].[All Gender].[M]` Specify a Value of *AllMembers* to expand all members of the dimension.
Mdx Drill-Up (child of Mdx Axis Dimension)	Collapses a Dimension member that has been drilled-down. This may be used to undo drill-downs caused by an Mdx Drill-Down of *AllMembers*.
Mdx Calculated Measure	Adds an additional Measure from an MDX formula. The MDX Formula basically follows script syntax and typically references other measures. For example: `VAL((Measures.[Store Sales] - Measures.[Store Cost]) / Measures.[Store Sales])` Add an Mdx Measure element to display the calculated Measure.
Mdx Filter	Filters or slices values out of the OLAP cell set. Each Dimension to be filtered should have an Mdx Filter Dimen-

Element	Description
Dimension	sion element. Add Mdx Filter Value child elements for each member to be included or excluded for cell calculations. Dimensions cannot be used in Mdx Filter Dimension and Mdx Axis Dimension elements at the same time.
Mdx Measure	Specifies the Measures that will be shown in the cells of an OLAP Grid. Multiple Mdx Measure elements may be used. The cube's default measures are shown when none are specified. Cells are formatted according to the format specified in the cube definition. Localization is applied to these formats according to the client browser's language setting.

OLAP Grid Child Elements

The following elements are available for configuring the OLAP Grid user interface and behavior. Panel-related elements can be used to hide panels; without them all panels are displayed.

Element	Description
OLAP Calculated Measure Panel	Specifies the configuration of the OLAP Grid's Calculated Measures panel. The Calculated Measures panel is not available when working with a XOLAP Cube.
OLAP	Specifies the configuration of the OLAP Grid's Chart panel.

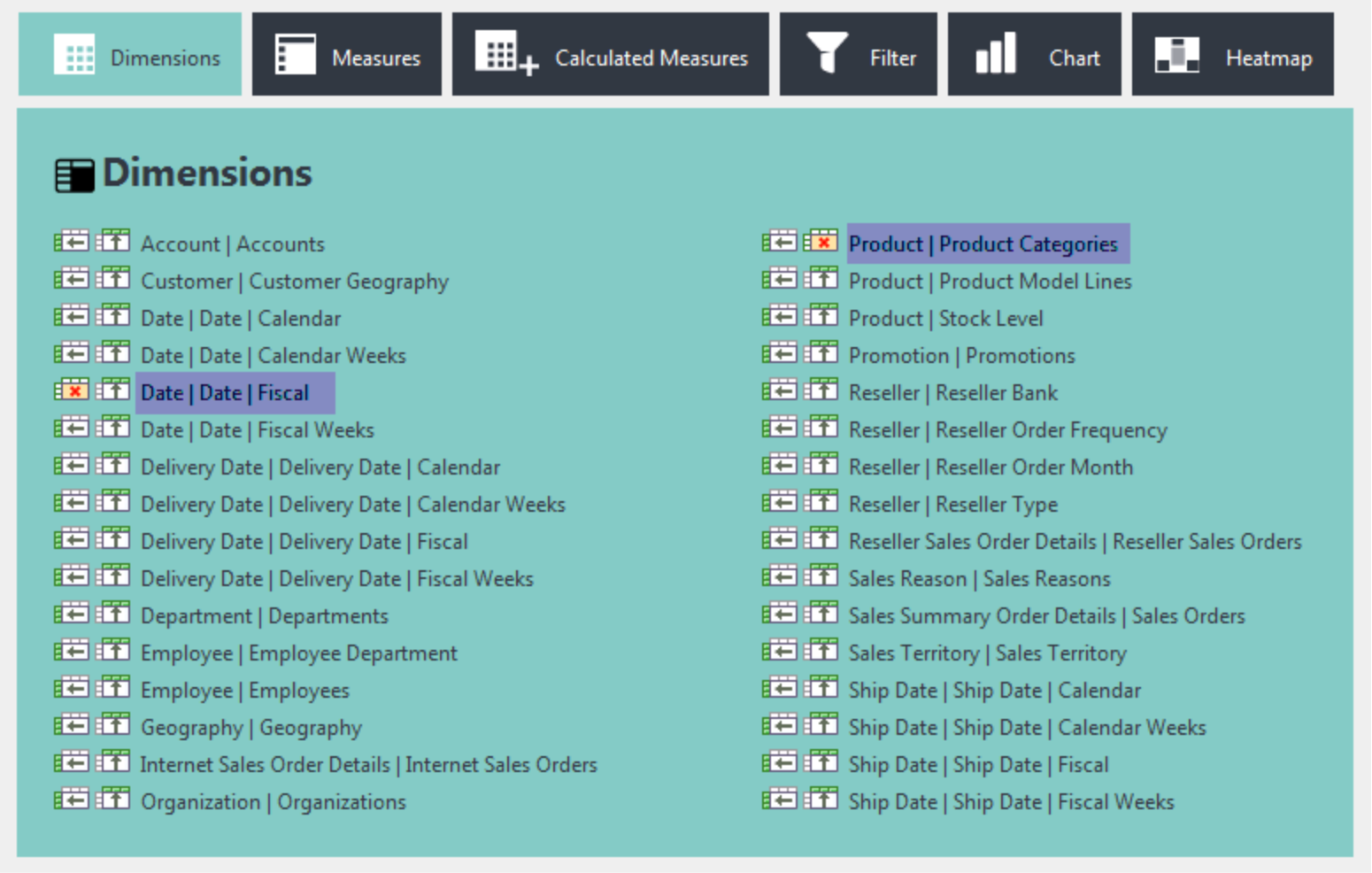
Element	Description
Chart Panel	
OLAP Dimension Panel	Specifies the configuration of the OLAP Grid's Dimensions panel.
Allowed Dimension (child of OLAP Dimension Panel)	Specifies the cube dimensions available in the OLAP Grid's Dimensions panel. Multiple elements may be used. Use the Security Right ID attribute to set the available dimensions based on the user's roles. If none of these elements are present, *all* cube dimensions will be available for selection.
OLAP Drill-through	This element enables "drill-through" on cell values. It adds drill-through links that appear as the user hovers over each cell. When clicked, a new page is displayed with a basic Data Table listing the drill-through rows with all the columns used to build the cube. The element is not available for calculated measures or when there are multiple dimensions on either the left or top axis. The Caption attribute sets title of the drillthrough page. The MaxRows attribute sets the maximum number of rows that will be returned shown in the drillthrough table. The default is *1000* rows.

Element	Description
	Drillthrough is functional when connecting to Analysis Services 2005 and later. It is *not* functional for AS2000 or XOLAP Cubes.
OLAP Filter Panel	Specifies the configuration of the OLAP Grid's Filters panel.
Allowed Filter Dimensions (child of OLAP Filter Panel)	Specifies the cube dimensions available in the OLAP Grid's Filters panel. Multiple elements may be used. Use the Security Right ID attribute to set the available dimensions based on the user's roles. If none of these elements are present, *all* cube dimensions will be available for filtering.
OLAP Heat Map Panel	Specifies the configuration of the OLAP Grid's Heat Map panel.
OLAP Measure Panel	Specifies the configuration of the OLAP Grid's Measures panel.
Allowed Measure (child of OLAP Measure Panel)	Specifies the cube measures available in the OLAP Grid's Measures panel. Multiple elements may be used. Use the Security Right ID attribute to set the available measures based on the user's roles. If none of these elements are present, *all* cube measures will be available for selection.

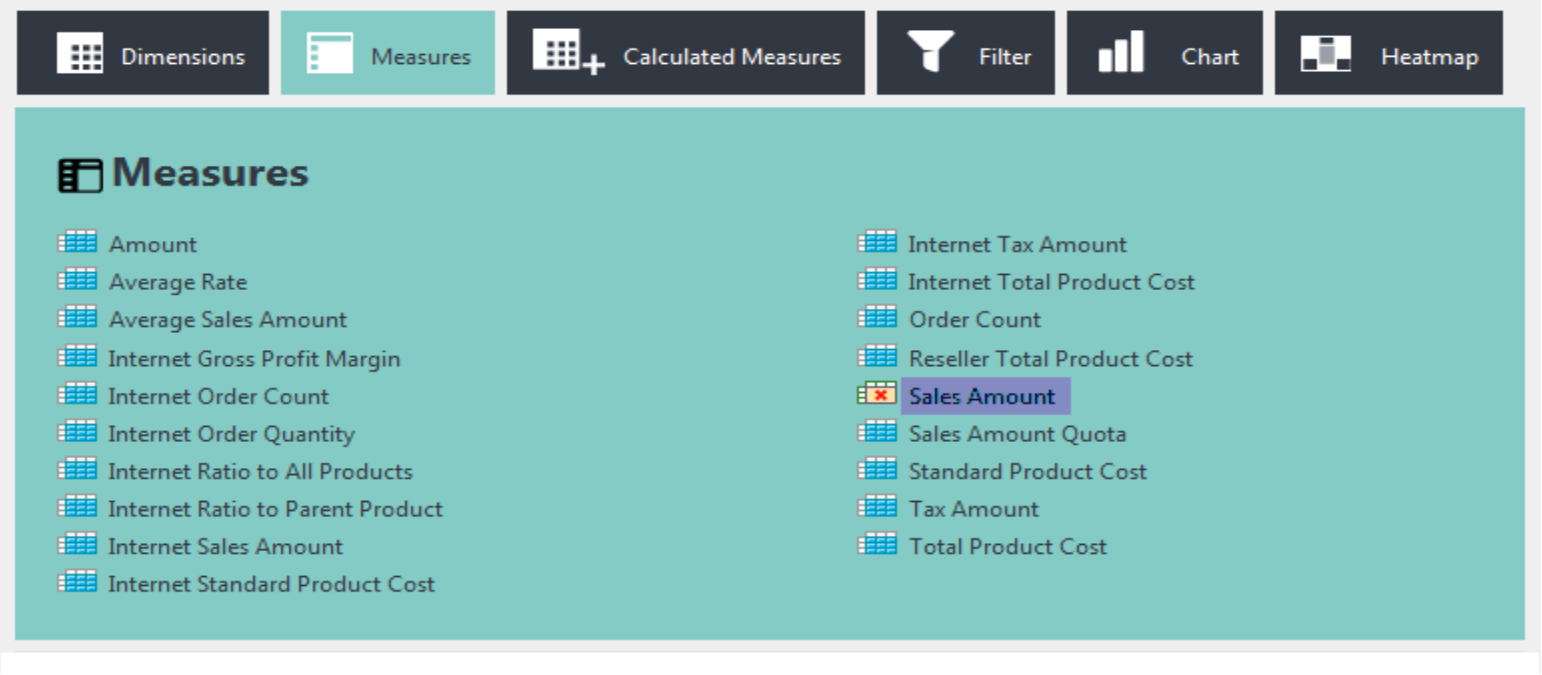
Element	Description
OLAP Menu Panel	Specifies the configuration of the OLAP Grid's Menu panel.
OLAP Table Panel	Specifies the configuration of the OLAP Grid's Table panel.
Generated Plug-in Call	Specifies a Plug-in method to be run. The method is called after its parent element is parsed into its XML definition.

Logi OLAP - Using the OLAP Grid

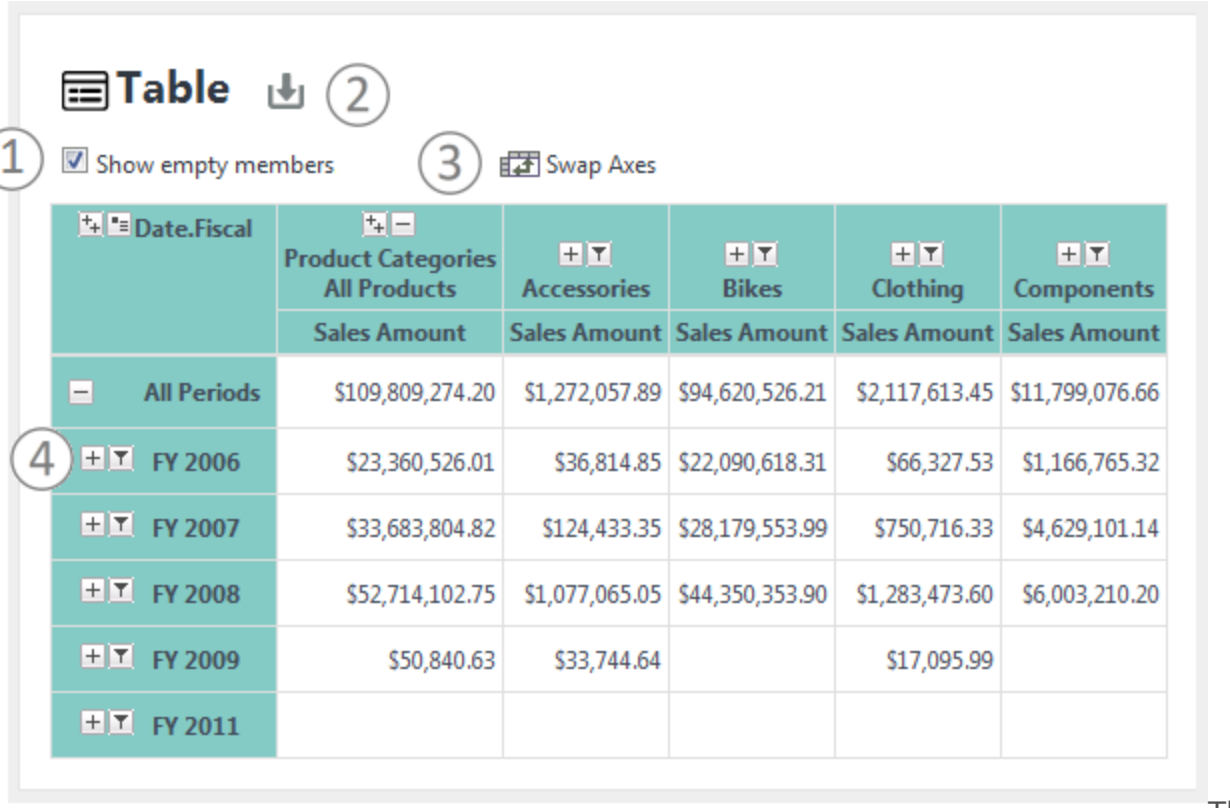
The following exercise will guide you through the steps of configuring the grid dynamically. The instructions and screen shots that follow assume you have made a connection to the AdventureWorks database but are applicable for any data source.



1. Preview or browse your OLAP Grid application.
2. Click the Dimensions button at the top to open the **Dimensions** panel, shown above. This is where users select dimensions for the left and top axes of the grid. In the Dimensions panel, select the data for *Date Fiscal* for the left axis and *Product Categories* for the top axis, as shown above. Depending on the grid's Batch Selection attribute setting, the changes will be applied, and the table updated, automatically, or you may have to click the Update Table button. You can click the Dimensions button again (or another button) to hide the Dimensions panel.

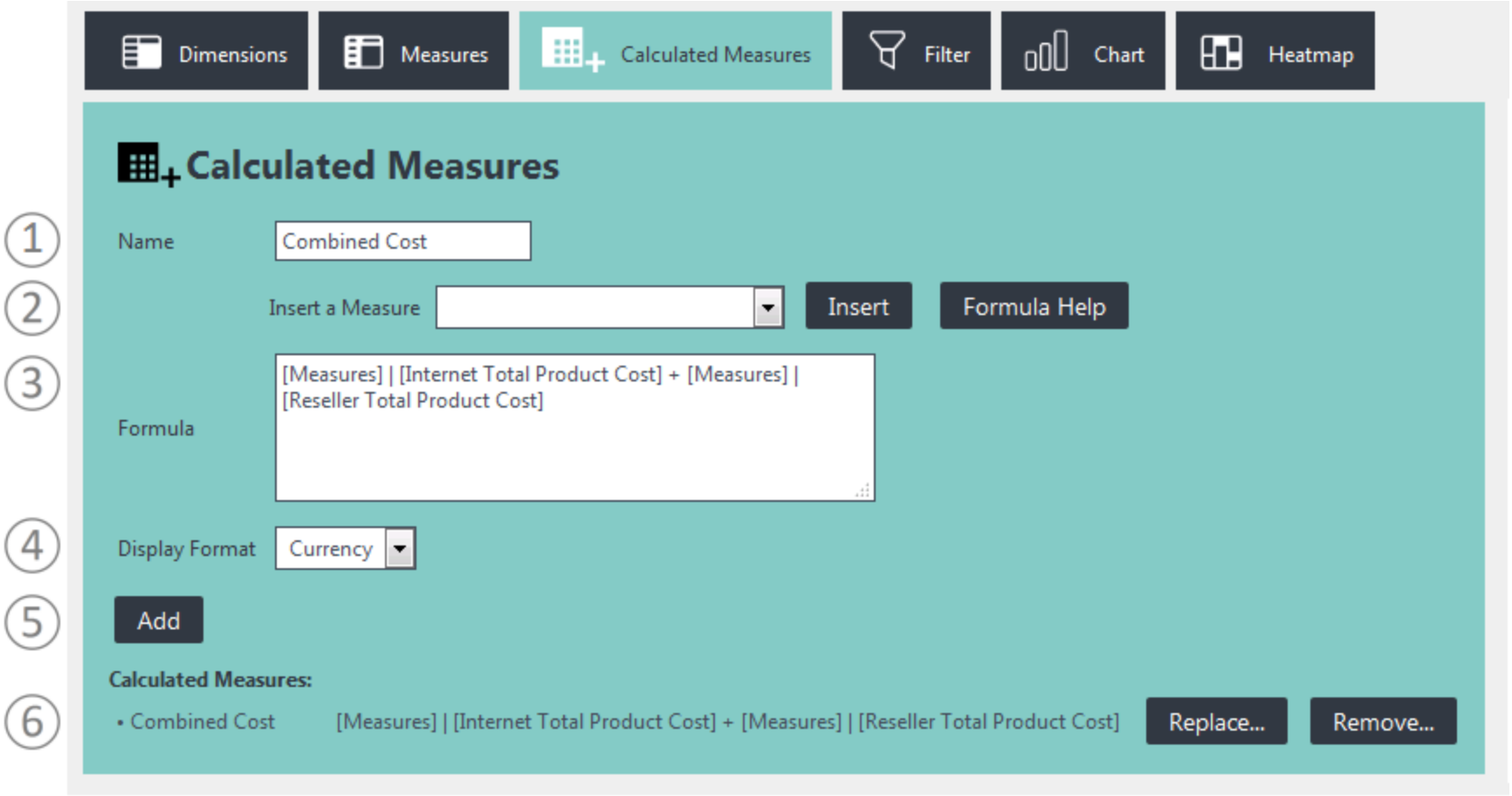


3. Click the Measures button to display the **Measures** panel, shown above. Select the measures *Sales Amount* item, as shown above. Click the Update Table button, if necessary. Click the Measures button again to hide the panel.



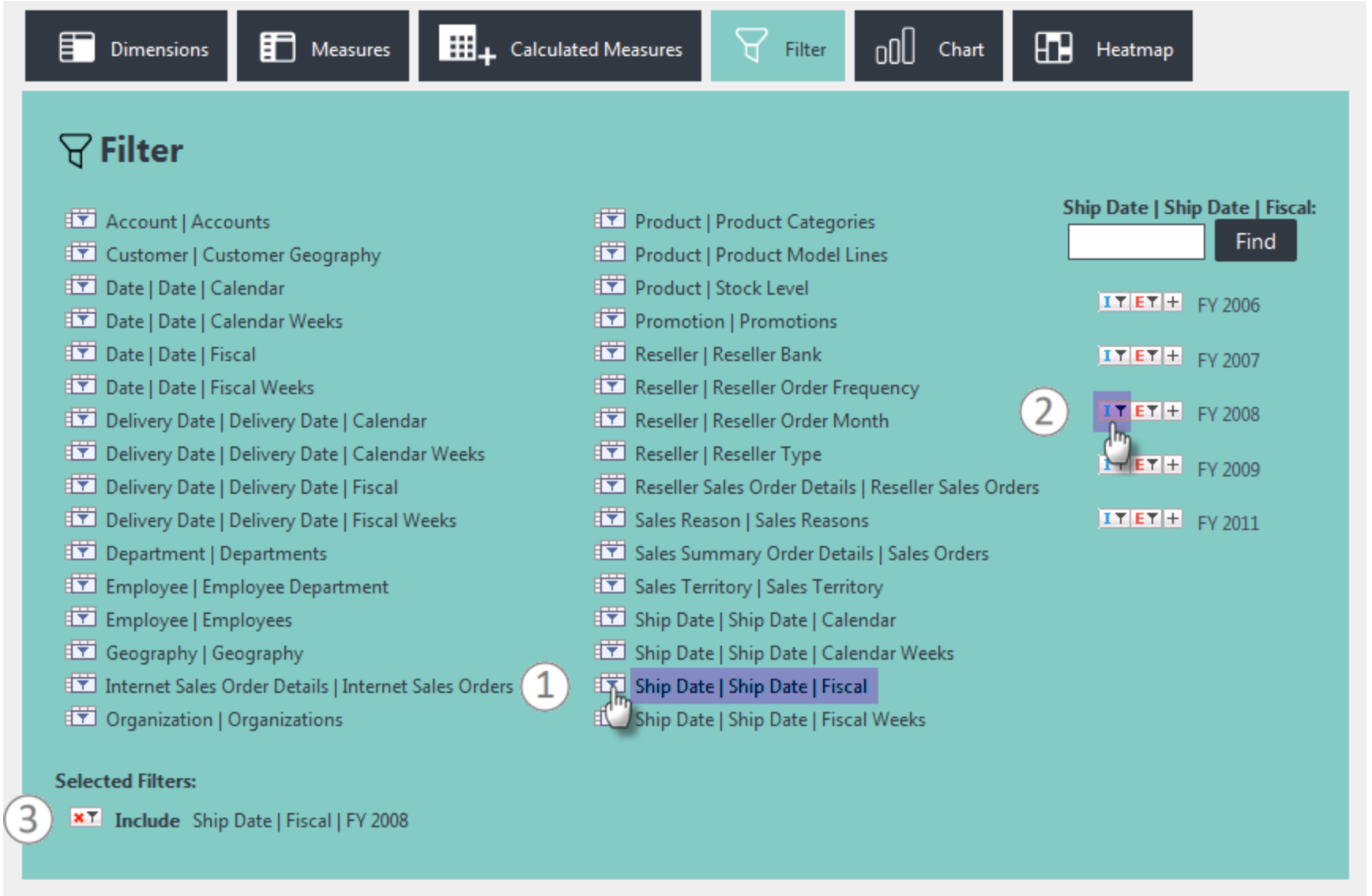
The resulting table should look something like the example shown above. Note these features: (1) Show or hide any empty members using this check box.
(2) Click this icon to for Excel export, if the OLAP Grid element has been configured to display it.
(3) Click this icon to swap the axes to view the data in a different relationship.
(4) Click these member icons to drill-down or filter the data.

4. Click the Calculated Measures button to open the Calculated Measures panel.



In the Calculated Measures panel, shown above, enter "Combined Cost" in the **Name** text box (1). Measures can be selected in the "Insert a Measure" select list (2) and added to the formula by clicking "Insert". The measures are inserted into the **Formula** text box (3), and/or you can type them in directly, to create a formula. Click the Formula Help button to view available functions and valid syntax. Then choose the **Display Format** (4) which can be Number, Percent, or Currency. When your formula is complete, click the **Add** button (5). The calculated measure will appear at the bottom of the panel (6) and it will appear immediately

as a new column in your table. Click the calculated measure name (6) to place it back into the controls for editing, and click the **Replace...** or **Remove...** button to update or delete it. Click the Calculated Measures button again to hide the panel.

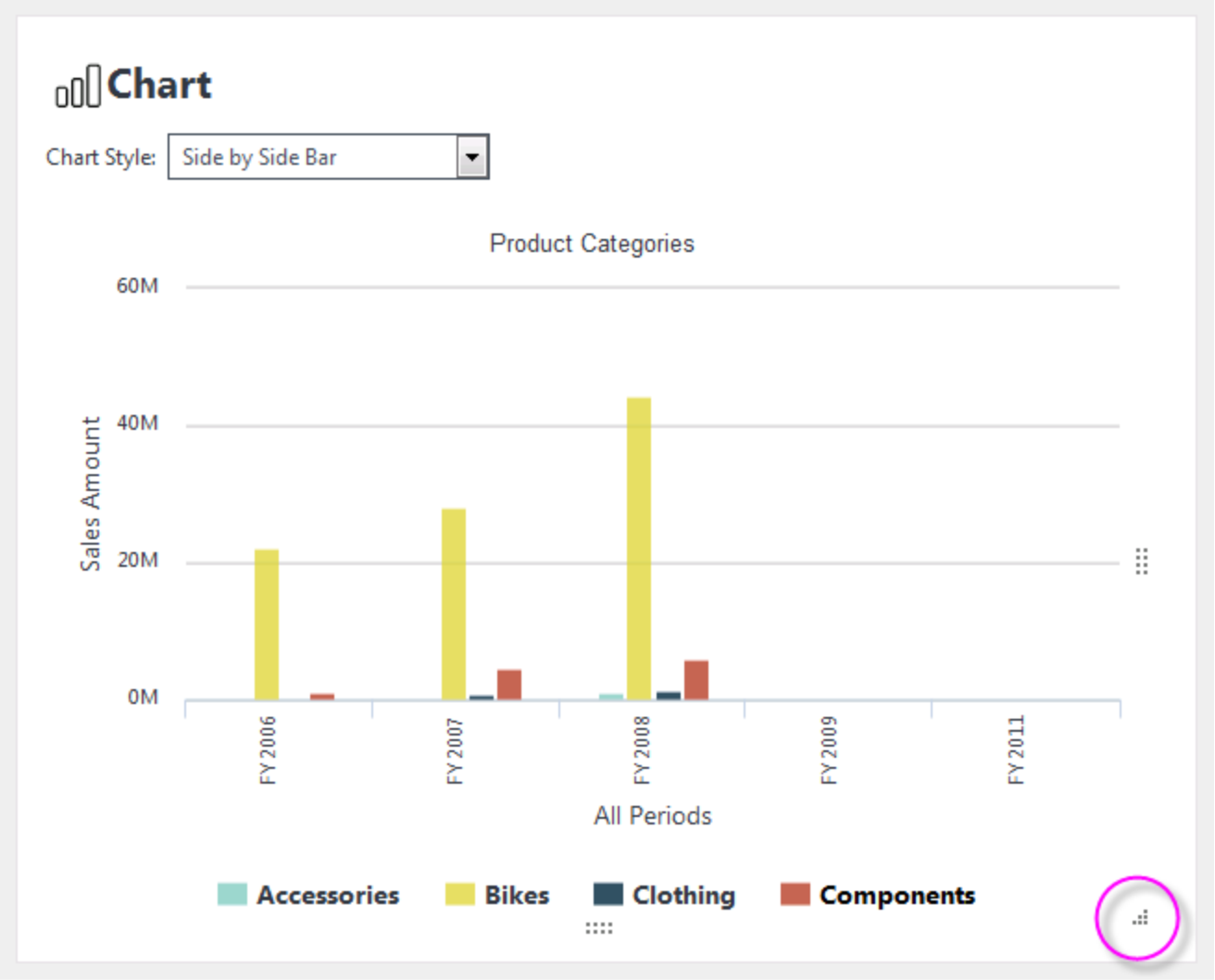

Dimensions
Measures
Calculated Measures
Filter
Chart
Heatmap
Filter
Account | Accounts
Customer | Customer Geography
Date | Date | Calendar
Date | Date | Calendar Weeks
Date | Date | Fiscal
Date | Date | Fiscal Weeks
Delivery Date | Delivery Date | Calendar
Delivery Date | Delivery Date | Calendar Weeks
Delivery Date | Delivery Date | Fiscal
Delivery Date | Delivery Date | Fiscal Weeks
Department | Departments
Employee | Employee Department
Employee | Employees
Geography | Geography
Internet Sales Order Details | Internet Sales Orders
Organization | Organizations
Product | Product Categories
Product | Product Model Lines
Product | Stock Level
Promotion | Promotions
Reseller | Reseller Bank
Reseller | Reseller Order Frequency
Reseller | Reseller Order Month
Reseller | Reseller Type
Reseller Sales Order Details | Reseller Sales Orders
Sales Reason | Sales Reasons
Sales Summary Order Details | Sales Orders
Sales Territory | Sales Territory
Ship Date | Ship Date | Calendar
Ship Date | Ship Date | Calendar Weeks
Ship Date | Ship Date | Fiscal
Ship Date | Ship Date | Fiscal Weeks
1
Ship Date | Ship Date | Fiscal:
Find
FY 2006
FY 2007
FY 2008
FY 2009
FY 2011
2
Selected Filters:
3
Include Ship Date | Fiscal | FY 2008

5. Now experiment with filtering the measures. Click the **Filter** button to open its panel, shown above, where you can filter your data by setting **Include** and **Exclude** parameters. For example, you can limit your view of sales data across regions of various departments to that which includes a fiscal 2008 shipping date by setting an Include Filter, as follows:

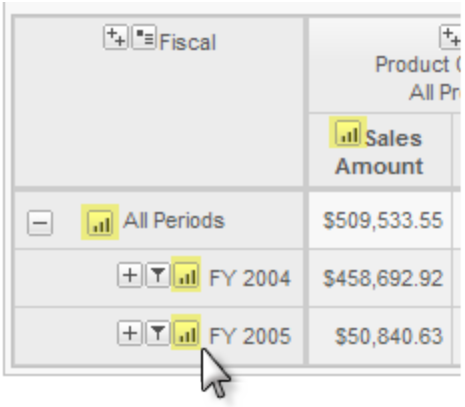
   Click the **Ship Date.Fiscal** measure (1) and, after a moment, the filter selector area will appear on the right. Click the "I" icon, for *Include*, adjacent to the FY 2008 item (2). This will add the filter into the "Selected Filters" area (3), which lets you see what filters are in effect and where you can also delete a filter. The table automatically updates to reflect the filters chosen. There's also an "E" icon, for *Exclude*, in the filter selector area.

   Click the Filter button to hide the panel.

6. Charts and Heatmaps, based on *all* of the current table data, are initially displayed just by clicking the **Chart** and **Heatmap** buttons to open their respective panels:



Once they're displayed, charts can be customized by changing their **Chart Style**, and by using their *re-sizing* handles (circled above) which may not be visible until the mouse cursor hovers over them. A variety of chart styles (Stacked, Line, Spline, etc.) are available using the selection list and charts and heatmaps can be drilled down into through all grid layers.

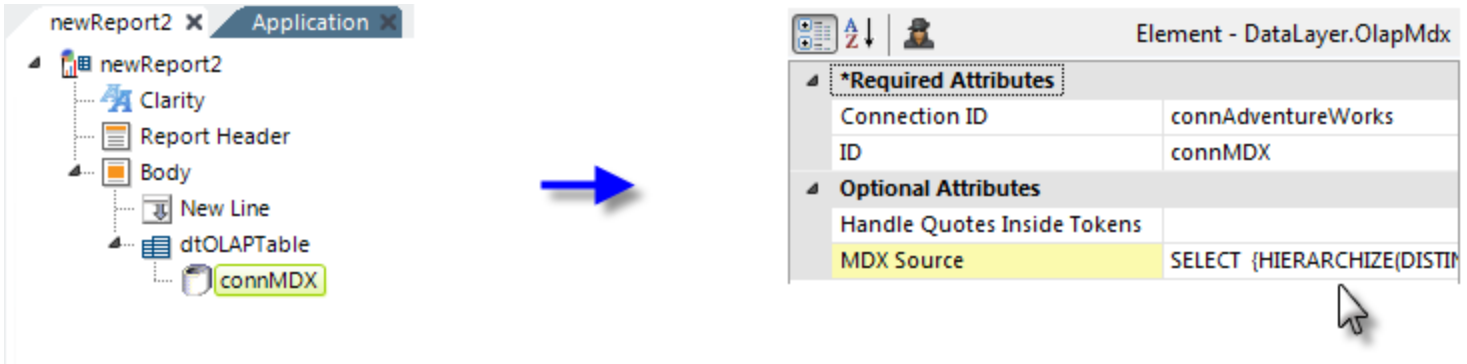


In addition, once the chart or heatmap has been displayed, additional small icons appear in each row and column in the table, as shown above; these allow you to pick which measure and dimension layer to chart.

That concludes this exercise in configuring an OLAP grid at runtime.

Logi OLAP - Using the OLAP Table

The **OLAP Table** element displays OLAP data in a Data Table, without any of the built-in user input controls for filtering and drilling down into the data found in the OLAP Grid. This element is useful if you just want to display data or you prefer to add your own user input elements. Like the OLAP Grid, the OLAP Table gets its data from a **DataLayer.MDX** element, and an **MDX Query** element can be used with the datalayer to automatically generate the query.



Or, if you prefer, instead of using the MDX Query element, you can enter your own query in the datalayer's **MDX Source** attribute, as shown above. This attribute is only functional when used with an OLAP Table and is ignored when the datalayer is used with the OLAP Grid. The OLAP Table elements has all of the attributes of a Data Table and also includes the **Level Indent** and **OLAP Cell Colors** attributes from the OLAP Grid. There are no Data Table Column child elements needed when using the OLAP Table; it includes an "auto columns" functionality and automatically shows all columns in the datalayer.

Logi OLAP - Using OLAP Elements with Logi XOLAP

Logi **XOLAP** technology brings the analytical power of data cubes to your relational data and data from other diverse sources such as web services and files. XOLAP is similar to OLAP but doesn't rely on an OLAP data cube. The OLAP Grid and OLAP Table elements are able, however, to use data from a Logi XOLAP cube. Logi Info includes a special super-element, the **Dimension Grid**, for use with XOLAP data, so why would you want to use an OLAP Grid instead? The Dimension Grid is designed to be less complicated to use and doesn't support some features, such as multi-dimension sorting and multiple dimensions on both axes, that are found in the OLAP Grid. So, for some use-cases, the OLAP Grid allows more complex analysis of the XOLAP data. See "Logi XOLAP" on page 525and "Dimension Grid" on page 535for more information.

Logi OLAP - Re-using the MDX Query

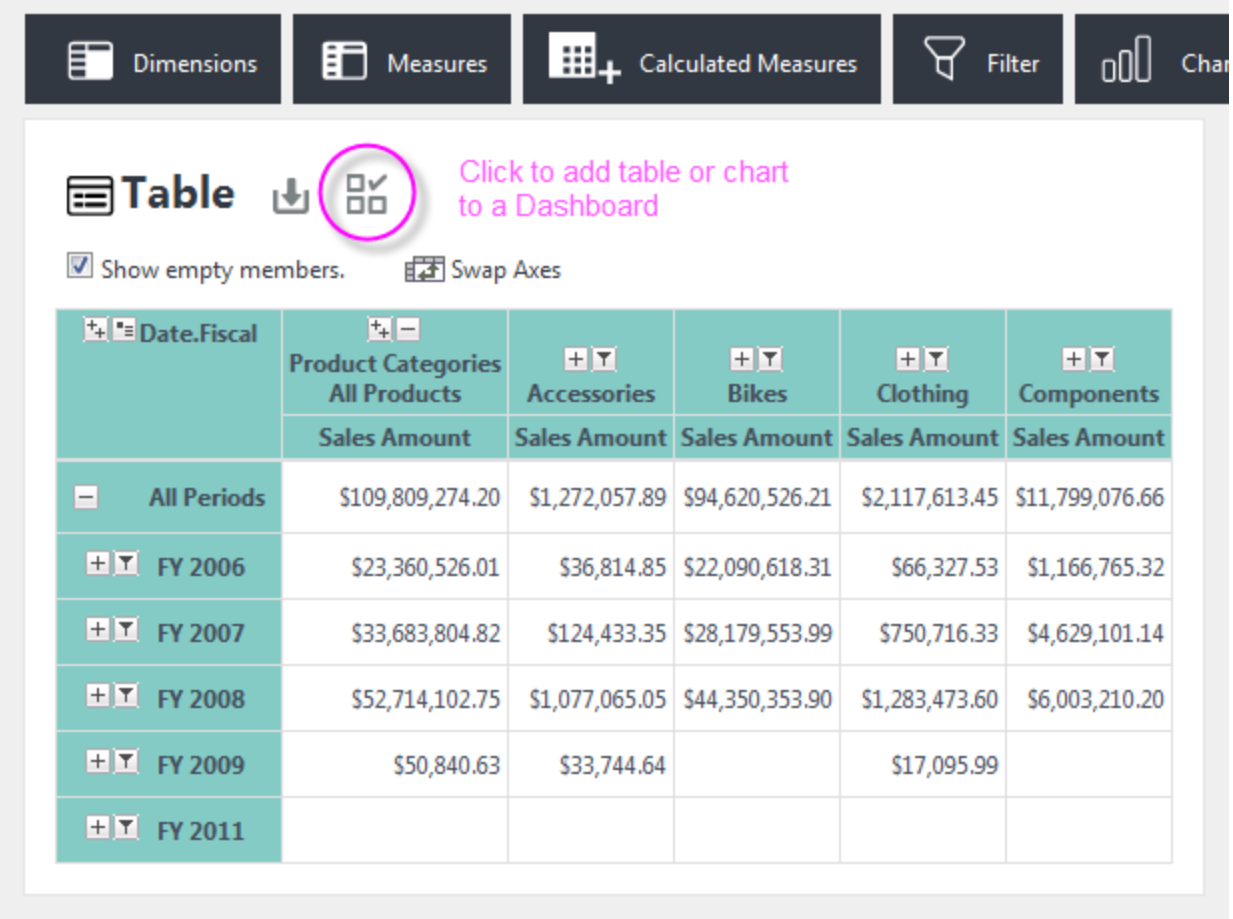
Many of Logi's other data visualization components are capable of "2-D" representation of OLAP cube data, using MDX queries. So it may be useful to be able to access the MDX query used in an OLAP Grid in one report for use with non-OLAP elements in another report. Each time the OLAP Grid is manipulated during runtime, a session variable is updated to reflect the resulting MDX query. This session variable, rdOlapLastMdxQuery, is available to other reports, Logi or otherwise. In a Logi report, it would be accessed using the token `@Session.rdOlapLastMdxQuery~`.

Logi OLAP - Using Two-Dimensional Data

It's possible to use two-dimensional data from SSAS cubes for use in standard Data Tables, charts and other visualizations. This is done using **Connection.OLEDB** and **DataLayer.SQL** with an MDX query statement in its Source attribute. Unlike your Connection.OLAP configuration, the connection string for Connection.OLEDB will need to include the User ID and Password needed to log into the database. Logi Studio includes a tool, the MDX Query Builder, that can help you construct such a query for use with DataLayer.SQL. For more information, see *The MDX Query Builder*.

Logi OLAP - Adding OLAP Grid Visualizations to Dashboards

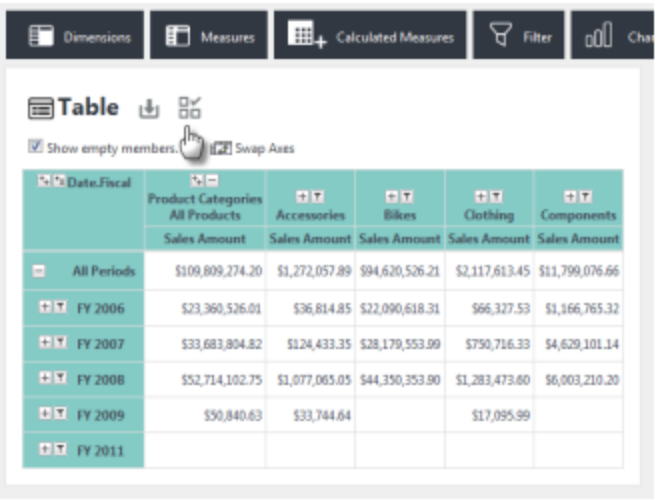
An exciting level of interaction between the **OLAP Grid** and the **Dashboard** element is available. Users working with an OLAP Grid in one report, at runtime, can generate a table or chart and then, with one mouse click, add it as a new panel in an existing Dashboard in another report.



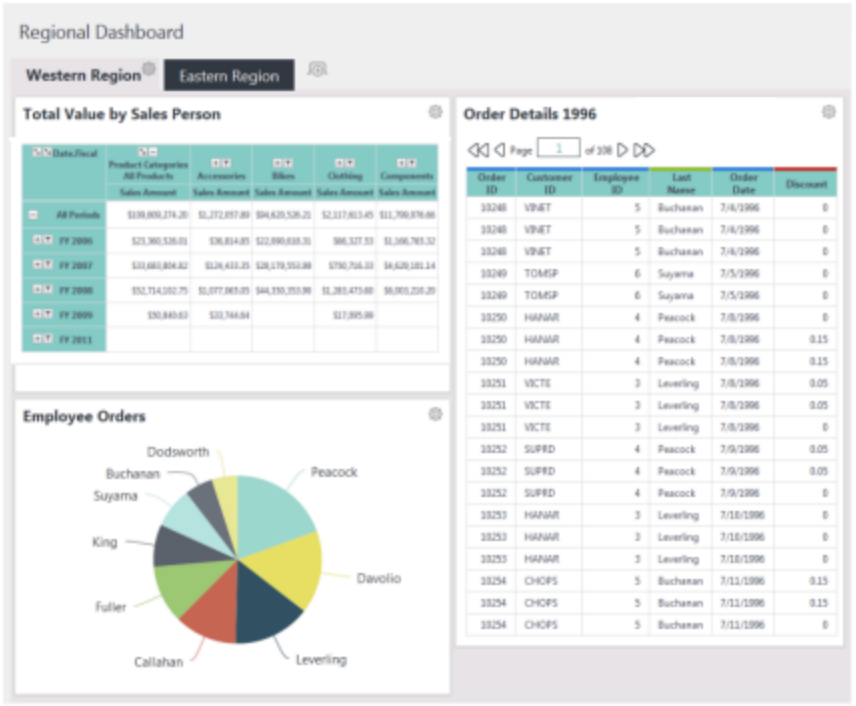
Date.Fiscal	Product Categories All Products	Accessories	Bikes	Clothing	Components
	Sales Amount	Sales Amount	Sales Amount	Sales Amount	Sales Amount
All Periods	$109,809,274.20	$1,272,057.89	$94,620,526.21	$2,117,613.45	$11,799,076.66
FY 2006	$23,360,526.01	$36,814.85	$22,090,618.31	$66,327.53	$1,166,765.32
FY 2007	$33,683,804.82	$124,433.35	$28,179,553.99	$750,716.33	$4,629,101.14
FY 2008	$52,714,102.75	$1,077,065.05	$44,350,353.90	$1,283,473.60	$6,003,210.20
FY 2009	$50,840.63	$33,744.64		$17,095.99	
FY 2011					

When properly configured, OLAP Grid tables and charts will display an **Add To Dashboard** icon, as shown above.

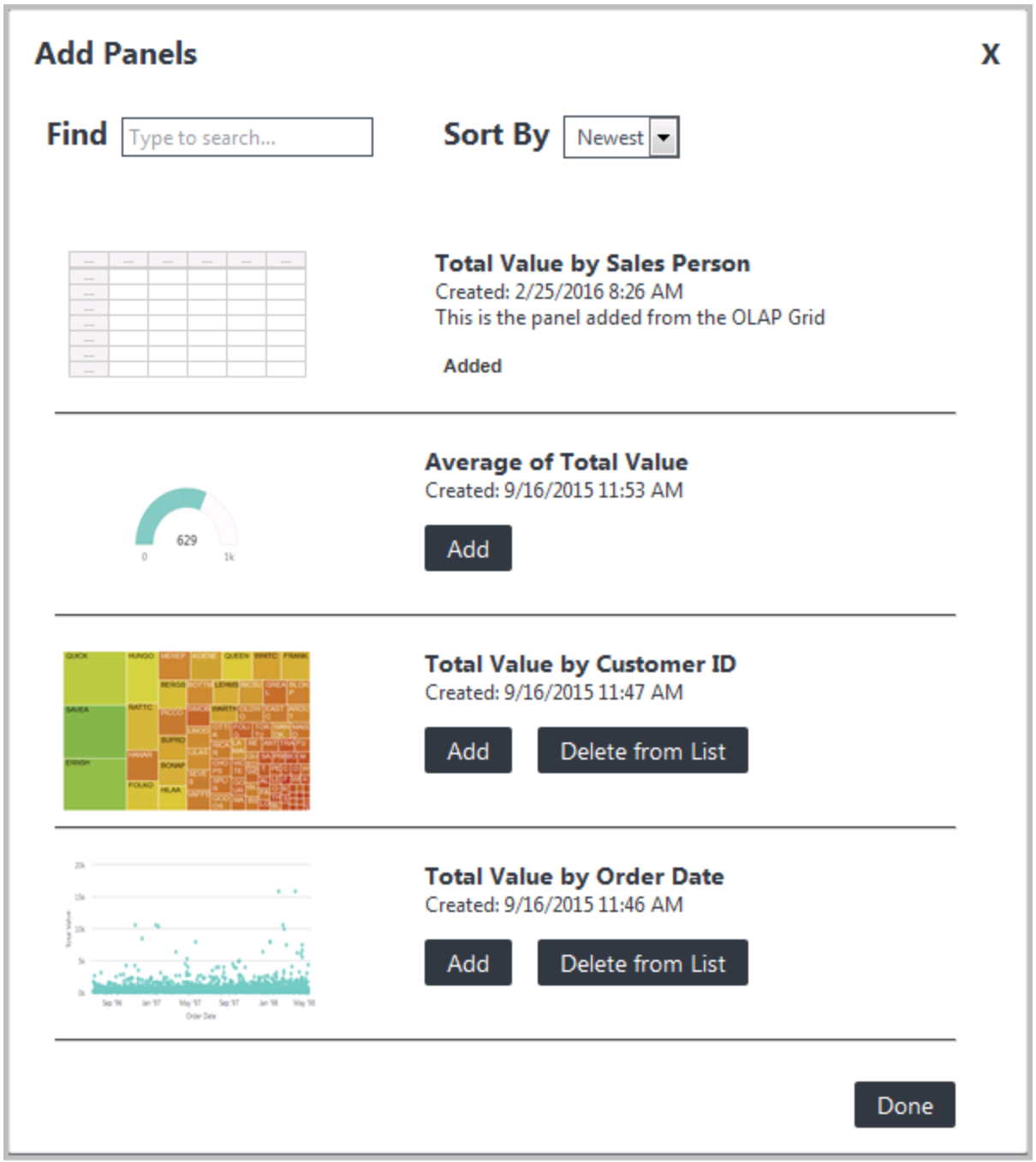
From OLAP Grid...



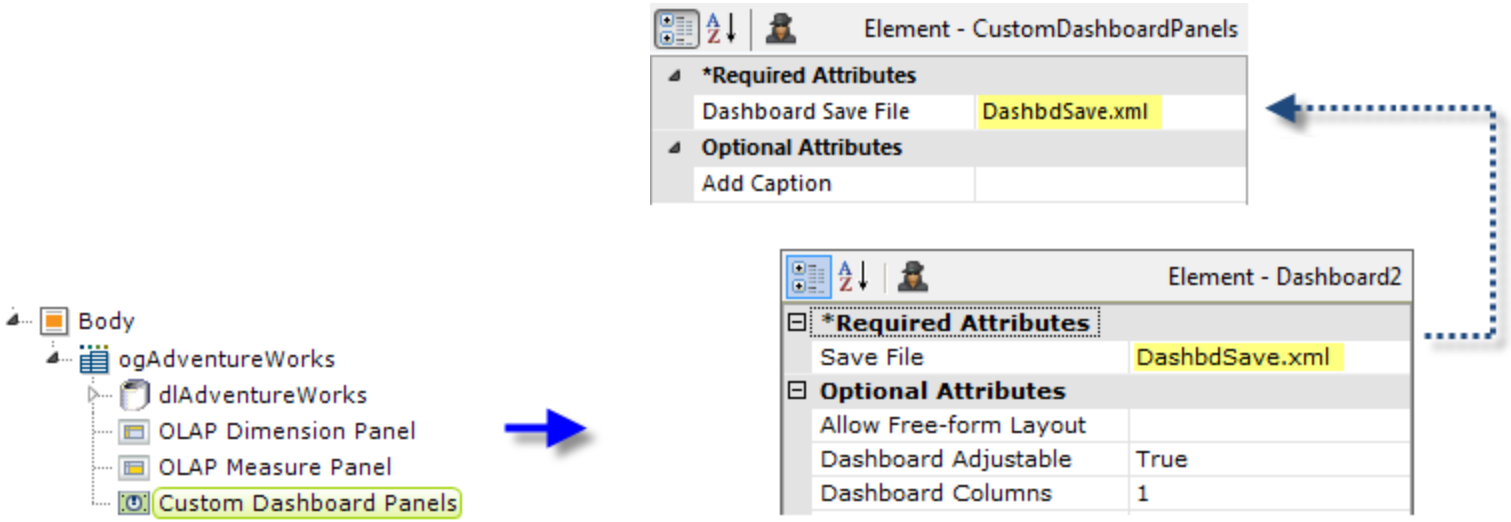
... to Dashboard



When the icon is clicked, the table or chart is added as a new panel in the Dashboard, as shown above. When the resizing control is used to resize a chart, the widths of the Dashboard columns and other panels will adjust dynamically. Behind the scenes, when the icon is clicked, the OLAP Grid writes the underlying XML for its chart into the Dashboard's configuration file (its "Save File"). In the Dashboard, the chart is inserted at the top of Column 1; if Dashboard tabs are being used, the insertion occurs in the currently active tab. Just before panel insertion, the user will be prompted for the **Panel Title** and a description for display on the Configuration Page.


Add Panels
X
Find
Type to search...
Sort By
Newest
Total Value by Sales Person
Created: 2/25/2016 8:26 AM
This is the panel added from the OLAP Grid
Added
Average of Total Value
Created: 9/16/2015 11:53 AM
629
0
1k
Add
Total Value by Customer ID
Created: 9/16/2015 11:47 AM
Add
Delete from List
Total Value by Order Date
Created: 9/16/2015 11:46 AM
Add
Delete from List
Done

The new chart panel thereafter appears in the Dashboard Configuration Page, as shown above, just like any other panel, complete with a thumbnail image. The new panel can be removed from the visible Dashboard panels and from the configuration page entirely, using the usual controls. The user can insert multiple tables and charts into a Dashboard using this technique.



In the report definition, the developer makes this functionality available by adding a **Custom Dashboard Panels** element, shown above, beneath an OLAP Grid element. Its **Dashboard Save File** attribute value should be set to match the target Dashboard's **Save File** attribute value.

The Save File attribute values shown above have been shortened for visual clarity. However, a fully-qualified path and file name, with .xml extension, is required, so a realistic attribute value would be something like:

```
@Function.AppPhysicalPath~\DashFiles\DashbdSave.xml
```

and the account the Logi application runs under (in Windows, ASPNET or NETWORK SERVICE) would be given **Write** permissions to the "DashFiles" folder. For experimentation purposes, you can use the rdDownload folder, for which appropriate permissions already exist (but its contents are periodically deleted, so don't use it for anything but experimentation). Dashboard Save Files are often named using a token, such as @Function.UserName~, in order to create personalized Dashboard configurations and, in that case, the Custom Dashboard Panel attribute would use the same token. The Custom Dashboard Panels element has a new attribute, **Add Caption**, which sets the caption for the button used to save the Data Table or chart. If left blank, the caption is "Add to Dashboard". For more information about Dashboards, see *Logi Info Dashboard* .

Logi OLAP - Customizing OLAP Grid Appearance

OLAP Grid appearance can be changed most easily by applying a theme to the report definition. Most of the screen shots in this topic were taken with the *Signal* theme applied. You can create your own custom theme, based on a standard theme, using the *Theme Editor*tool.

Changing Appearance Using Style Classes

OLAP Grid appearance can be also customized using **style classes** and Logi Studio provides the following standard style sheet for all OLAP Grids:

```
<YourAppFolder>\rdTemplate\rdOlapGrid\rdOgStyle.css
```

Developers can override classes in this style sheet by *copying* them to their application style sheet and modifying them there.

 Do not make changes in the standard style sheet in the rdTemplate folder.

Changing Appearance Using Template Modifiers

The OLAP Grid element uses a "template file" to define certain element properties that are not otherwise available as attributes to the developer for modification. These include language- and culture-specific **Caption** attributes that you may want to change for locale-based reasons (or you may simply want to change the captions to better suit your application). The element's **Template Modifier File** attribute identifies a custom XML file developers can create containing elements that will override the same elements in the template file. For example, the OLAP Grid template file:

```
<yourAppFolder>\rdTemplate\rdOlapGrid\rdOgTemplate.lgx
```

contains several Label elements. One of them has an ID of "lblShowEmpty"; this controls the caption that appears next to the "Show empty members" check box. This text can be modified by changing the **Caption** associated with that Label element. For example, to change the Label text from its default to "View empty members", create your own XML file, identify it in the OLAP Grid element's Template Modifier File attribute, and add this code to it:

<TemplateModifier>
<SetAttribute ID="lblShowEmpty" Caption="View empty members" />
</TemplateModifier> You can set the attributes for any number of elements in this file; examine the rdOgTemplate.lgx file to learn the ID and Caption attributes available. The template modifier file can be in any folder accessible to the web application; if a fully-qualified file path is not provided in the Template Modifier File attribute value, then the application expects it to be in your project's _SupportFiles folder. More detailed information about template modifier files can be found in *Template Modifier Files*.

Logi XOLAP

Logi **XOLAP** (pronounced "zo lap") is a technology that brings the analytical power of data cubes to your relational data and other diverse data sources such as web services, XML, and files.

The following topics discuss basic Logi XOLAP concepts:

- About Data Cubes
- Logi XOLAP Data Cubes
- The Dimension Grid
- Using XOLAP with OLAP

Logi XOLAP has been deprecated; the related elements are still supported, for now, and will work but they're no longer available in Logi Studio.

About Logi XOLAP

Unlike traditional OLAP, all processing of multi-dimensional data within **Logi XOLAP** happens at run-time through the Logi Info engine, without the development and processing overhead of more traditional OLAP software. Now you can "slice and dice" large amounts of data from your SQL queries and other data sources *on demand*. While traditional OLAP data is usually generated only periodically and may become *out of date* until it's refreshed, Logi XOLAP data is always current.

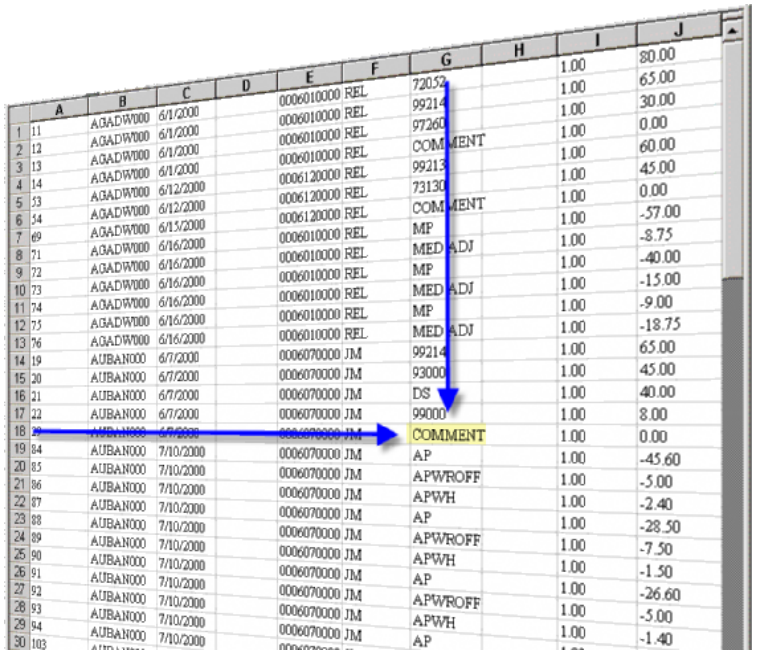
This highly flexible model for bringing easy analysis to users can be very complimentary to traditional OLAP analysis. XML and web-server powered, it's very consistent with the extensible and flexible approach at the heart of all of Logi Analytics tools.

In addition, due to Logi's elemental development approach and our Logi Studio design tool, Logi XOLAP implementations can happen very quickly.

Logi XOLAP - About Data Cubes

A **data cube** is a collection of data that allows analysis in multiple "dimensions".

Consider, for example, a spreadsheet. This is a classic arrangement of data in two dimensions: *rows* and *columns*. The intersection of a row and column provides us with a specific data value:



	A	B	C	D	E	F	G	H	I	J
1	11	AGADW000	6/1/2000		0006010000	REL	72052		1.00	80.00
2	12	AGADW000	6/1/2000		0006010000	REL	99214		1.00	65.00
3	13	AGADW000	6/1/2000		0006010000	REL	97260		1.00	30.00
4	14	AGADW000	6/1/2000		0006010000	REL	COMMENT		1.00	0.00
5	53	AGADW000	6/12/2000		0006120000	REL	99213		1.00	60.00
6	54	AGADW000	6/12/2000		0006120000	REL	73130		1.00	45.00
7	69	AGADW000	6/15/2000		0006010000	REL	COMMENT		1.00	0.00
8	71	AGADW000	6/16/2000		0006010000	REL	MP		1.00	-57.00
9	72	AGADW000	6/16/2000		0006010000	REL	MED ADJ		1.00	-8.75
10	73	AGADW000	6/16/2000		0006010000	REL	MP		1.00	-40.00
11	74	AGADW000	6/16/2000		0006010000	REL	MED ADJ		1.00	-15.00
12	75	AGADW000	6/16/2000		0006010000	REL	MP		1.00	-9.00
13	76	AGADW000	6/16/2000		0006010000	REL	MED ADJ		1.00	-18.75
14	19	AUBAN000	6/7/2000		0006070000	JM	99214		1.00	65.00
15	20	AUBAN000	6/7/2000		0006070000	JM	93000		1.00	45.00
16	21	AUBAN000	6/7/2000		0006070000	JM	DS		1.00	40.00
17	22	AUBAN000	6/7/2000		0006070000	JM	99000		1.00	8.00
18							COMMENT		1.00	0.00
19	84	AUBAN000	7/10/2000		0006070000	JM	AP		1.00	-45.60
20	85	AUBAN000	7/10/2000		0006070000	JM	APWROFF		1.00	-5.00
21	86	AUBAN000	7/10/2000		0006070000	JM	APWH		1.00	-2.40
22	87	AUBAN000	7/10/2000		0006070000	JM	AP		1.00	-28.50
23	88	AUBAN000	7/10/2000		0006070000	JM	APWROFF		1.00	-7.50
24	89	AUBAN000	7/10/2000		0006070000	JM	APWH		1.00	-1.50
25	90	AUBAN000	7/10/2000		0006070000	JM	AP		1.00	-26.60
26	91	AUBAN000	7/10/2000		0006070000	JM	APWROFF		1.00	-5.00
27	92	AUBAN000	7/10/2000		0006070000	JM	APWH		1.00	-1.40
28	93	AUBAN000	7/10/2000		0006070000	JM	AP			
29	94	AUBAN000	7/10/2000							
30	103									

We can also use all of the column values along a single row or down a single column for comparison or summarization purposes. This row-column model also describes a database table.

Now, let's consider what happens when we "stack" several identically-formatted spreadsheets, one behind the other. If we pick a specific cell in the front sheet and "drill through" it, we can examine and compare the values for that one cell on *all* of the sheets. This is a "three-dimensional" comparison or summarization, and an example of how data can be arranged into a "cube". The drill-

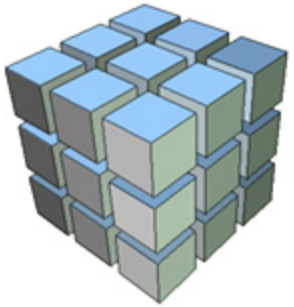
through provides a third dimension. This is particularly useful in comparing or summarizing data over time. The data cube, which is the foundation of *Online Analytical Processing*, or OLAP, is particularly useful in finding trends in very large amounts of data.

OLAP data cubes are usually marvels of data warehouse engineering, and designing and implementing them is a time-consuming, complicated process. Specialized database servers are often employed and queries made against an OLAP cube use a special query syntax (MDX) that's similar, but notably different, from that of standard SQL queries.

Logi Info includes OLAP elements that can issue these queries against cubes created using Microsoft SQL Server Analysis Services and the results can be viewed, aggregated, summarized, and visualized. Logi Info, however, does not *create* OLAP cubes. More information can be found in "Logi OLAP" on page 487.

Logi XOLAP - Logi XOLAP Data Cubes

Working from the premise that there can be useful multi-dimensional analysis of data that's not necessarily in an OLAP cube, Logi Info includes the ability to create similar data structures *on the fly* and report on them.



At run time, a Logi Info report can query a wide variety of data sources and build a temporary, **XML-based virtual data cube** from the results.

Specialized Logi Info elements allow analysis of, and reporting on, these XML cubes. *Logi XML Online Analytical Processing*, or **Logi XOLAP**, delivers much of the analytical power of data cubes without the overhead associated with building large OLAP systems.

Naturally, there are some limits. While Logi XOLAP is similar to OLAP in its analytical methods, it doesn't necessarily support the same *scale*.

OLAP cubes usually include vast amounts of often historical data and are designed with a range of queries in mind, so that specific and ad hoc queries can be handled. They are relatively permanent structures and are often only dynamic in that new data is constantly added to them. Essentially, an OLAP cube is designed and built long *before* the first query against it is issued.

Logi XOLAP, on the other hand, builds a virtual data cube *on demand* in response to specific query from an individual user. Therefore, queries that involve *millions* of result rows may result in unacceptably slow performance. As with all attempts to quantify database limits, a number of factors such as the number of columns, data types, and the number of concurrent queries make it difficult to provide specific row limits.

Logi XOLAP is generally optimized for 10's to 100's of thousands of records, and with the benefit of the Hybrid Caching of the Logi Server Engine, it can also scale based on hardware such as RAM and CPU/Processing power. For more information, see *Hybrid Caching*.

Logi XOLAP performance is monitored in a special section of the Logi Info **Debugger Trace Page**, so you can see how long it takes to process your data.

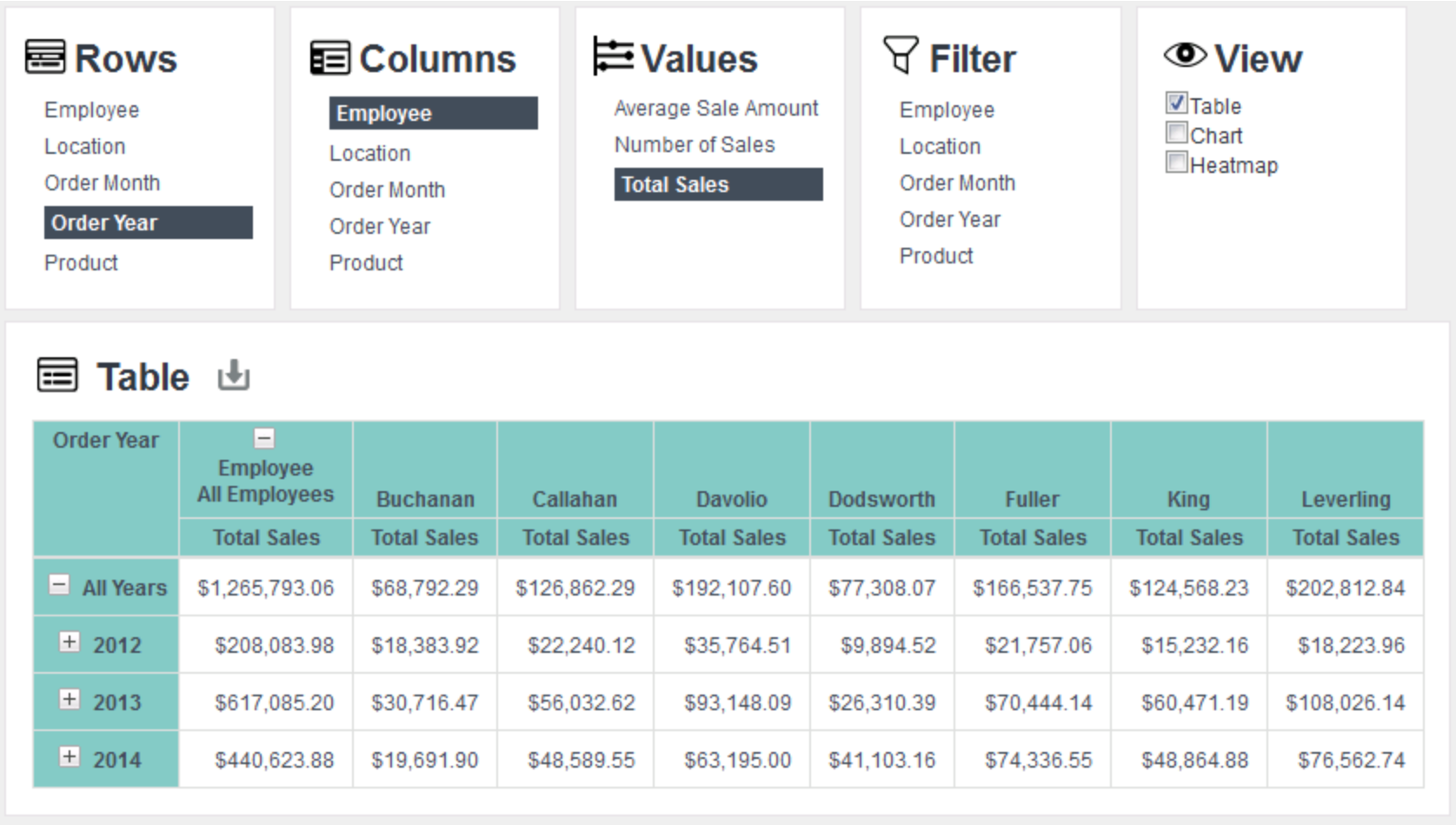
Start OLAP XML Cube			
XOLAP	Starting XOLAP object		.197
	Cache ID	XOLAPCubeXML||Total Sales,|[Order Year]::[Order Year].[All Years],[Order Year].[All Years].[1998],|[Employee]::[Employee].[All Employees],|	.197
	Added drilldown to Left axis	[Order Year].[All Years]	.198
	Added drilldown to Left axis	[Order Year].[All Years].[1998]	.199
	Added drilldown to Top axis	[Employee].[All Employees]	.199
	Added measure	Total Sales	.200
	Writing attributes of top output row		.201
	Parsed dimensions, building output from datalayer		.201
	XOLAP data parsed		.263
	Data parsed, building output XML		.264
	Completed XOLAP object		.265
	Data loop complete	10129 records in 0.11 sec. 5,556,373 per minute	.268
	XML Cube Complete		.269

For example, the Debugger entry above shows that it took just over 1/10th ofa second to query the data and create a Logi XOLAP cube using a 10,000+ record data set on a desktop-class PC. This can be extrapolated, given the exact same data, cube dimensions and measures, and query load, to 5.5 million+ rows a minute. We can't guarantee a specific performance level but this example is representative.

The ability to create virtual cubes of XML data, even from data sources not normally associated with OLAP, and report on them is a fantastic capability that allows new opportunities for data analysis.

Logi XOLAP - The Dimension Grid

The **Dimension Grid** is a super-element that provides a complete user interface for data analysis using Logi XOLAP cubes:



Order Year	Employee All Employees	Buchanan	Callahan	Davolio	Dodsworth	Fuller	King	Leverling
	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales
All Years	$1,265,793.06	$68,792.29	$126,862.29	$192,107.60	$77,308.07	$166,537.75	$124,568.23	$202,812.84
2012	$208,083.98	$18,383.92	$22,240.12	$35,764.51	$9,894.52	$21,757.06	$15,232.16	$18,223.96
2013	$617,085.20	$30,716.47	$56,032.62	$93,148.09	$26,310.39	$70,444.14	$60,471.19	$108,026.14
2014	$440,623.88	$19,691.90	$48,589.55	$63,195.00	$41,103.16	$74,336.55	$48,864.88	$76,562.74

As shown above, the Dimension Grid consists of panels across the top and the Data Table. Charts and a Heatmap can be displayed beneath the table as well. Users can dynamically change the contents of the Data Table or charts at runtime using the controls in

the panels (subject to the way the report developer has configured the grid). Users, through their browsers, can slice-and-dice and drill into the XOLAP data at runtime to perform the analyses they need.

The *appearance* of the Dimension Grid can be controlled using Logi Themes and custom CSS. For example, the grid shown above uses the Signal Theme. Its *functionality* can also be customized on a per-user basis and the changes users make at runtime can be saved between sessions and restored for their next session.

The Dimension Grid and the XOLAP-family of elements that support it are described in detail in "Dimension Grid" on page 535.

Logi XOLAP - Using XOLAP with OLAP

The XOLAP Cube element, and its child elements, which support the Dimension Grid, can be also used with the **OLAP Grid** and **OLAP Table** elements. This provides alternate datasources for the OLAP elements and the ability to use the more flexible OLAP Grid with that data.

What are the differences between the Dimension Grid and the OLAP Grid? The Dimension Grid is designed to be less complicated to use and doesn't support some features, such as multi-dimension sorting and multiple dimensions on both axes, that are found in the OLAP Grid. So the OLAP Grid allows more complex analysis of the XOLAP data.

The Dimension Grid, however, only displays data contained in Logi XOLAP cubes; it's not able to connect to a OLAP cube, nor do we offer a mechanism for building a XOLAP cube by querying an OLAP cube. Therefore, you must use the OLAP Grid element, not a Dimension Grid, to display OLAP data.

Dimension Grid

The **Dimension Grid** is a "super-element", similar to the **Analysis Grid** and the **OLAP Grid** elements, used with Logi **XOLAP** data cubes.

The following topics describe the Dimension Grid and related elements:

- Dimension Grid Attributes
- Dimension Grid Settings Panels
- Configuring the Data Cube
- Defining an Initial View
- Logi XOLAP on the Debugger Page
- Adding Visualizations to Dashboards
- Refresh, Reset, and Save Grid Settings
- Customizing Dimension Grid Appearance

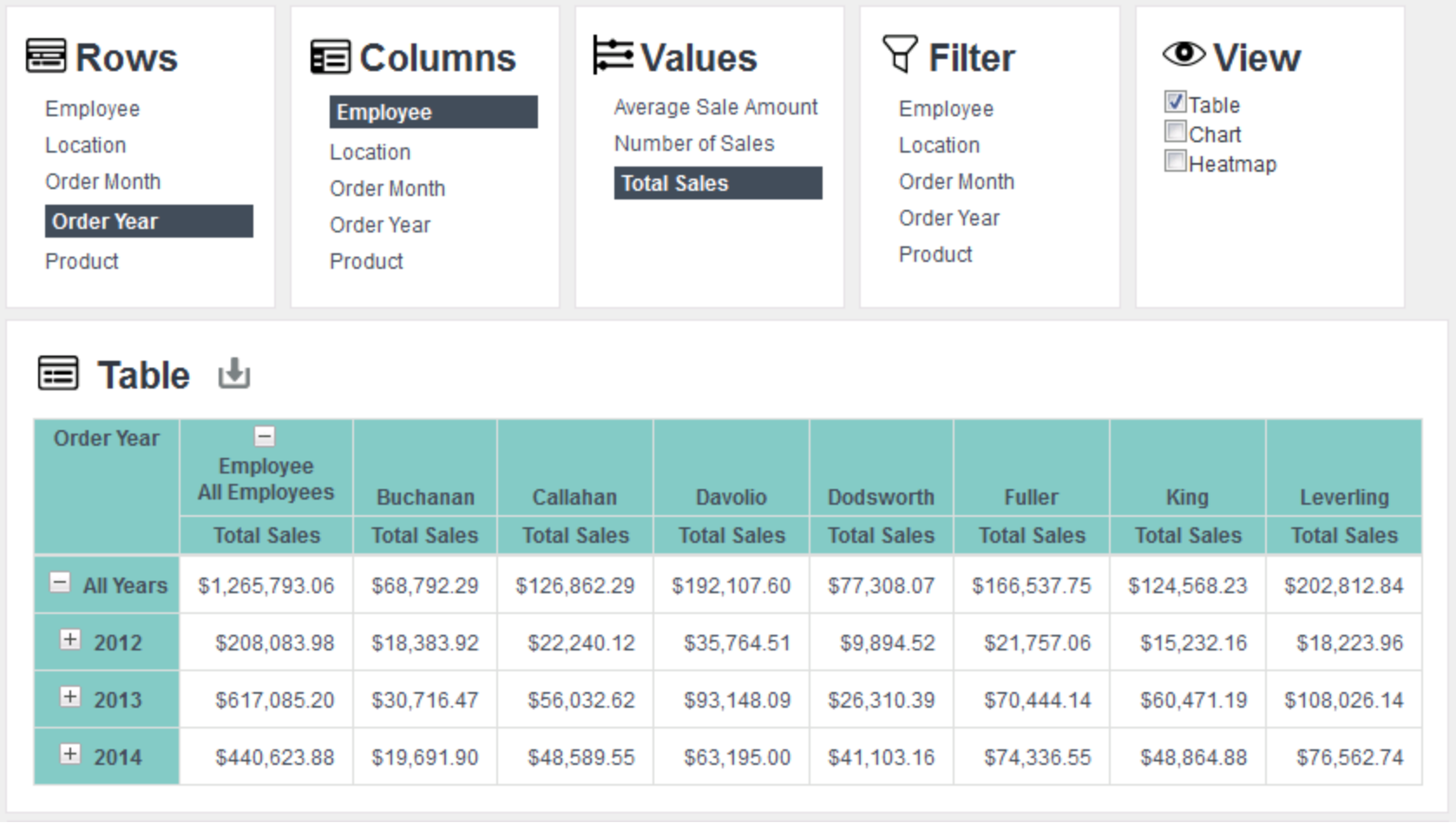
 *Logi XOLAP has been deprecated; the related elements are still supported, for now, and will work but they're no longer available in Logi Studio*

About the Dimension Grid

If you have not done so already, see "Logi XOLAP" on page 525 for important basic information before proceeding here.

The two major parts of the Logi XOLAP technology are the **Dimension Grid** element, which is the user interface into the data, and its child element, the **XOLAP Cube**, which retrieves the data, constructs the data cube, and connects it to the Dimension Grid. The XOLAP Cube uses a number of child elements, including one or more datalayers, to retrieve the desired data.

At runtime, Dimension Grid users can manipulate the data they see and the way in which it's presented using the Dimension Grid's built-in user interface controls. The report developer has complete control over which controls are available to a user and can present them selectively to different users.

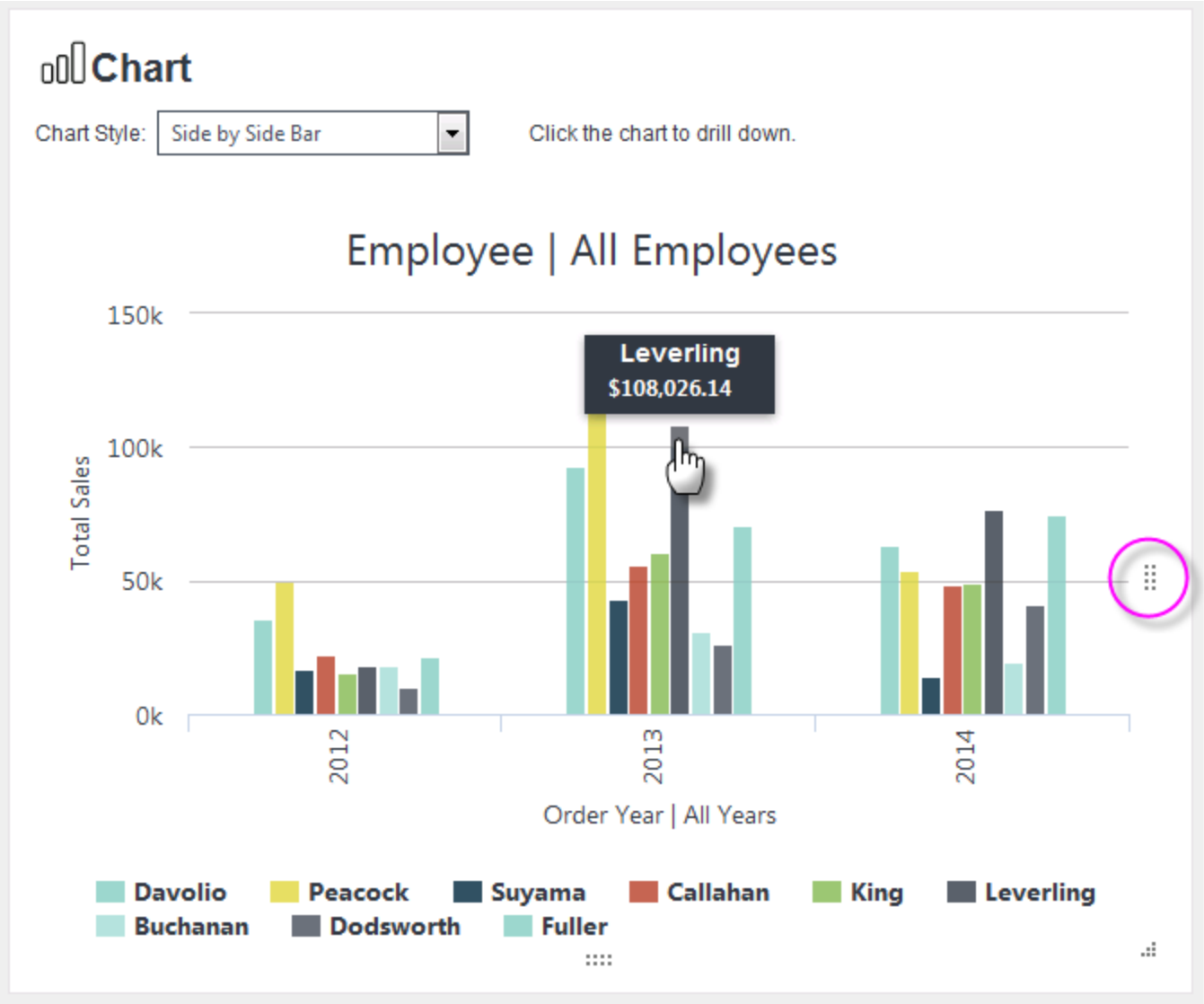


Order Year	Employee All Employees	Buchanan	Callahan	Davolio	Dodsworth	Fuller	King	Leverling
	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales
All Years	$1,265,793.06	$68,792.29	$126,862.29	$192,107.60	$77,308.07	$166,537.75	$124,568.23	$202,812.84
2012	$208,083.98	$18,383.92	$22,240.12	$35,764.51	$9,894.52	$21,757.06	$15,232.16	$18,223.96
2013	$617,085.20	$30,716.47	$56,032.62	$93,148.09	$26,310.39	$70,444.14	$60,471.19	$108,026.14
2014	$440,623.88	$19,691.90	$48,589.55	$63,195.00	$41,103.16	$74,336.55	$48,864.88	$76,562.74

The example above shows a Dimension Grid configured to show two of its option panels and set to include a Bar chart. The Data Table behaves like a Crosstab Table, and can pivot data.

The Dimension Grid settings configured by a user can be *saved* at the end of their session and *restored* during a later session.

Dimension Grid charts include a number of versatile features:

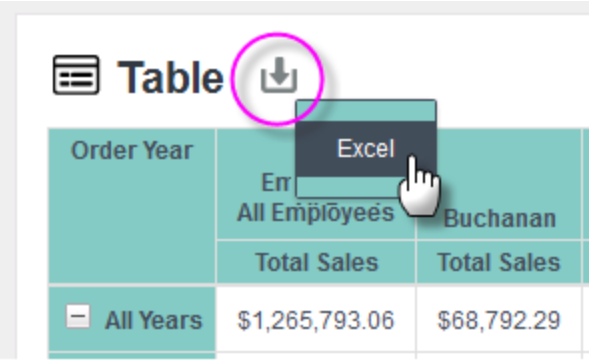


An automatic **tooltip** is displayed, as shown above, when the mouse cursor is hovered over a chart segment. The chart can be resized dynamically, by dragging the *resizer handles* (circled above), which appear when the mouse cursor is hovered over the chart.

In addition, you can *drill-down* into the data by clicking on chart segments. When you do, the chart will redraw itself at the new level of detail. Click the appropriate chart icon in the Data Table to return to the previous data level.

Chart Canvas charts are used by default by the Dimension Grid, in all *new applications*. Older Logi applications that are *upgraded* to v12 will use the Classic static charts for their Dimension Grids by default. To force upgraded apps to use Chart Canvas charts, add the constant `rdFavorChartCanvas = True` to your _Settings definition.

The **Dimension Grid Group Drillthrough** child element enables drill-through on cell values and adds a drill-through link in each cell. When the drill-through icon is clicked, a new report page containing a basic Data Table is displayed. The table lists the drill-through rows with all the columns used to build the cube. The element's **Caption** attribute sets the title of the drill-through report page.



Data from the Dimension Grid table can be exported to Excel using the Export icon, circled above.

Logi Studio includes a Dimension Grid Wizard that can add and configure all of the elements necessary to work with Logi XOLAP for you. For information about this wizard, see *Dimension Grid Wizard*.

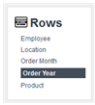
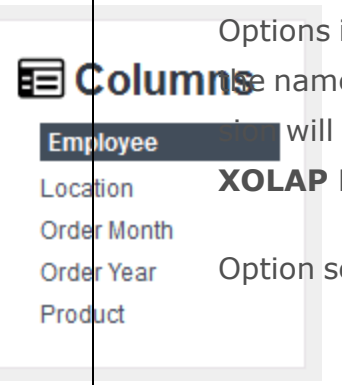
Dimension Grid Attributes

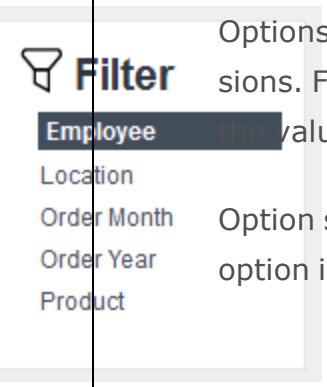
The **Dimension Grid** element is the parent element of all other XOLAP family elements. Its attributes are:

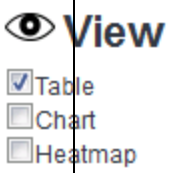
Attribute	Description
ID	(Required) A unique element ID.
Caption	Specifies a title that will appear in the bar above the control panels.
Level Indent	Specifies the amount of indentation spaces used when showing levels of a hierarchy on the left side of the Dimension Grid. Default value: *2*
Saved Dimension Grid Folder	Specifies the name of the folder in which Dimension Grid settings made by a user at runtime may be saved by using the **Procedure.Save Dimension Grid**element in a process task. The settings can later be reloaded by passing the saved setting filename in the rdDgLoadSaved parameter. The @Function.AppPhysicalPath~ token can be used here as part of the path to the folder.
Security Right ID	Specifies one or more IDs of Rights defined in the application's _Settings/Security section when using Logi Security and controls who can see and use the Dimension Grid element.
Template Modifier File	Specifies the name of a custom template file that can be used to change language- and culture-specific Caption attributes in the grid.
Width Width Scale	Specifies the width, and type of width units, for the Dimension Grid.

Dimension Grid Settings Panels

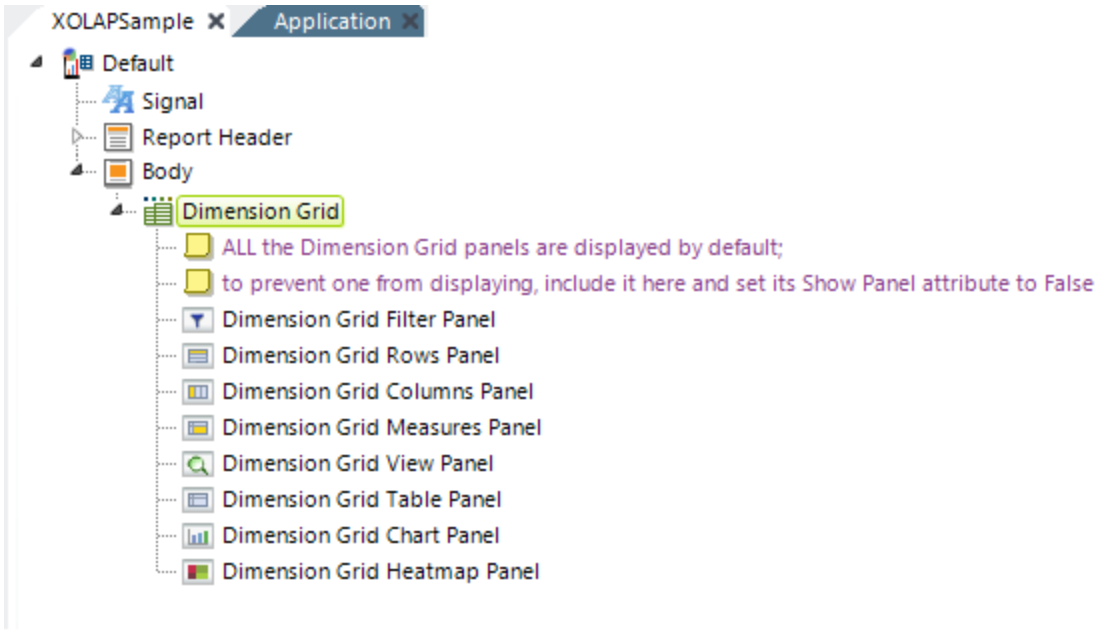
The **Dimension Grid Panel** elements determine what data and which options are available to the user at runtime. Let's examine the panels:

Panel	Description
	Options in this panel allow the user to select the data that will populate each **row** in the Data Table. The options are the names of *Dimensions* in the cube, which represent specific columns in the datalayer. The data for the selected dimension will appear in the left-most column of the Data Table. They're added to the report definition using **XOLAP Dimension** elements. Option selection is mutually-exclusive: clicking one cancels a previously selected option.
	Options in this panel allow the user to select the data that will populate each **column** in the Data Table. The options are the names of *Dimensions* in the cube, which represent specific columns in the datalayer. The data for the selected dimension will appear in the top-most row of the Data Table, as column headers. They're added to the report definition using **XOLAP Dimension** elements. Option selection is mutually-exclusive: clicking one cancels a previously selected option.

Panel	Description
	Options in this panel allow the user to select the data values that will appear in the **cells** that are the *intersections* of rows and columns. The links are the names of *Measures*, which are aggregations, calculations, and summarizations of the data in the datalayer. They're added to the report definition using **XOLAP Measure** elements. Option selection is *not* mutually-exclusive: multiple options can be selected at once and clicking a selected option "unselects" it.
	Options in this panel allow the user to set one or more filters that exclude some or all of the data values for all dimensions. Filter options are automatically created for each dimension. When an option is selected, a popup panel displays the values and check boxes; un-checking a box removes that data from the grid. Option selection is *not* mutually-exclusive: multiple options can be used at once. When a filter has been imposed, the option is highlighted, as shown at left, as an indication that filtering is being used.

Panel	Description
	check boxes in the panel determine which data visualizations (table, chart, and heatmap) will be displayed.

Table, charts, and heatmaps are themselves also Dimension Grid Panel elements and, along with all the rest, can be selectively shown or hidden by the report developer.



The report definition for the Dimension Grid shown earlier, with all its panel elements, looks like the example above. Regardless of the arrangement of the elements, the panels will always be displayed in the browser in the order shown in the first image in this topic. So, for example, you can't make the Rows and Columns panels appear *below* the Data Table.

Dimension Grid Panel elements are *not* required. By default, if you leave them out, all panels will be displayed. To hide a panel, include its grid panel element and set its **Show Panel** attribute to *False*. This value can also be set using @Request and @Session tokens.

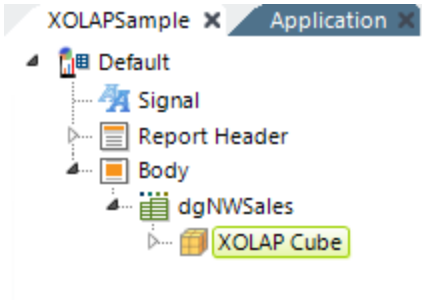
The table, chart, and heatmap panels include attribute values for initially setting them as visible or hidden, and the chart panel includes an additional **Color** attribute that lets you select a custom color set for the charts. To set custom colors, enter a color by

name, decimal RGB value, or hex RGB value. Prefix hex values with the pound sign, e.g. #112233. Colors for charts with multiple series can be specified as a comma-separated list. The use of a built-in Theme will provide colors automatically.

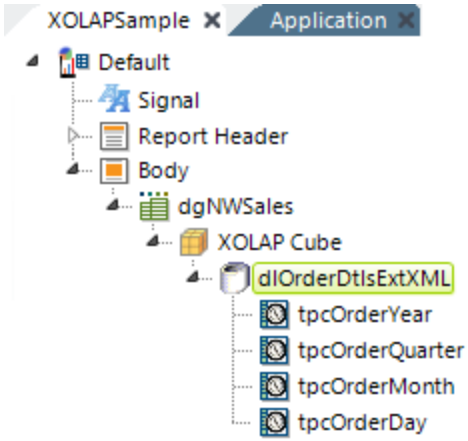
One interesting possible use-case is to show only the chart and not the Data Table, heatmap, or any other panels. Drill-down into the data would still be available through the chart and this might be a useful configuration, for example, in a Dashboard.

Configuring the Data Cube

In order to create the data cube, we need to use a **datalayer** to retrieve the data and several XOLAP-family elements to define the aspects of the cube related to that data.



As shown above, the **XOLAP Cube** element is added to a definition as a child of the Dimension Grid.

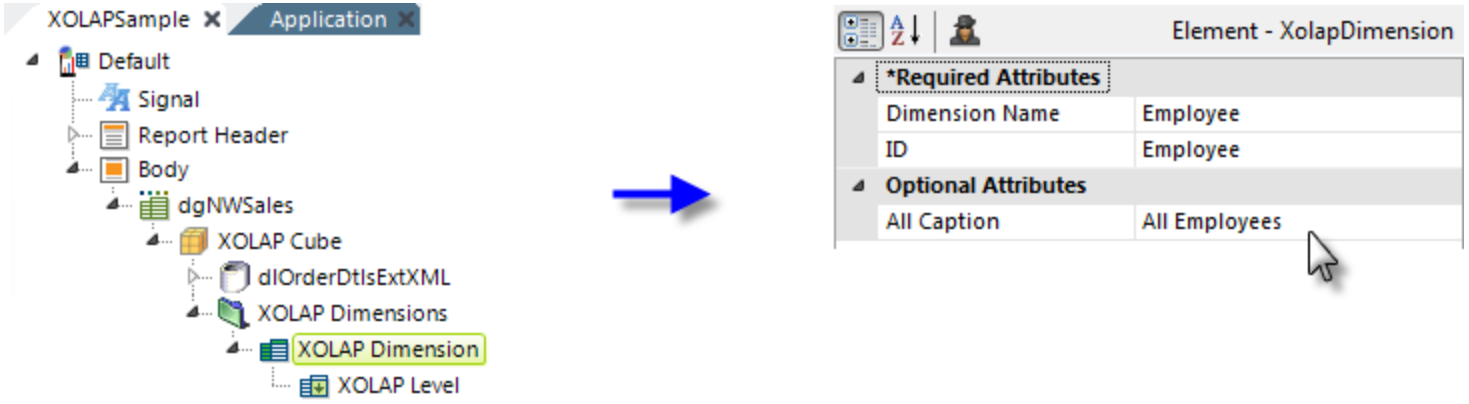


Next, a datalayer appropriate to our data source is added to retrieve the data. As shown above, any of the usual datalayer child elements (in this case, Time Period Columns) can be added as well, to shape the data as desired.

The actual **Source** attribute value for the DataLayer.SQL element in this example is:

```
SELECT [Order Details Extended].UnitPrice, [Order Details Extended].Quantity,
[Order Details Extended].ExtendedPrice, Categories.CategoryName,
Products.ProductName, Orders.OrderDate, Customers.CompanyName,
Customers.Country, Customers.Region, Customers.City, Employees.LastName, Employees.FirstName
FROM [Order Details Extended] INNER JOIN
Orders ON [Order Details Extended].OrderID = Orders.OrderID INNER JOIN
Products ON [Order Details Extended].ProductID = Products.ProductID INNER JOIN
Categories ON Categories.CategoryID = Products.CategoryID INNER JOIN
Customers ON Customers.CustomerID = Orders.CustomerID INNER JOIN
Employees ON Employees.EmployeeID = Orders.EmployeeID
```

Given that we're analyzing the data from a number of tables, the large number of joins is typical of Dimension Grid queries.



Now we add, as shown above, a **XOLAP Dimensions** element and, beneath it, a **XOLAP Dimension** element. The former is just a container, without any attributes, and can have many XOLAP Dimension children. The latter provides a way to "slice" data in the Dimension Grid.

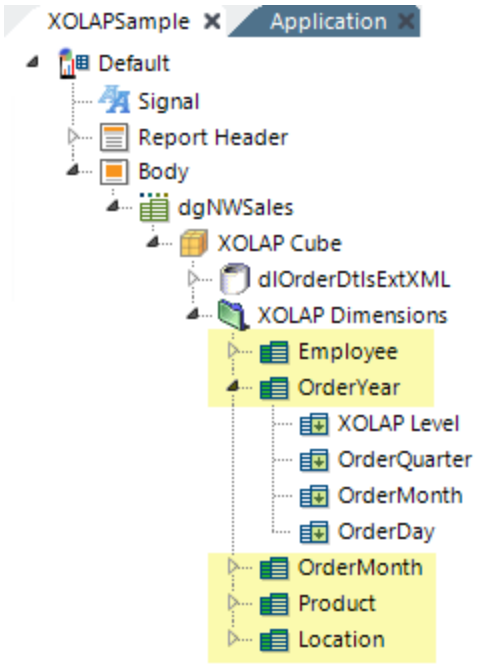
There can be multiple dimension child elements, and each contains one or more **XOLAP Level** elements, enabling the creation of a general-to-specific hierarchy of data. For example, a single-level dimension could be based on a data field indicating the marital status of a customer, while a multi-level dimension could reflect the location of a customer, with levels for country, state, city, and street address.

The XOLAP Dimension element's **Dimension Name** attribute value appears as an option in the Dimension Grid's Rows, Columns, and Filters panels. Dimensions automatically have an "All" category at the highest level. Defining cube levels enables drilling down into the dimension.

The **XOLAP Level** element's attributes identify the data column which contains the value at a dimension level.

Sorting Dimensions by Dates

Dates in dimension values are sorted as strings, which can be confusing. To make a date-type dimension appear in sorted order across the Dimension Grid, ensure that its data is returned into the datalayer as a DateTime value in ISO format (yyyy-mm-dd). To sort the dimension by month and display the month number, set its XOLAP Level element's **Format** attribute to *MM*. To sort it by year and display the year number instead, set this attribute to *yyyy*.

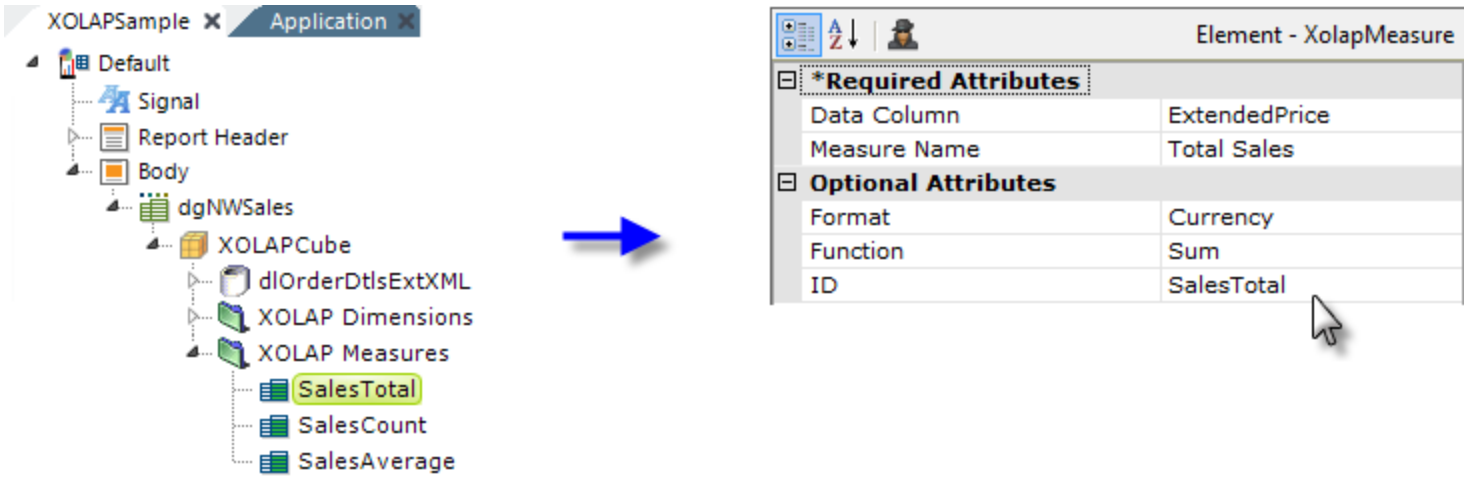


The example above shows the definition after additional dimensions and levels have been added. Look at the image of the Rows panel above and you'll see how the XOLAP Dimension names equate to the options in the panels.

Member Properties and Calculated Measures

The **XOLAP Member Property** element allows use of a "member property", an additional data field that can be displayed alongside a dimension level on the left axis. It does not have to be related to the dimension of the level to which it belongs.

For example, a member property for "City Population" data could be defined on the City level of the Customers dimension, between levels for State and Address. The City Population will then be available for display whenever the City level of the Customers dimension is visible on the left axis.



In the example shown above, the datalayer and dimensions are collapsed for reading clarity. A **XOLAP Measures** element is added, with child **XOLAP Measure** elements beneath it. The XOLAP Measure element specifies a numeric data-type field that can be displayed in the grid. Data values are aggregated based on the **Function** attribute, and the aggregate value can be formatted using the **Format** attribute.

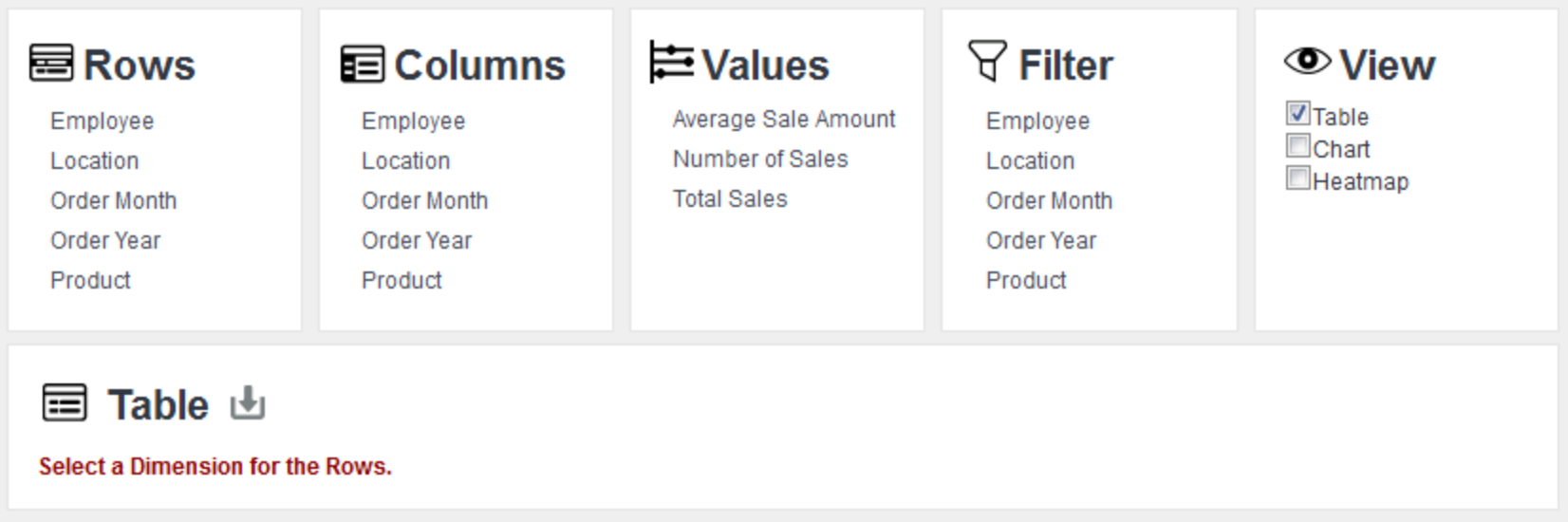
The **XOLAP Calculated Measure** element (not shown) allows you to create measures that use formulae that reference other measures. A new token, **@Measure**, allows you to reference the values of other measures in those formulae.

The **Measure Name** attribute for these elements is picked up and shown as an option in the Dimension Grid's **Values** settings panel.

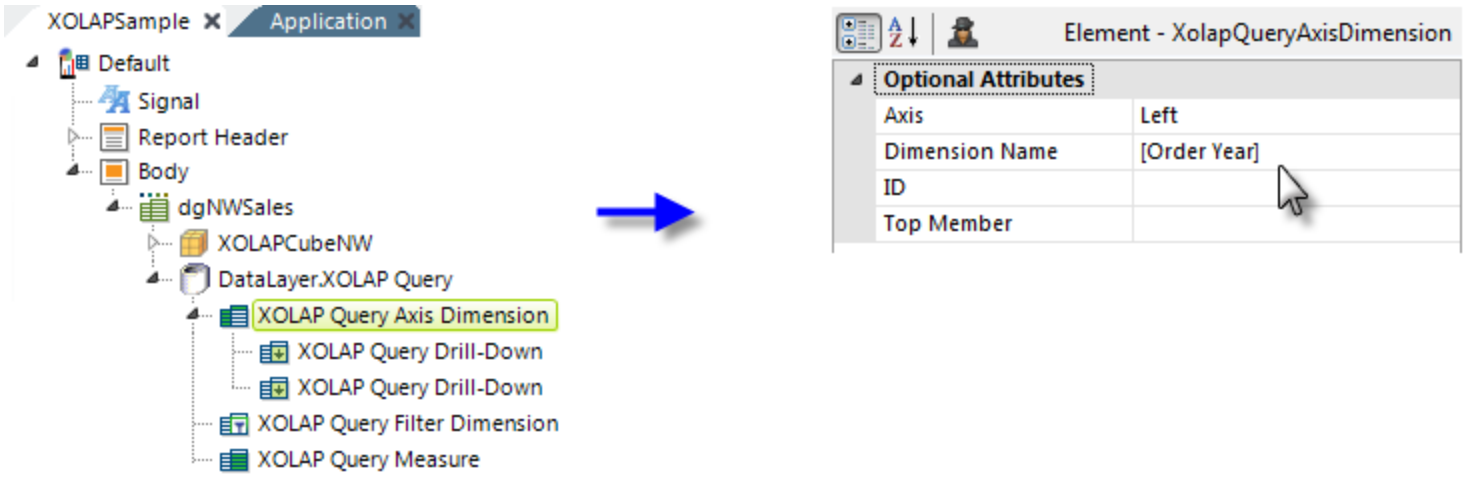
And, with that, we have everything we need for a functional Dimension Grid.

Defining an Initial View

If you were to run the example XOLAP definition shown in "Configuring the Data Cube" on page 546 you would notice that, while we define all the dimensions and measures, when the Dimension Grid is first displayed, no data is actually shown.



The example above shows what a user typically first sees: no data and an instruction to selection a Dimension. What if you want the Dimension Grid to show data right away? Well, we have an element for that.



A special datalayer, **DataLayer.XOLAP Query**, shown above, is used for this purpose. This element, and its child elements, allows you to retrieve the data, identify default Dimensions and Measures, and present the user with visible data the first time the report runs.

Logi XOLAP on the Debugger Page

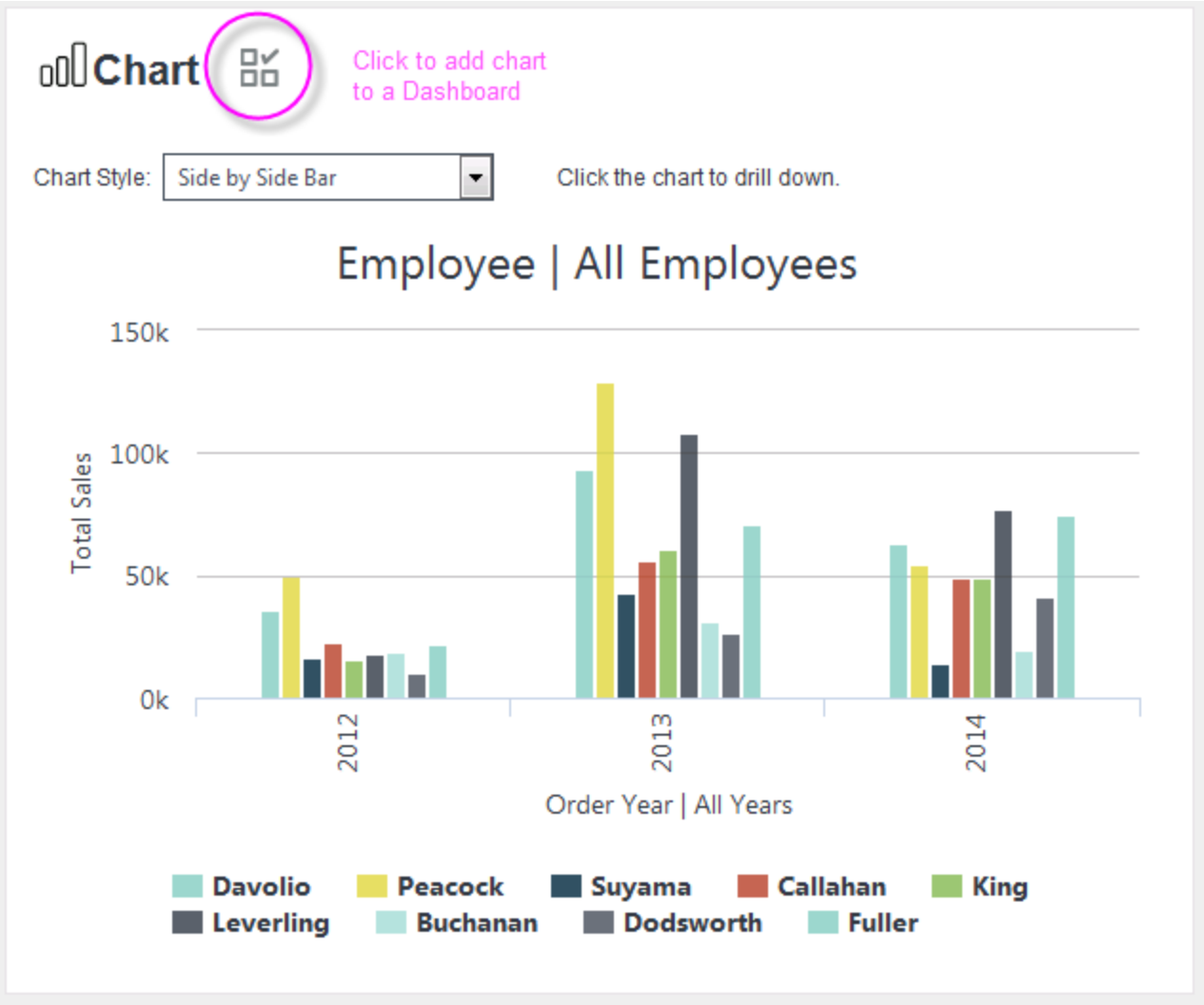
The Debugger Trace Page that appears when debugging is turned on contains very useful information when Logi XOLAP is used. For more information, see *Debug Reports*.

DimensionGrid	Generated	View Definition	.016
	Cache ID	MyXolapCube|||[Location]:[Location].[All Location],|[Employee]:[Employee].[All Employee],|	.053
Get DataLayer - XMLFile	ID	xgCached	.053
	XMLFile	C:\Inetpub\wwwroot\V10Test\rdDataCache\161_22d15545-23c9-467b-bd50-0dedb8f71b06.xml	.053
	XMLFile Physical Filename	C:\Inetpub\wwwroot\V10Test\rdDataCache\161_22d15545-23c9-467b-bd50-0dedb8f71b06.xml	.054
	Setting Up XML Data	Applying transform.	.054
	Setting Up XML Data	Starting copy.	.054
	Setting Up XML Data	2,155 rows of data prepared.	.120
	Formatted XML Data	The XML file is in the correct format.	.120
	DataLayer for Element MyXolapCube	View Memory Stream Data (626,845 bytes)	.120
	Moving Processed Data	Starting move.	.139
	Moving Processed Data	2,155 rows of data moved.	.161
Get DataLayer - EmptyDataLayer	ID	DataLayer	.162
	Data Engine	Data Processing completed.	.162
XOLAP	Starting XOLAP object		.162
	Cache ID	MyXolapCube|||[Location]:[Location].[All Location],|[Employee]:[Employee].[All Employee],|	.163
	Added drilldown to Left axis	[Location].[All Location]	.163
	Added drilldown to Top axis	[Employee].[All Employee]	.163
	Writing attributes of top output row		.164
	Parsed dimensions, building output from datalayer		.164
	XOLAP data parsed		.252
	Data parsed, building output XML		.257
	Completed XOLAP object		.262
	Data loop complete	2155 records in 0.13 sec. 1,034,400 per minute	.287
	CellSet	View Data	.302
XolapTable	Generated	View Definition	.352

As shown in the example above, the Debugger Trace Page includes the details of Logi XOLAP processing. It also includes the total elapsed processing time in seconds, and an extension of that time in minutes, shown highlighted in yellow. This can be useful in understanding and possibly tweaking performance, and is especially helpful when using DataLayer.XOLAP Query.

Adding Visualizations to Dashboards

An exciting level of interaction between the **Dimension Grid** and the **Dashboard** element is available. Users working with an Dimension Grid in one report, at runtime, can generate a table or chart and then, with one mouse click, add it as a new panel in an existing Dashboard in another report.

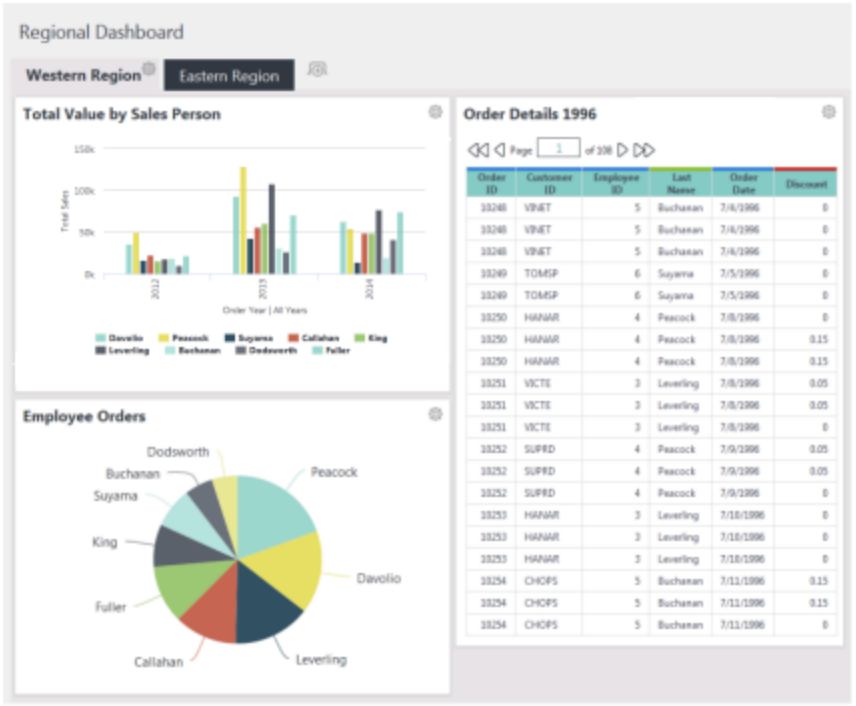


When properly configured, Dimension Grid table and charts will display an **Add To Dashboard** icon, as shown above.

From Dimenson Grid...



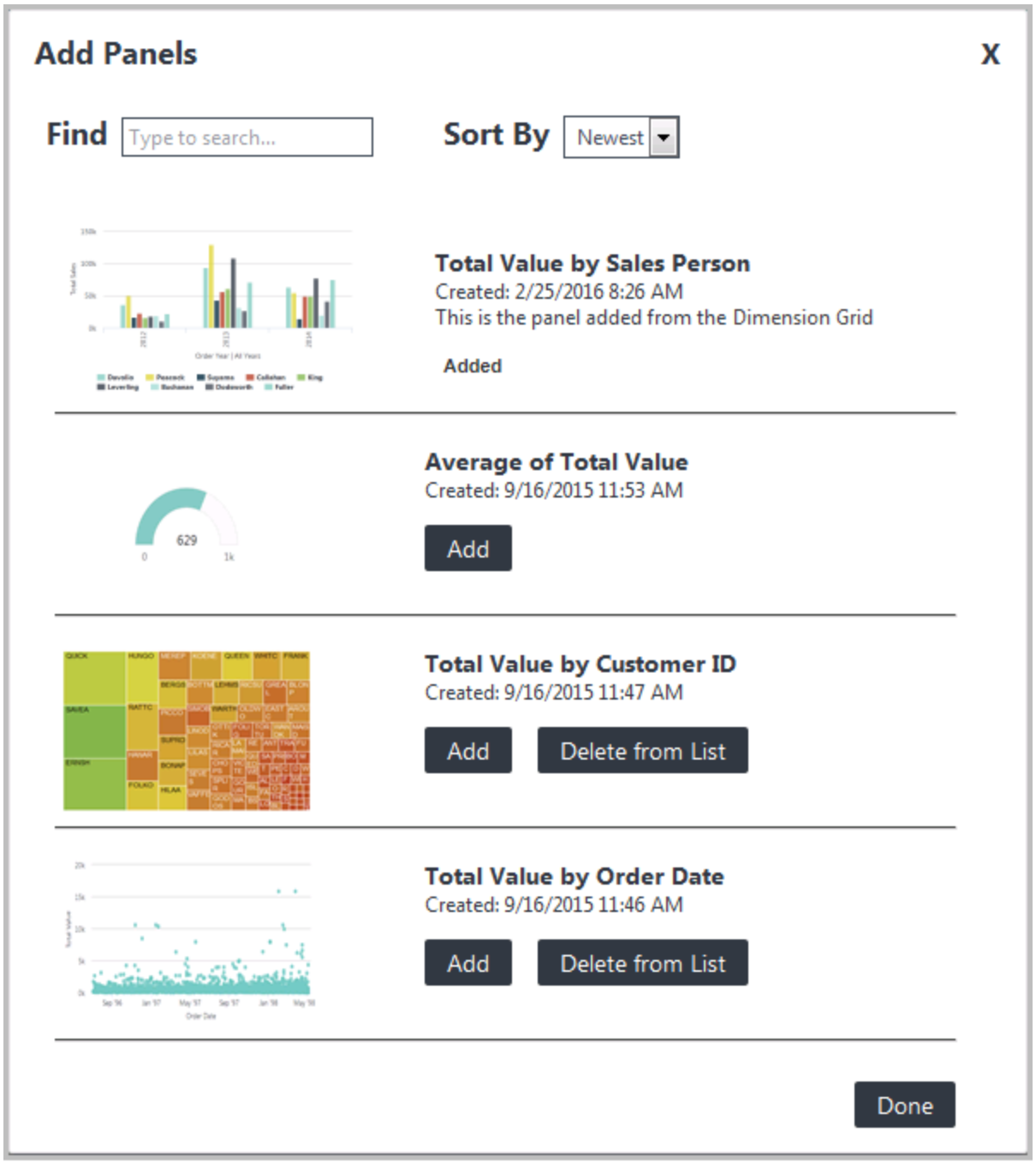
...to Dashboard



When the icon is clicked, the table or chart is added as a *new panel* in the Dashboard, as shown above. When the resizing control is used to resize a chart, the widths of the Dashboard columns and other panels will adjust dynamically.

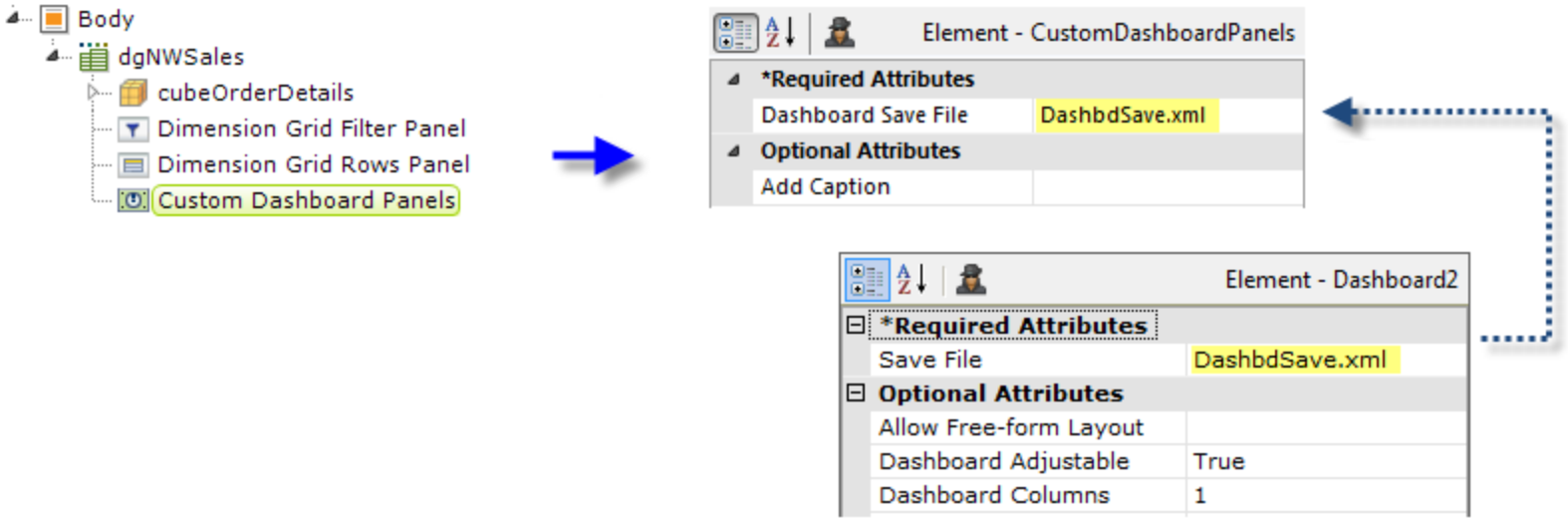
Behind the scenes, when the icon is clicked, the Dimension Grid writes the underlying XML for its chart into the Dashboard's configuration file (its "Save File"). In the Dashboard, the chart is inserted at the top of Column 1; if Dashboard tabs are being used, the insertion occurs in the currently active tab.

Just before panel insertion, the user will be prompted for the **Panel Title** and a description for display on the Configuration Page.


Add Panels
X
Find
Type to search...
Sort By
Newest
Total Value by Sales Person
Created: 2/25/2016 8:26 AM
This is the panel added from the Dimension Grid
Added
Average of Total Value
Created: 9/16/2015 11:53 AM
629
0
1k
Add
Total Value by Customer ID
Created: 9/16/2015 11:47 AM
Add
Delete from List
Total Value by Order Date
Created: 9/16/2015 11:46 AM
Add
Delete from List
Done

The new chart panel thereafter appears in the Dashboard Configuration Page, as shown above, just like any other panel, complete with a thumbnail image. The new panel can be removed from the visible Dashboard panels and from the configuration page entirely, using the usual controls.

The user can insert multiple tables and charts into a Dashboard using this technique.



To enable this functionality, the developer adds a **Custom Dashboard Panels** element, shown above, beneath an Dimension Grid element. Its **Dashboard Save File** attribute value should be set to match the target Dashboard's **Save File** attribute value.

The Save File attribute values shown above have been shortened for visual clarity. However, a fully-qualified path and file name, with .xml extension, is required, so a realistic attribute value would be something like:

```
@Function.AppPhysicalPath~\DashFiles\DashbdSave.xml
```

Dashboard Save Files are often named using a token, such as @Function.UserName~, in order to create personalized Dashboard configurations and, in that case, the Custom Dashboard Panel attribute would use the same token.

The Custom Dashboard Panels element's **Add Caption** attribute specifies the caption for the *button* used to save the Data Table or chart. If left blank, the default caption is "Add to Dashboard".

For more information about Dashboards, see *Logi Info Dashboard* .

Refresh, Reset, and Save Grid Settings

There are several possible actions that require links outside of the grid which you might want to make available to Dimension Grid users:

For This Action	Do This
Refresh	To refresh the data in the grid, without affecting any other settings, call the report definition recursively and include this Link Param: `rdDgRefreshData = True`The parameter name is case-sensitive.
Reset	To clear all user settings and restore the grid to its default state, call the report definition recursively and include this Link Param: `rdDgReset = True`The parameter name is case-sensitive.
Save Settings	To save all user settings for future sessions, use an Action.Process element to call a process task. In the task use a **Procedure.Save Dimension Grid** element to save the settings. The file name itself can incorporate a login name, a GUID, or another method of making it unique to the user, and can be preserved in a number of ways, including as a Cookie on the user's machine.
Restore Settings	Once settings have been saved, they can be restored by calling the report and passing the Link Param: `rdDgLoadSaved = <filename>` Once again, the parameter name is case-sensitive. You can use this method to provide a "default" grid configuration for your users: configure the grid, save its settings, distribute the saved settings file with the report, and call the report with the rdDgLoadSaved parameter in the URL.

The following XML source code example demonstrates the use of these special parameters:

```
<Division ID="divControlMenu">
 <LineBreak LineCount="2" />
 <Label Caption="Reset" ID="lblReset">
 <Action Type="Report"
ID="actRestart">
 <Target
Type="Report" ID="tgtRestart" />
 <LinkParams rdDgReset="True" />
 </Action>
 </Label>
 <Label Caption=" | " ID="bar"
/>
 <Label Caption="Refresh Data"
ID="lblRefreshData">
 <Action Type="Report"
ID="actRefreshData">
 <Target
Type="Report" ID="tgtRefresData" />
 <LinkParams rdDgRefreshData="True" />
 </Action>
 </Label>
 <Label Caption=" | " ID="bar"
/>
 <Label Caption="Save"
```

```
ID="lblSave">
 <Action Type="Process"
Process="MyTasks" ConfirmMessage="Save the Grid?" TaskID="SaveDG"ID="actSave"/>
 </Label>
 <Label Caption=" | " ID="bar"
/>
 <Label Caption="Restore"
ID="lblRestore">
 <Action Type="Report">
 <Target Type="Report"
/>
 <LinkParams
rdDgLoadSaved="@Cookie.DgFilename~" />
 </Action>
 </Label>
 <LineBreak LineCount="2" />
</Division>
```

When saving the grid settings in a Process task, using the **Procedure.Save Dimension Grid** element, you need to provide a fully-qualified file path and file name (including an .xml file extension) in the **Filename** attribute. For example:

```
C:\inetpub\wwwroot\SampleDimensionGrid\SavedGridData\mySettings.xml
```

The @Function.AppPhysicalPath~ token may be useful here.

However, when restoring saved settings, two separate values are used so that you do not have to expose the file path in your query string:

1. Set the Dimension Grid element's **Saved Dimension Grid Folder** attribute to the fully-qualified file path for the *folder* that contains the stored settings file. For example:

   ```
   C:\inetpub\wwwroot\SampleDimensionGrid\SavedGridData
   ```

   The @Function.AppPhysicalPath~ token may be useful here.

2. In the Process task that restores the saved settings, call the report definition that contains the Dimension Grid and pass the link parameter **rdDgLoadSaved** mentioned earlier, and set its value to the filename, with .xml extension, only. Do not include any path information here. For example: `mySettings.xml`

The Dimension Grid will automatically combine the attribute value and the request variable to obtain the fully-qualified path and filename for the saved settings file.

Customizing Dimension Grid Appearance

Dimension Grid appearance can be changed most easily by applying a theme to the report definition. Most of the screen shots in this topic were taken with the *Signal* theme applied.

You can create your own custom theme, based on a standard theme, using the *Using the Theme Editor* tool.

Changing Appearance Using Style Classes

Dimension Grid appearance can be also customized using **style classes** and Logi Studio provides the following standard style sheet for all Dimension Grids:

```
<YourAppFolder>\rdTemplate\rdOlapGrid\rdDgStyle.css
```

Developers can override classes in this style sheet by *copying* them to their application style sheet and modifying them there.

 Do not make changes in the standard style sheet in the rdTemplate folder.

Changing Appearance Using Template Modifiers

The Dimension Grid element uses a "template file" to define certain element properties that are not otherwise available as attributes to the developer for modification. These include language- and culture-specific **Caption** attributes that you may want to change for locale-based reasons (or you may simply want to change the captions to better suit your application).

The element's **Template Modifier File** attribute identifies a custom XML file developers can create containing elements that will override the same elements in the template file.

For example, the Dimension Grid template file:

`<yourAppFolder>\rdTemplate\rdOlapGrid\rdDgTemplate.lgx`

contains several Label elements. One of them has an ID of "lblChartDrillDown"; this controls the caption that says "Click the chart to drill down.". This text can be modified by changing the **Caption** associated with that Label element.

For example, to change the Label text from its default to the French version, Cliquez sur le graphique pour forer vers le bas", create your own XML file, identify it in the Dimension Grid element's Template Modifier File attribute, and add this code to it:

```
<TemplateModifier>
<SetAttribute ID="lblChartDrillDown" Caption="Cliquez sur le graphique pour forer vers le bas" />
</TemplateModifier>
```

You can set the attributes for any number of elements in this file; examine the rdDgTemplate.lgx file to learn the ID and Caption attributes available. The template modifier file can be in any folder accessible to the web application; if a fully-qualified file path is not provided in the Template Modifier File attribute value, then the application expects it to be in your project's _SupportFiles folder.

More detailed information about template modifier files can be found in *Template Modifier Files*.

User Input Elements

User Input elements produce input controls, which allow users to interact with applications at runtime. They're often used to often to provide data selection and filtering criteria.

The following topics discuss each of the Logi Info Input elements:

- Input Hidden
- Input Text
- Input Password
- Input Date
- Date Picker
- Input Email
- Input Number (Mobile Only)
- Input Telephone (Mobile Only)
- Input Time
- Input File Upload
- Input Text Area
- Input Check Box
- Input Select List
- Input Check Box List
- Input Combo List
- Input Radio Buttons
- Input Slider
- Input Color Picker
- Input Chart.List

- Input Chart.Range
- Input Selection Point & Range

About Input Elements

Logi Info Input elements are based on the standard HTML input control types and provide useful interactivity. They can be used to gather user input at runtime in order to login to an application, to filter or restrict data, to modify the style or appearance of a report, to insert new or update existing data, and more.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

When you include Input elements in a report definition, a ;<FORM> tag is automatically added to the HTML page generated. Following standard HTML behavior, when a user clicks links or buttons that reload the report or pass control to another definition, the form is submitted, along with all of the input control values that have been entered.

 When an report page is submitted, the values of *all* of its Input elements are *automatically* passed to the next Report or Process definition. DO NOT use **Link Parameters** to try to pass the entered values - they'll become null.

In the *next* Report or Process definition, you can access the entered values using Request tokens. For example, if an Input Text element has an element ID of *inpLastName*, its value is available in the token @Request.inpLastName~.

Because the values are passed using HTTP POST, they will *not* appear in the URL. They are clearly shown, however, in the Debugger Trace Page when you turn debugging on, and this is a great tool for determining what's happening with user input values.

Due to the stateless nature of HTML pages, you generally have to submit (or refresh) a page before its value is usable in its own report definition. There are some exceptions to this; it's possible to use JavaScript to get and set the values of Input elements without submitting the page. This, and many other useful techniques, are described in "Working with User Input Elements" on page 642.

Our Sample Center is also an good resource. Special mobile input elements are discussed in *Logi Mobile Reports*.

Input Hidden

This is the *invisible* member of the Input elements family, so some might say it's debatable whether it's actually a *user input* element at all. However, it's extremely useful.

Using It

This element does not appear on the report page and has no caption but it can serve either of two purposes:

1. It can hold any data value that you want to pass discreetly. Because the POST method is used when the current page is submitted, the data does not appear in the URL's query string. Contrast this behavior to a **Link Parameter**, which *is* passed as part of the query string and is therefore visible in the browser's URL display. Input Hidden is especially useful when passing data from within Data Tables to another Report definition or a Process task (see "Working with User Input Elements" on page 642) and its value can be set using JavaScript.
2. It can act as a "change flag", indicating to another Report definition or Process task whether or not any input data has been *changed* prior to the page being submitted. This can be used with conditional processing to determine, for example, whether or not a database record needs to be updated. For more information, see Working with Change Flags in "Working with User Input Elements" on page 642.

This element's two attributes are *required*: a unique element **ID** and a **Default Value**, which may be set using tokens.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

To turn off formula support for the Input Hidden element, add rdIgnoreFormula=True in the Template Modifier File. This ensures the text in the Input Hidden displays exactly as you enter it. Currently, the only character supported is the equal sign (=). Below is a usage example for rdIgnoreformula:

```
<TemplateModifier>
<SetAttribute XPath="//InputHidden[@ID='rdCustomColumn1']" rdIgnoreFormula="True"></SetAttribute>
<SetAttribute XPath="//Row[@ID='rowButtons']/Column/Division[@ID='divDone']/Label[@ID-
='lblDone']/Action[@ID='rdActionDoneEmail']" Javascript="if(rdSaveSchedule(&apos;Refresh&apos;,
&apos;true&apos;))rdHideSchedulePanel()"></SetAttribute>
<SetAttribute XPath="//Row[@ID='rowButtons']/Column/Division[@ID='divRunNow']/Label[@ID-
='lblRunNow']/Action[@ID='rdActionRunNow']" Javascript="rdRunNowSaveFirst(&apos;true&a-
pos;)"></SetAttribute>
</TemplateModifier>
```

Getting Its Data

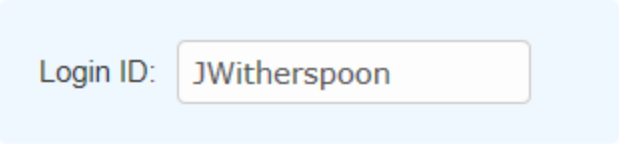
Input Hidden's value will be available in the next Report or Process task in a @Request token. For example, if the element's ID is set to *hdnChangeFlag*, then its data will be available in the token `@Request.hdnChangeFlag~`.

More Information

For additional information, see the Element Reference entry for Input Hidden.

Input Text

The **Input Text** element is probably the most commonly-used of the group and allows users to type in characters. It looks like this in use:



Using It

The Input Text element attributes include:

Attribute	Description
ID	(Required) The element ID, unique within the definition.
Caption	Specifies text that will be displayed to the left of the text input box.
Caption Class	Specifies one or more style sheet classes that will be applied to the element's Caption, if a different class than that applied to the Input element as whole is desired.
Change Flag Element ID	Specifies the ID of an **Input Hidden** element that will be used as a "change flag".

Attribute	Description
Class	Specifies one or more style sheet classes to be applied to the entire Input element.
Default Value	Specifies the default value that will appear in the text box. Can be used with tokens to display values from data, or passed from another report or a process. Overrides the Save In Local Storage attribute, *sodo not use them at the same time.*
Format	Specifies a format for the Default Value text displayed. This does *not* act as a *mask* (i.e. it does not force entry of characters in a specific pattern) nor does it change the visible data after entry. More information can be found in *Format Data*.
Input Max-imum Length	Specifies the maximum number of characters than can be entered into the input box. If this is greater than the Input Size value, text will automatically scroll horizontally in the box.
Input Size	Specifies the physical width of the input box, in characters, on the screen.
Placeholder	Specifies the text of a short hint or caption, often used to describe the expected value, that appears *within* the input box when it's empty. The text is displayed in a gray color and you may care to use it instead of a Caption. The text disappears when data is entered and reappears if the entered data is deleted.
Save In Cookie	When *True*, stores the data entered by the user in a cookie, named after the element ID. The cookie can then be used to set the element's default value, so that previously entered data is retrieved, by setting the Default Value attribute to *@Cookie.myInputId~*.
Save In Local Storage	When *True*, stores the data entered by the user in the browser's "local storage" (if the browser supports this HTML5 technology). Data stored this way is retained between sessions and automatically restored as the

Attribute	Description
	Input element's default value when the page is redisplayed. The local storage size limit is approximately 5 MB and all values are stored as strings. Save In Local Storage is overridden by the Default Value attribute. *Do not use them at the same time.*
Security Right ID	Controls access to this element when using Logi Security. Provide the ID of a Right defined in the application's _Settings definition and only users that have a Role referenced in the Right will be able to view the element. Multiple Right IDs, separated by commas, may be entered.
Tooltip	Specifies text that appears when the user hovers the pointer over the input box. Tokens may be used here.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

The Input Text element can be used with the AutoComplete element (see "Working with User Input Elements" on page 642), to assist users in entering text based on data. This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier.

Getting Its Data

The text entered into this element will be available in the next Report or Process task by using an @Request token. For example, if the element's ID is set to *txtFirstName*, then its data will be available as the token `@Request.txtFirstName~`.

Input Text Delimiter Child Element

The **Input Text Delimiter** child element is available to be added beneath an **Input Text** element to control special characters in the input data and its attributes are:

Attribute	Description
Class	Specifies an optional class to be applied to each delimited value within the Input Text control. For example, if using a Logi theme, the *ThemeLinkButton* class can be specified here to cause the values to appear as "pills".
Delimiter	Specifies the single character to be used to delimit multiple values entered into the Input Text control. Default value: *comma*.
Escape Character	Specifies the single character to be used as an escape character in the values entered in the input value. It will be *ignored* if no Qualifier attribute value is specified, but is *required* if a Qualifer value is specified. The Escape Character will be used to "escape" any instances of the Qualifier found within the input value. For example, if the Escape Character is a backslash and the Qualifier character is a double-quote and the input string is: `Lewis "Chesty" Puller` When the page is submitted, the resulting string value will have the Escape Character inserted and become: `Lewis \"Chesty\" Puller`
Qualifier	Specifies the single character that will enclose complete data values, allowing the Delimiter character to be

Attribute	Description
	included in the data without being processed as a delimiter. Leave this value blank only if there is no chance that the Delimiter character can be used in the data. For example, consider an addresses that includes a city and state, separated by a comma: if the Delimiter is a comma, the city and state will be processed as twovalues when the page is submitted: `McLean, Virginia` = values: McLean,Virginia However, if the Delimiter is not a comma and the Qualifier is a double-quote, only one value will be processed: `McLean, Virginia` = value: "McLean,Virginia"

When an Input Text element has both **Input Text Delimiter** and **Auto Complete** child elements, the Auto Complete element's **Multiple Selections** attribute is ignored and assumed *True*, and the Delimiter specified in the Input Text Delimiter element will be used (although the delimiter will not be visible).

More Information

For additional information, see the Element Reference entry for Input Text.

Input Password

The purpose of this element is to allow the entry of **passwords**. It's identical to the Input Text element except that it *does not show* the characters being typed. Instead, a black dot is shown for each character entered. This is a commonly-accepted method of preventing a bystander from reading a password as it's entered.



Using It

The Input Password element's attributes are the same as those of the Input Text element and are not repeated here. This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier.

Getting Its Data

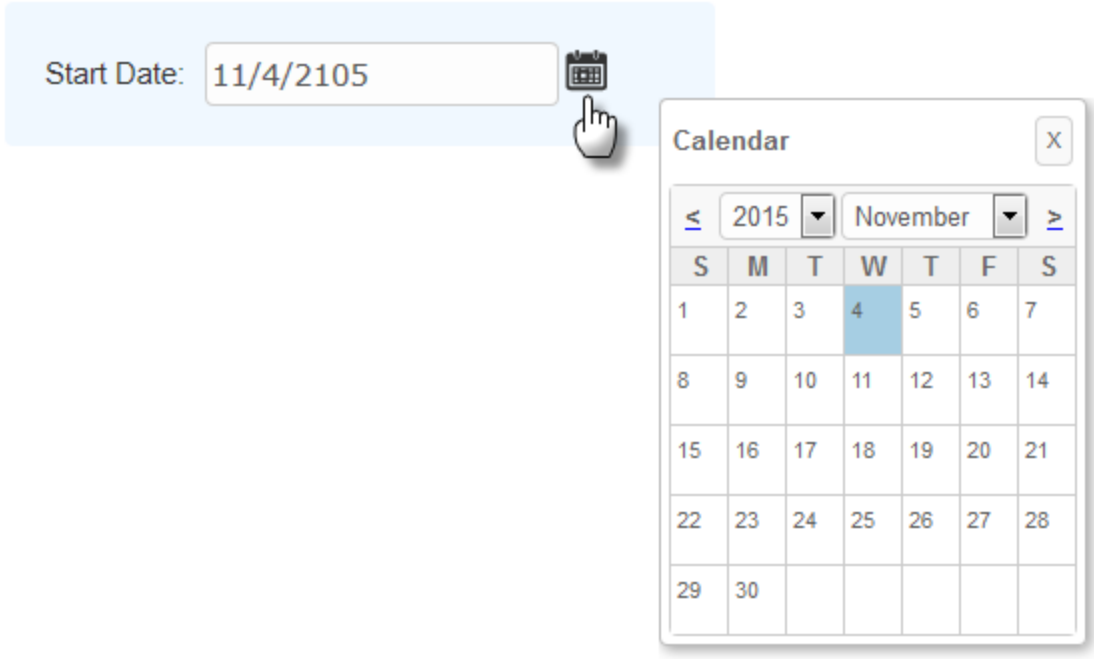
The text entered into this element will be available in the next Report or Process task by using an @Request token. For example, if the element's **ID** is set to *txtPassword*, then its data will be available as the token `@Request.txtPassword~`.

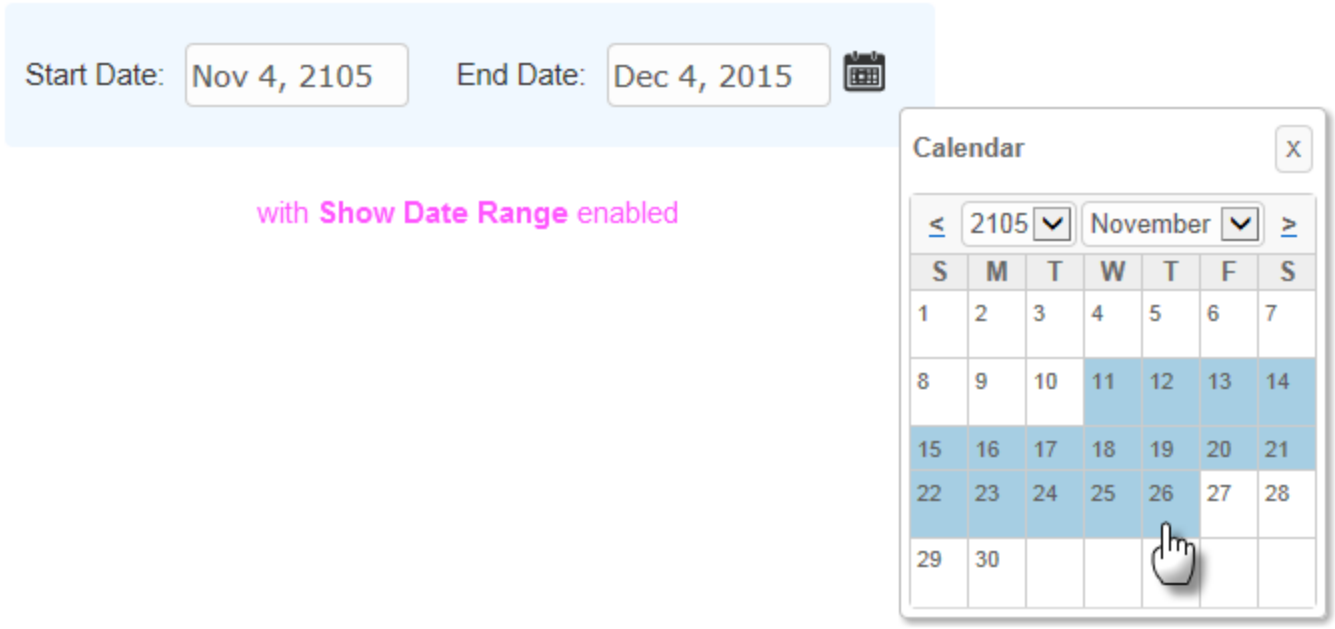
More Information

For additional information, see the Element Reference entry for Input Password.

Input Date

The purpose of this element is to allow the entry of a **date** or **date range**. It's similar to the Input Text element and a date can be typed right into it. It can also include an icon or link beside it which allows a date to be selected from a small pop-up calendar.





with **Show Date Range** enabled

In the pop-up calendar, the current date is highlighted with one color and the selected date (or range of dates) is highlighted with another. Colors can be configured with CSS by overriding style classes found in `<yourApp>\rdTemplate\rdCalendar\rdDataCalendarStyle.css`.

Using It

The Input Date element shares many of the attributes of the Input Text element and also has these additional attributes:

Attribute	Description
Calendar Caption Format	Specifies the format of the text above the popup calendar. Set to *None* to prevent display of the caption row entirely.
Calendar Link Calendar Link Class Calendar Link Type	These three attributes specify the type of popup calendar link, if any, that appears beside the text box. A small calendar image is provided or a custom image or text can be used.
Default Value	Specifies the default date that will appear in the text box. Can be used with @Function.Date~ token to display the current date in *Short Date* format.
End Date Default Value	Specifies the default value for the "end" date in a date range, if used.

Attribute	Description
End Date Range Caption	Specifies the caption for the "end" date text box, if used.
End Date Range ID	Specifies the ID to be associated with the "end" date text box, if used. This is the ID that will be used with a @Request token when the page is submitted.
Input Date Reformat	Causes the date entered to be reformatted into one of several formats *before* being made available as a request variable. More information can be found *Format Data*.
Number of Dropdown Years	Specifies the number of years that will appear in the Year drop-down list in the pop-up calendar associated with the input element. This value creates a range of years with the current year in the middle of the list.
Show Date Range	This attribute causes two text input boxes for dates to be displayed. If the popup calendar feature is used (Calendar Link Type = *Image* or *Text*), the first date selected becomes the value for the starting date; the second date selected sets the value for the ending date. Additional attributes are available that allow you to specify a Caption, Default Value, and ID for the second text input box. Dates can range over 1, 2, or 3 months.

The **Input Date Calendar** element can be added as a child of Input Date and used to configure the pop-up calendar. Input Date Calendar has the same attributes and behavior as the Date Picker element, which is described in "Date Picker" on page 585.

The control does not provide an "input mask" to control the format of what can be typed into it. Instead, developers must validate the date entered when the page is submitted, using Validation elements, or when various DHTML events occur, using Event Handler elements. For more information about validation and events, see "Working with User Input Elements" on page 642.

This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier, and a special element, **Validation.Date**, is available as a child of Input Date, to ensure that a valid value is entered by the user.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

The data placed into this element will be available in the next report or process by using an @Request token. For example, if the element's **ID** is set to *txtHireDate*, then its data will be available as the token `@Request.txtHireDate~`.

If **Show Date Range** is being used to allow entry of a date range, then the starting date is available using an @Request token that includes the Input Date element's ID, and the ending date is available using an @Request token that includes the ID specified in the **End Date Range ID** attribute.

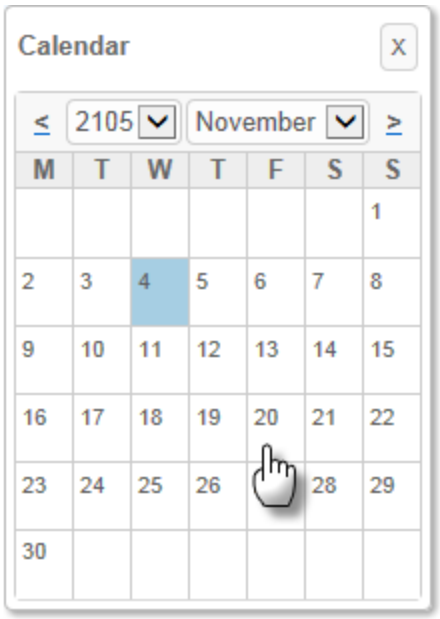
Globalization and Localization

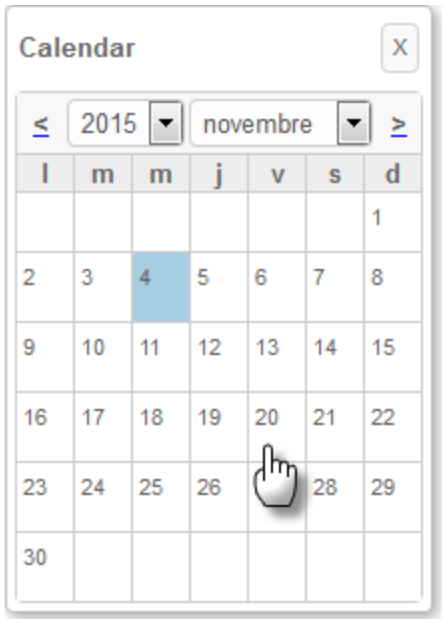
The format and presentation of calendars and dates varies widely depending on country and/or language. Some aspects of the date-related input elements are controlled by the "preferred language" settings of the **browser** being used and others may be affected by the default date and time format settings of the client **operating system**.

At the application level, date-related aspects of your Logi application can also be changed by adding and configuring a **Globalization** element in the application's _Settings definition. For example, it has two attributes, **Default Input Date Format** and **Default Input Date Reformat**, that can be used, if desired, to specify a format with global scope.

Setting the Globalization element's **First Day of Week** attribute will change the display of the calendars associated with Input elements. If the attribute value is set to 1, then *Monday* will be the first day in the calendar; the default is *Sunday*. More information about the Globalization element can be found in Internationalization and Localization .

In addition, the browser's preferred **language setting** will affect the pop-up calendar display and date value format:





In the examples shown above, a Globalization element has been used to change the first day of the week to *Monday* and the browser's preferred language has been set first to *English* and then to *French*. Notice that the **month name** and **day-of-weekabbreviations** in the calendar changed based on the browser's preferred language setting.

In addition, the actual value returned after the user clicks on a date, shown below the calendars, also reflects the language preference in any text it contains - "Nov" vs. "nov.". The element's **Format** attribute, of course, controls the format of the date value returned into the element's input text box after the mouse click.

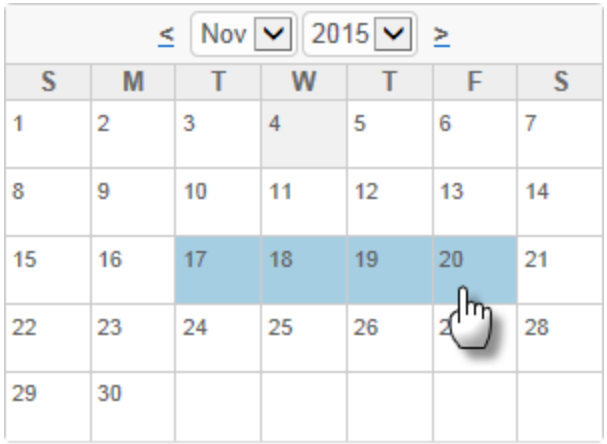
 In a "globalized date" scenario, when passing/submitting dates from your input controls to other definitions for use within tokens, as filters on queries, or within elements such as the **Compare Filter**, we *highlyrecommend* utilizing the ISO 8601 standard date format (yyyy-MM-dd) to avoid any locale/browser date conflicts that might arise. The input element's **Input Date Reformat** attribute can be used for this purpose.

More Information

For additional information, see the Element Reference entry for Input Date.

Date Picker

The **Date Picker** element displays a calendar on the page, rather than a text input box, and allows the user to select a date. It can be configured to allow selection of a range of dates by selecting the starting and ending date. Its behavior is closely related to the Input Date element's pop-up calendar.



The current date is highlighted with one color and the selected date (or range of dates) is highlighted with another. Colors can be configured with CSS by overriding style classes found in `<yourApp>\rdTemplate\rdCalendar\rdDataCalendarStyle.css`.

Using It

The Date Picker element shares many of the attributes of other Input elements and also has these additional attributes:

Attribute	Description
Border	Specifies the width of the border line, in pixels, drawn around the calendar. The default value is *1 pixel*.

Attribute	Description
Calendar Caption Format	Specifies the format of the text above the popup calendar. Set to *None* to prevent display of the caption row entirely.
Cell Spacing	Specifies the width, in pixels, of the space between calendar cells.
Dropdown Year and Month	Specifies if the calendar header will have a drop-down list for year and month selection, allowing the user to change years and months. When set to *False*, the drop-down lists are replaced by a caption with the current calendar year and month names. Default: *True*
End Date Default Value End Date Range ID	Specifies the default value for the "end" date in a date range and the ID to be used for that value.
Input Date Reformat	Causes the date selected to be reformatted into one of several formats *before* being made available as a request variable. Date format information relevant to this attribute can be found in *Format Data*.
Number of Dropdown Years	Specifies the number of years to be listed in the calendar's Year drop-down list.
Number of Months	Specifies if 1, 2, or 3 monthly calendars are displayed. If Show Date Range (see below) is *True*, highlighting for selected dates in different months can span the calendars.

Attribute	Description
Show Date Range	Specifies if a single date can be selected, or if a range of dates can be selected. If set to *True*, then all days in the range of dates selected will be highlighted. Default: *False*
Weekday Caption Format	Specifies the format for the weekday name captions that appear across the top of the calendar. The default value is *dd*, which displays a single letter for each day; use *ddd* to display a weekday name abbreviation ("Mon"); use *dddd* to display the full weekday name ("Monday"); and set the value to *None* to prevent display of the weekday names.
Width Width Scale	Specifies the width of the element's calendar, in pixels or as a percent of its containing element. If left blank, a default value is automatically applied.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

The Date Picker automatically adds Input Hidden elements to your definition and they are assigned the date(s) selected in the calendar. The IDs of the Input Hidden elements are based on the ID attributes of the Date Picker element. The selected dates will be available in the next Report or Process task by using an @Request token. For example, if the Date Picker element's **ID** is set to *inpHireDate*, then the selected date will be available as the token `@Request.inpHireDate~`.

If **Show Date Range** is enabled to allow selection of a date range, then the starting date is available using an @Request token that includes the Date Picker element's ID, as above, and the ending date is available using an @Request token that includes the ID specified in the **End Date Range ID** attribute.

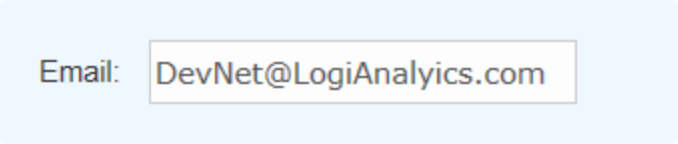
More Information

The globalization and localization information presented about the Input Date element also applies to the Date Picker element.

For additional information, see the Element Reference entry for Date Picker.

Input Email

The purpose of this element is to allow the entry of one or more **email addresses**. It's similar to the Input Text element and addresses can be typed right into it.



Using It

This element's attributes are the same as those of the Input Text element with include one additional attribute:

Attribute	Description
Multiple Addresses	When set to *True*, multiple email addresses, separated by commas or semi-colons, may be entered.

This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier, and a special element, **Validation.Email**, is available to ensure that valid email address values are entered by the user.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frame-works or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

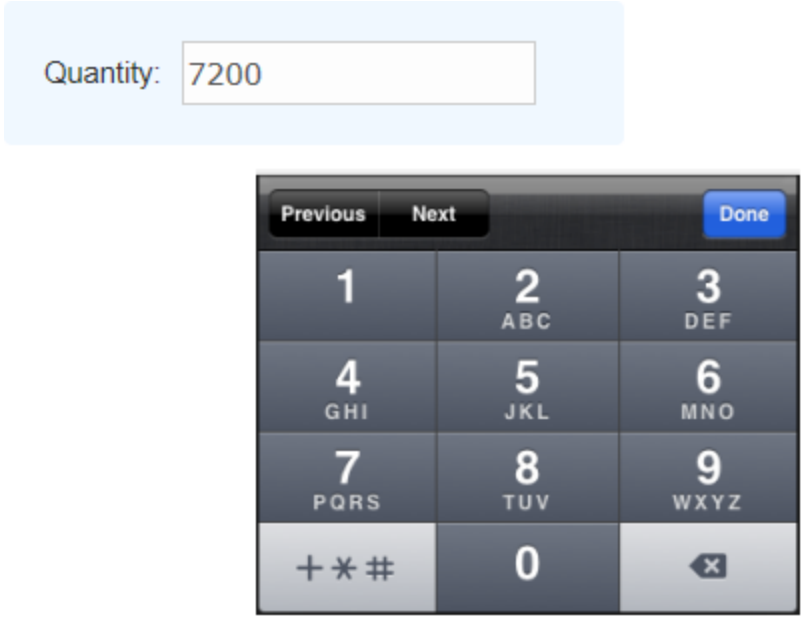
The data entered into this element will be available in the next Report or Process task by using an @Request token. For example, if the element's **ID** is set to *inpEmail*, then its data will be available as the token `@Request.inpEmail~`.

More Information

For additional information, see the Element Reference entry for Input Email.

Input Number (Mobile Only)

Available only in Logi Info **Mobile Report** definitions, this element allows the entry of a number and may trigger a special data entry control in the mobile device interface. Otherwise, it's similar to the Input Text element and numbers can be typed right into it.



Using It

This element's attributes are the same as those of the Input Text element.

This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier, and a special element, **Validation.Numeric**, is available to ensure that valid numeric values are entered by the user.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

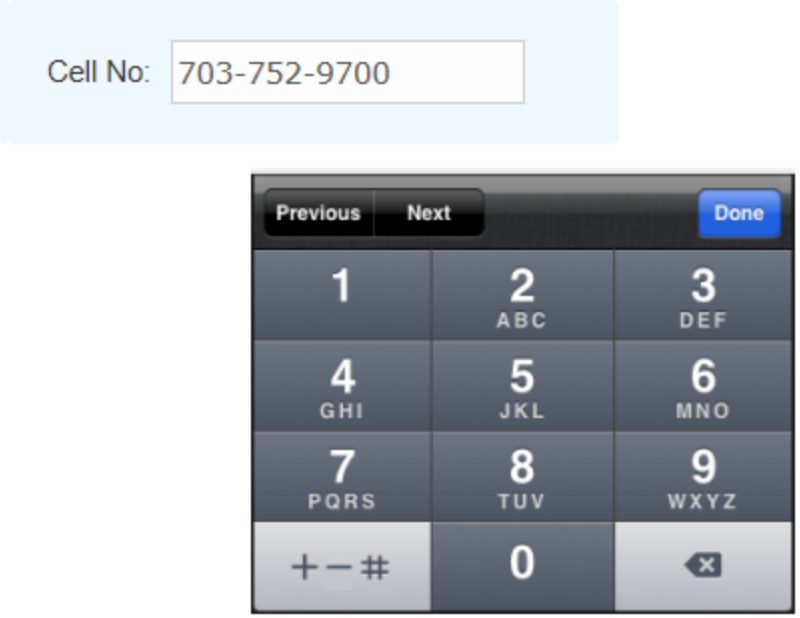
The data entered into this element will be available in the next Report or Process task by using an @Request token. For example, if the element's **ID** is set to *inpQuantity*, then its data will be available as the token `@Request.inpQuantity~`.

More Information

For additional information, see the Element Reference entry for Input Number.

Input Telephone (Mobile Only)

Available only in Logi Info **Mobile Report** definitions, this element allows the entry of a telephone number and may trigger a special data entry control in the mobile device interface. Otherwise, it's similar to the Input Text element and numbers can be typed right into it.



Using It

This element's attributes are the same as those of the Input Text element. This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

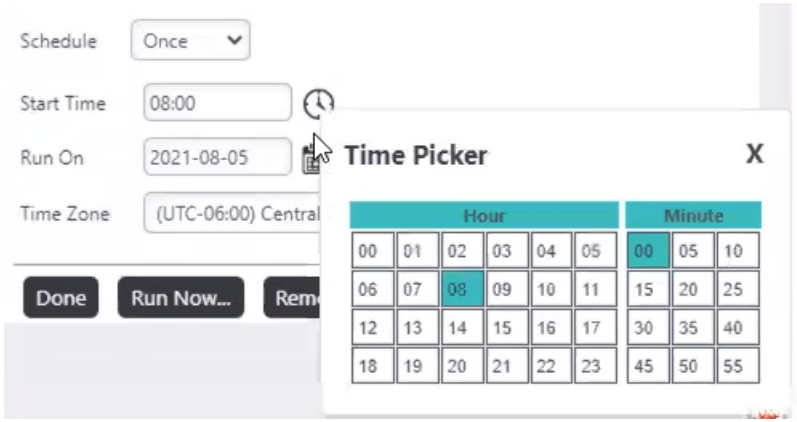
The data entered into this element will be available in the next Report or Process task by using an @Request token. For example, if the element's **ID** is set to *inpCellNo*, then its data will be available as the token `@Request.inpCellNo~`.

More Information

For additional information, see the Element Reference entry for Input Telephone.

Input Time

The purpose of this element is to allow the entry of **time**. It's similar to the Input Text element and a time can be typed right into it. It can also include an icon or link beside it, which allows a time to be selected from a small pop-up panel.



As shown above, the pop-up panel has separate sections for hours and minutes, and selections are highlighted with a special color. Colors can be configured with CSS by overriding style classes found in `<yourApp>\rdTemplate\rdClock\rdTimePickerStyle.css`.

Using It

This element shares many of the attributes of the Input Text element and also includes these special attributes:

Attribute	Description
Clock Link	These three attributes specify the type of popup link, if any, that appears beside the text box. A small clock

Attribute	Description
Clock Link Class Clock Link Type	image is provided, or a custom image or text link may be used.
Default Value	Specifies the default time that will appear in the text box. Can be used with @Function.Time~ token to display the current date in `mm:hh:ss` format.
Format	Specifies one of the three standard formats (*Short*, *Medium*, or *Long Time*) or your own custom format. More format information can be found in *Format Data*.
Input Time Reformat	Causes the time entered to be reformatted into one of several formats *before* being made available as a request variable.

The **Input Time Clock** element can be added as a child of Input Time and used to configure the popup panel. Input Time Clock has the following optional attributes:

Attribute	Description
Clock Caption	Specifies a specific title in the popup panel. Default: *Time Picker*
Hours Header	Specifies a title for the Hours section. Default: *Hours*
Minutes Header	Specifies a title for the Minutes section. Default: *Minutes*

Attribute	Description
Seconds Header	Specifies a title for the Seconds section. Default: *Seconds*
Show Minutes	Specifies if the Minutes section is displayed. Default: *True*
Show Seconds	Specifies if the Seconds section is displayed. Default: *False*
Time Picker Hours	Specifies the hour scheme: 12-hour, 24-hour, or whatever the browser's Locale setting requires. Default: *BrowserLocale*

This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier, and a special element, **Validation.Time**, is available to ensure that a valid time value is entered by the user.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

The data placed into this element will be available in the next report or process by using an @Request token. For example, if the element's **ID** is set to *inpStartTime*, then its data will be available as the token `@Request.inpStartTime~`.

Globalization and Localization

The format and presentation of times varies depending on country and/or language. Some aspects of the time-related input elements are controlled by the "preferred language" settings of the **browser** being used and others may be affected by the default time format settings of the client **operating system**.

At the application level, time-related aspects of your Logi application can also be changed by adding and configuring a **Globalization** element in the application's _Settings definition. For example, it has two attributes, **Default Input Time Format** and **Default Input Time Reformat**, that can be used, if desired, to specify a format with global scope.

More information about the Globalization element can be found in *Internationalization and Localization* .

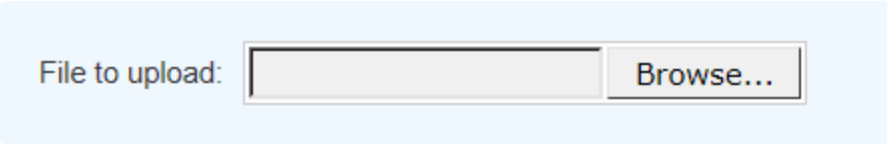
 In a "globalized time" scenario, when passing/submitting dates and times from your input controls to other definitions for use within tokens, as filters on queries, or within elements such as Condition Filters, we *highly* recommend utilizing the ISO-standard date format (hh:mm:ss, with leading zeros) to avoid any locale/browser time conflicts that might arise. The input element's **Input Time Reformat** attribute can be used for this purpose.

More Information

For additional information, see the Element Reference entry for Input Time.

Input File Upload

The purpose of this element is to allow client system files to be *uploaded* to the web server. It's similar to the Input Text element but a value cannot be entered directly. Clicking the text box or the included "Browse..." button launches the client file explorer tool so the user can navigate through the file system to identify the desired file. The data selected must be a fully-qualified file path and file name.



Using It

The element has many of the attributes of the Input Text element but none that are special or unique. It can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frame-works or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

When the page containing this element is submitted, an **RFC 1867** file upload occurs. The uploaded file is then available and must be processed or saved using a Process task and the **Procedure.Save File Upload** element. A variety of information about

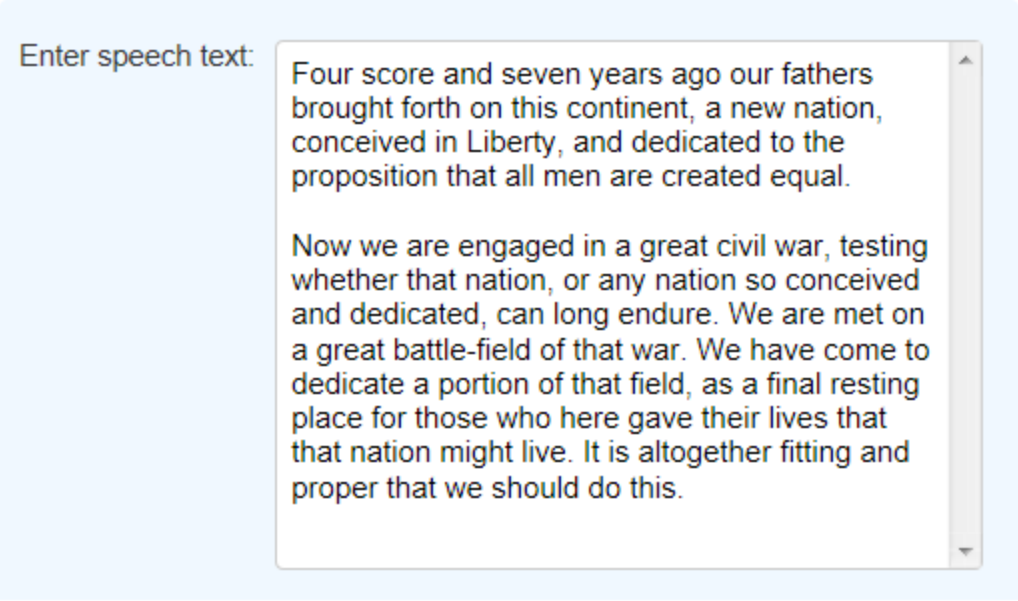
the uploaded file (name, type, size, etc.) is then available using @Procedure tokens. For detailed information about these tokens and file upload processing, see Upload Files to the Web Server.

More Information

For additional information, see the Element Reference entry for Input File Upload.

Input Text Area

The purpose of this element is to allow *multi-line* entry of text. It's very similar to the Input Text element but allows paragraph-style text entry.



Using It

This element has most of the attributes of the Input Text element and includes these two additional attributes:

Attribute	Description
Input Columns	Specifies the width of the input box, in characters.

Attribute	Description
Rows	Specifies the number of lines of text that will be visible in the input box, affecting its height on the page. Text entered will automatically wrap to the next line if it exceeds the width of the text box. If the number of lines of text entered exceeds this attribute's value, vertical scroll bars are displayed and the text scrolls.

This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

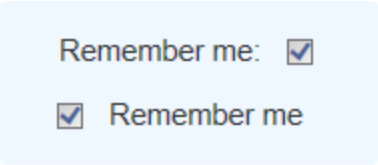
The data placed into this element will be available in the next report or process by using an @Request token. For example, if the element's **ID** is set to *txtSpeech*, then its data will be available as the token `@Request.txtSpeech~`.

More Information

For additional information, see the Element Reference entry for Input Text Area.

Input Check Box

The purpose of the Input check box element is to provide a single, square box (with a caption) that can be checked or unchecked.



Using It

This element has many of the Input Text element's attributes and these attributes you should note:

Attribute	Description
Caption	Specifies identifying text that is displayed *to the left* of the check box (top example shown above). To display a caption that *follows* the check box (bottom example), leave this attribute blank and add trailing **Space** and **Label** elements.
Checked Value	Specifies the value that will be passed to the next page or process task as a Request parameter if the check box is checked. No value is passed if the check box is unchecked and the token will not exist, unless an Unchecked Value (see below) is specified.
Default Value	Specifies the default value for the control. If it matches the **Checked Value** attribute, a check symbol will appear in the check box. Can be specified using a Request token for a value passed from another report or a process task, and other tokens.

Attribute	Description
Unchecked Value	Specifies a value that will be passed to the next page or process task as a Request parameter when the item is *not* selected.

This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier. If *multiple* Input check box elements are placed on a page, there is no inherent relationship between them and each operates independently of the others.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

The "state" of the check box, checked or unchecked, can be determined in the next Report or Process task with an @Request token. For example, if the element's **ID** is set to *chkRememberMe* and the check box is checked, then in the next page or process task the token `@Request.chkRememberMe~` will equal the value in the **Checked Value** attribute. If it is not checked, then this @Request token *will not exist,* unless a value has been specified in its **Unchecked Value** attribute. This is an important distinction to note when creating conditional processing that evaluates this token.

More Information

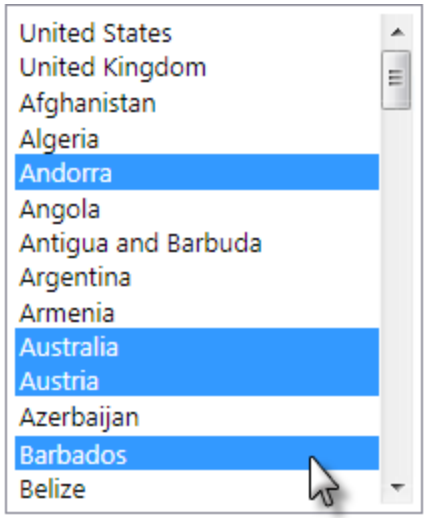
For additional information, see the Element Reference entry for Input Check Box.

Input Select List

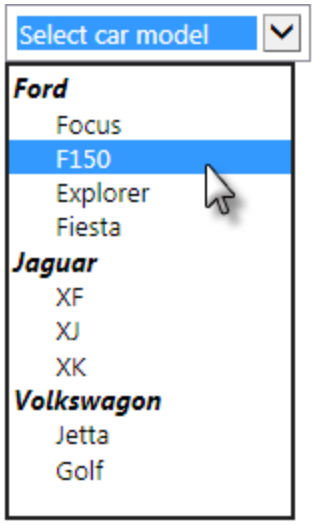
The purpose of the **Input Select List** element is to provide a *drop-down list* or *fixed-length list* of options, one or more of which can be selected. Here are three possible configurations:



Standard drop-down list



Fixed-length list allowing multiple selections



Drop-down list with Grouping

This user Input element operates using **collections of data** and requires a datalayer to provide that data.

Using It

The default configuration is for a drop-down list, but a static list can be created by entering a value in the element's **Rows** attribute.

This element requires a datalayer to provide the items that will appear in the list. If the list of items is short and unchanging, the **DataLayer.Static** element can be used to "hard code" the items. If the list is longer, the **DataLayer.XML** element can be used to read the data from an XML file. If the list is dynamic, elements such as **DataLayer.SQL** can be used to retrieve the items from a datasource, possibly with grouping to provide unique values.

If the datasource provides one, the value passed to the next Report or Process task can be *different* from the displayed text. The control automatically sizes itself to accommodate the widest display text.

The element has many of the attributes common to all Input elements and includes these additional attributes:

Attribute	Description
Caption Column	(Required) Specifies the name of the datalayer column that contains the text to be displayed in the selection list.
Value Column	(Required) Specifies the name of the datalayer column that contains the value to be passed to the next report or process task. For example, if "United Kingdom - Pound" is selected in the list, the value "GBP" might be passed to the next report. This allows the flexibility to display and pass different values and even different types of values, such as numbers.
Default Value	Specifies a default *value* (not the displayed caption text). If it matches the value of an item in the list, then that item will be pre-selected in the control. Can be used with @Request tokens passed from another report or a process task, and other tokens. To specify multiple default selected values in an Input Select List configured for multiple selections, enter a comma-separated list of values. Do not include any spaces in the list.

Attribute	Description
	To use a data-driven set of default selected values, see the Default Values section below.
Group Caption Column	Specifies the datalayer column that provides the text for group headings in the option list. The is the same column used by the datalayer's Group Filter to group the options.
Group Class Column	Specifies the datalayer column that provides style sheet class names for the list's group captions. This allows each group caption to have its own styling, based on the data.
Include Blank Include Blank Caption Include Blank Value	These three attributes allow you to display and use a default item that is not based on data from the datalayer, for example, an instruction such as "Make a selection". See the notes below regarding use of these attributes under "Getting Its Data".
List Captions Element ID	Specifies the element ID of an Input Hidden element, which will contain the Caption Column values for the items selected when the definition is submitted. The values will be available in a @Request token using the same ID.
Multiple Selections	Specifies if multiple list items can be selected simultaneously, using the Ctrl or Shift keys. Default: *False*.
Rows	Specifies the number of items to display in a static list, determining the height of the control on the page. If the number of items in the list exceeds this value, vertical scroll bars are enabled and the control will scroll. The

Attribute	Description
	default blank setting for this attribute causes a drop-down list to be displayed instead.
Tooltip	Specifies text to be displayed as a tooltip when the mouse is hovered over the control and, if no Tooltip Column is specified, over the list of options.
Tooltip Column	Specifies the datalayer column that contains text to be shown as a tooltip when the mouse is hovered over *each item* in the list of options. Can be used with or without the Tooltip attribute.

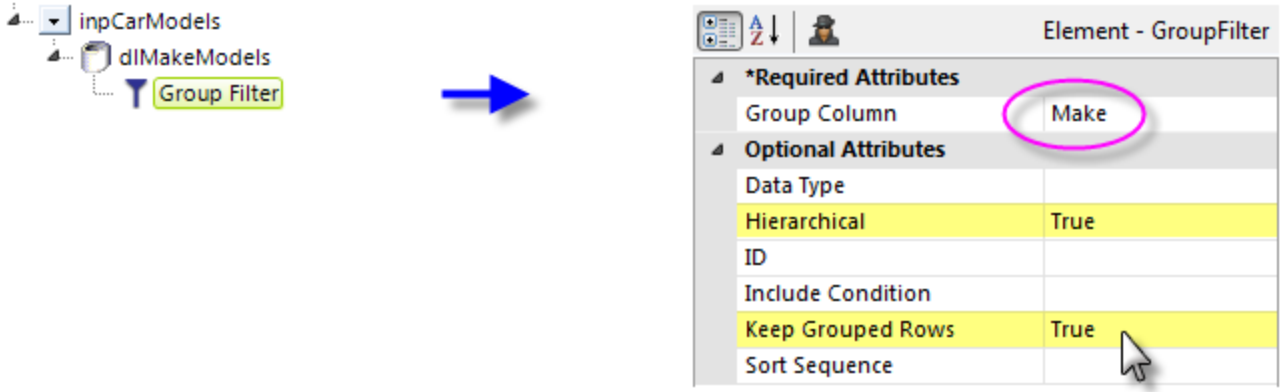
This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

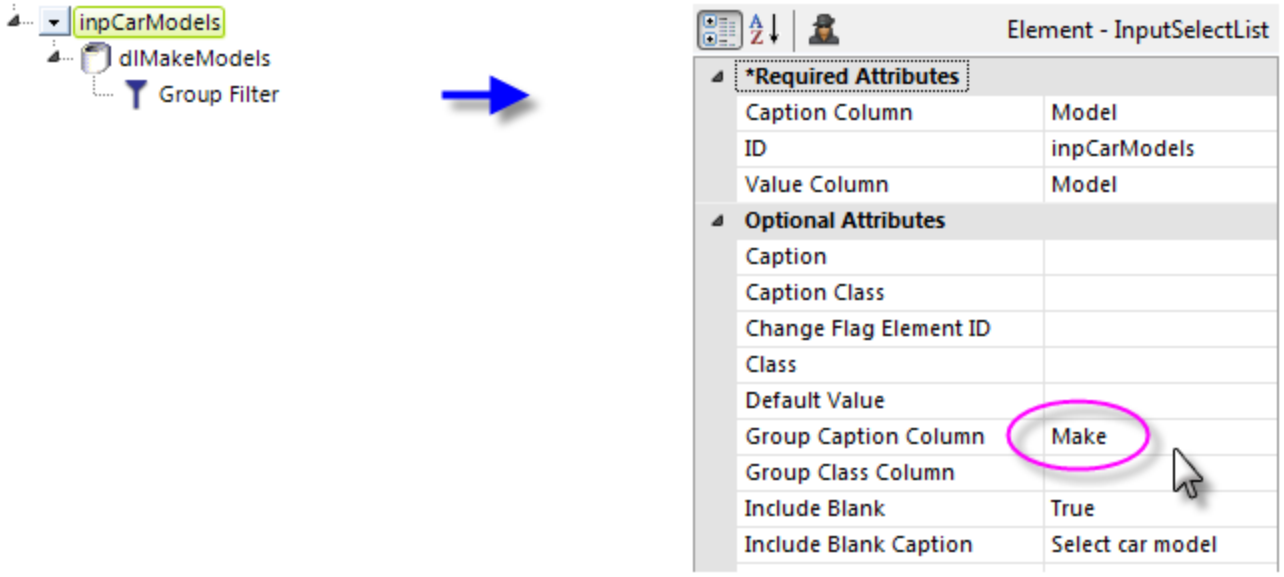
 If the Input Select List is placed under a **Data Table** that produces multiple rows, the Input Select List's datalayer only runs once and thus all select lists will have the same rows. In such an arrangement, you cannot reference @Data tokens from the parent Data Table in the Input Select List's datalayer.

Hierarchical Grouping

Input Select List elements can be configured to list their options in *groups*, with a group caption, as shown earlier.



To use this, a Group Filter must be applied to the data, and its attributes set as shown above. Its **Group Column** value is the datalayer column that will supply the group captions.



The Input Select List element's **Group Caption Column** attribute is configured with the same column name used in the Group Filter, as shown above. The **Group Class Column** attribute gives you the option of setting a different style for each group caption, in the data.

Adding Default Values

A list of default values for the Input Select List can also retrieved from a datalayer. The Input Select List element's **Default Value** attribute must be left blank in order to use this technique.



As shown above, a **Default Values***element*, with its own datalayer, is added as a child of the Input element. Its **Data Column** attribute is set to the name of the datalayer column containing the default values. This has be effect of turning all of the values in all datalayer rows in this column into a comma-separated list which is then fed into the Input element as its default value.

Getting Its Data

The value (from the "Value Column") associated with the selected item (from the "Caption Column") is available in the next Report or Process task with a @Request token. For example, if the element's **ID** is set to *lbxCurrency* then the token `@Request.lbxCurrency~` will equal the selected item's value.

If you want to access the caption(s) of the item(s) selected, instead of or in addition to their values, set the **List Captions Element ID** attribute to the ID of an Input Hidden element. The caption(s) will be placed in this element when the report is submitted and will be available using a @Request token with the same ID. For earlier Info versions, see "Passing Input Select List Captions" on page 683.

 The availability of default data when using the **Blank** attributes varies based on the list type:

If you've specified a Blank Value and nothing is deliberately selected in a *drop-down* list (no Rows value) and the page is submitted, then the item *visible* in the control is considered selected and its value will be available using the @Request token.

However, if you've specified a Blank Value and nothing is selected in a *static* list (Rows > 0) and the page is submitted, then nothing is considered selected and this @Request token *will not exist*. This is an important distinction to note when creating conditional processing that evaluates this token.

If the **Multiple Selections** attribute is set to *True*, the @Request token will contain a comma-separated list of selected values. For use with SQL statements, it may be desirable to surround each individual value in the list with single quotes. This can be done using the **SingleQuote** token prefix. For example, consider the token:

```
@Request.MySelections~
```

which contains the string `1,2,3`. When we apply the SingleQuote token prefix, like this:

```
@SingleQuote.Request.MySelections~
```

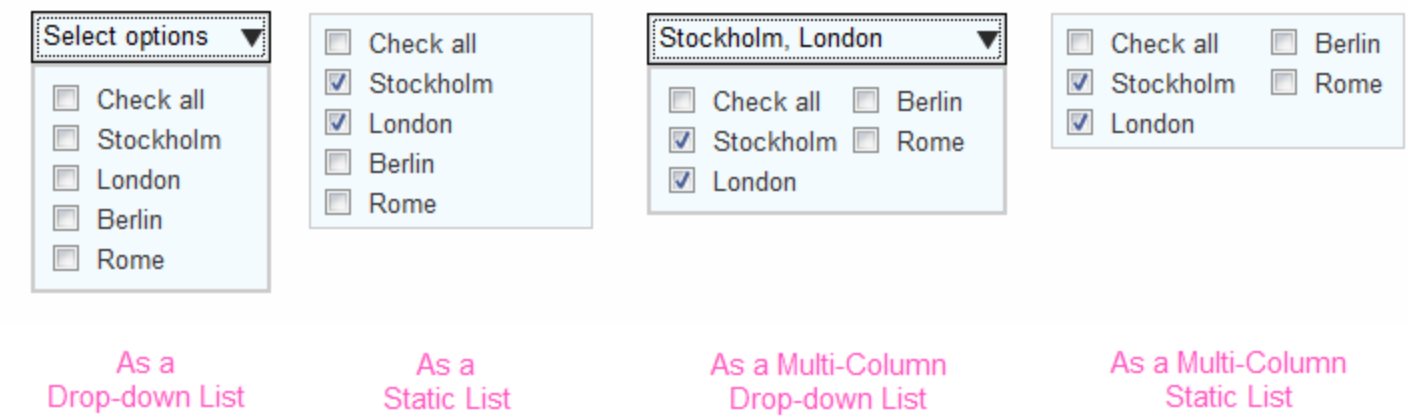
the token is modified so it now contains `'1','2','3'` which is acceptable in a SQL query.

More Information

For additional information, see the Element Reference entry for Input Select List.

Input Check Box List

The Input Check Box List element combines the features of the "Input Select List" on page 606 and "Input Check Box" on page 603 elements: it displays items in a list, each with its own check box.



This user input element operates using **collections of data** and requires a datalayer to provide that data.

Using the Input Check Box List

This element requires a datalayer to provide the items that will appear in the list. If the list of items is short and unchanging, the **DataLayer.Static** element can be used to "hard code" the items. If the list is longer, the **DataLayer.XML** element can be used to read the data from an XML file. If the list is dynamic, elements such as **DataLayer.SQL** can be used to retrieve the items from a datasource, possibly with grouping to provide unique values.

If the datasource provides one, the value passed to the next report or process task can be *different* from the displayed text. The control automatically sizes itself to accommodate the *widest* display text, or you can specify a width.

The element has many of the attributes common to all Input elements and includes these additional attributes:

Attribute	Description
Caption Column	(Required) Specifies the name of the datalayer column that contains the text to be displayed in the selection list.
Value Column	(Required) Specifies the name of the datalayer column that contains the value to be passed to the next report or process task *if* the check box is checked.
Check All Caption	Specifies the text for the "Check all" check box. Enter a value of "none" to hide it altogether.
Check Box List Drop-down	Specifies whether the list is rendered as a drop-down list. Default: *False* (i.e. static list). If the check box List Dropdown attribute is set to *True*, then the drop-down element is rendered. When the drop-down is rendered, you can utilize the search feature via input box. The search will be performed every time the search content changes. Basic conditions are: ignoring case, not full word matching, and the search content will not be treated as regular.
Column Count	Specifies how many columns of items should be displayed. Default: *1*
Default Value	Specifies the value of the item to be pre-checked in the control. Enter the actual the value (from the "Value" column), not the displayed text, in order to set this. Can be used with @Request tokens passed from another report or a process task, and other tokens. To specify multiple items to be pre-checked, add a child Default Values element.

Attribute	Description
Dropdown Button Class	Specifies the style sheet class used to control the drop-down header and button appearance, such as font color. The actual button icon image can be overridden by creating a class named `.rd-check boxlist-icon` and setting its background-image selector to the desired image file.
Dropdown None Selected Caption	Specifies the text that appears in the drop-down header when there are no items selected. Default: *Select options*
Dropdown Selected Caption	As items are checked, their Caption text is displayed in a comma-separated list in the drop-down header. When there are too many items to be shown, the text changes to show the number of items selected. This attribute specifies the text displayed at that time. The "#" character is the placeholder for the number of selected items. Default: *# selected*
Enable Search	Specifies whether the search box displays. Default: *True*. Setting the attribute to *False* turns off the search box. You can also enter any value to specify the threshold for the search box. If the value is a valid integer, the search box will be visible if the item count is equal to or greater than the value. Any invalid input will be treated as *True*, the search box displays.
Height Height Scale	Specifies the height of the element, in pixels or as a percent of its containing element. If left blank, a default value is automatically applied.
List Captions Element ID	Specifies the element ID of an Input Hidden element, which will contain the Caption Column values for the items selected when the definition is submitted. The values will be available in a @Request token using the

Attribute	Description
	same ID.
Multiple Selections	Specifies if multiple list check boxes can be checked at the same time. Default: *True*
Tooltip	Specifies text to be displayed as a tooltip when the mouse is hovered over the control and, if no Tooltip Column is specified, over the list of options.
Tooltip Column	Specifies the datalayer column that contains text to be shown as a tooltip when the mouse is hovered over *each item* in the list of options. Can be used with or without the Tooltip attribute.
Width Width Scale	Specifies the width of the element , in pixels or as a percent of its containing element. If left blank, a default value is automatically applied.

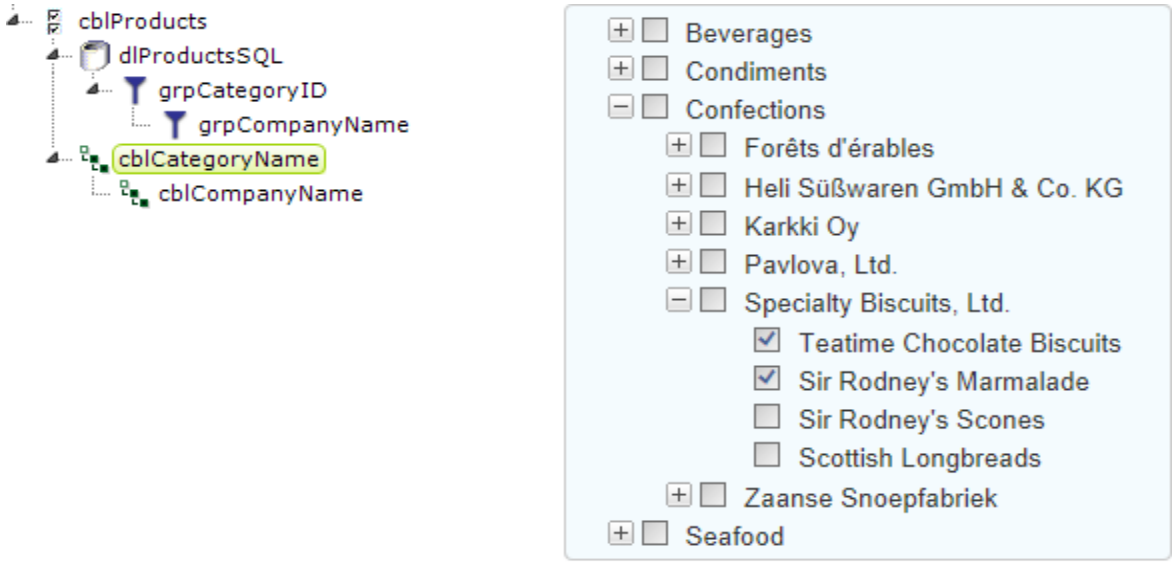
This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frame-works or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

If the Input Check Box List is placed under a **Data Table** that produces multiple rows, the Input Check Box List's datalayer only runs once and thus all check box lists will have the same rows. In such an arrangement, you cannot reference @Data tokens from the parent Data Table in the Input Check Box List's datalayer.

Displaying Hierarchical List Items

The **Check Box List Branch** element allows the items inside Input Check Box List to be displayed in a hierarchical arrangement. The example below shows two levels of items below the major item:



When one or more Check Box List Branch elements are used, data for the Input Check Box List must come from a *hierarchical* datalayer. This can be created by using a datalayer that has a **Group Filter** element with its **Hierarchical** attribute set to *True*.

The data should have the same number of hierarchical levels as there are Check Box List Branch elements or, put more simply, the number of Group Filters under the datalayer should equal the number of Check Box List Branch elements.

As with the Input Check Box List element, Check Box List Branch provides optional custom expanded/collapsed image attributes, and Class and initial state attributes

Getting Its Data

The value (from the "Value column") associated with a checked item (from the "Caption column") is available in the next Report or Process task with an @Request token. For example, if the element's **ID** is set to *inpCity* then the token `@Request.inpCity~` will equal that value.

If nothing is checked in the list and the page is submitted, then this @Request token will equal an empty string ("").

If the **Multiple Selections** attribute is not set to *False*, the @Request token will contain a **comma-separated list** of the checked values. For use with SQL statements, it may be desirable to surround each individual value in the list with single quotes. This can be done using the SingleQuote token modifier. Thus

> `@Request.MySelections~` which hasa value of `London,Berlin,Rome` becomes
> `@SingleQuote.Request.MySelections~` which has a value of `'London','Berlin','Rome'`

If you want to access the caption(s) of the item(s) selected, instead of or in addition to their values, set the **List Captions Element ID** attribute to the ID of an Input Hidden element. The caption(s) will be placed in this element when the report is submitted and will be available using a @Request token with the same ID. Gettings multiple captions is dependent on setting the Multiple Selections attribute = *True*.

If you're using Input Check Box List to show a hierarchical list of items, its @Request token will contain the value for the lowest checked item in a branch, but the values of the other checked items above it in the branch are not available in the same way.

Default values for an Input Check Box List element with multiple selections can be provided using a child **Default Values** element. The default values are retrieved by a child datalayer beneath the Default Values element. If a Default Values element is present, then the Input Check Box List's default value attribute is ignored.

More Information

For additional information, see the Element Reference entry for Input Check Box List.

Input Combo List

The Input Combo List element combines features of the Input Text, Input Select List, and AutoComplete elements.



This user Input element operates using **collections of data** and requires a datalayer to provide that data.

Using It

Users can select values from a drop-down list by clicking the down arrow to show the list, then clicking an item in the list. Or they can start typing a value into the text box and the list will appear, dynamically filtering itself as more characters are typed.

If its **Multiple Selections** attribute is set to *True*, then users can select or enter multiple items, building a comma-separated list of values. As values are added to this list, they're remove from the drop-down list. These are the values that will be passed to the next Report or Process task.

Like other "list" elements, this one requires a datalayer to provide the list items and its **Combo List Column** attribute specifies the datalayer column that contains the values to be displayed in the drop-down list.

When blank values are part of the data set, they will be included in the drop-down list.

If you don't specify a width (Input Size attribute) the control will automatically size itself to accommodate the widest value, plus the down arrow icon.

The element has many of the attributes common to all Input elements and also these additional attributes:

Attribute	Description
Combo List Column	(Required) Specifies the name of the datalayer column that contains the text to be displayed in the drop-down list.
Multiple Selections	Specifies whether users can select or enter multiple items from the drop-down list. If set to *True*, users can select or enter multiple items, building a comma-separated list of values. As values are added to this list, they're remove from the drop-down list, and the values list will be passed to the next Report or Process task.
Tooltip Column	Specifies the name of an column in the datalayer that contains explanatory text to be displayed as a tooltip unique to each list item.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frame-works or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

Like an Input Text element, the value entered into or selected in this element will be available to the next Report or Process task as an @Request token. For example, if the element's ID is set to *inpCountry*, then its data will be available as the token

`@Request.inpCountry~`. If no values is entered into the text box and the page is submitted, then its @Request token will equal an empty string ("").

If the element's Multiple Selections attribute is set to *True*, the @Request token will contain a **comma-separated list** of the selected or entered values. For use with SQL statements, it may be desirable to surround each individual value in this list with single quotes. This can be done using the SingleQuote token modifier. Thus

> `@Request.inpCountry~` which has a value of `Australia,Sweden` becomes
> `@SingleQuote.Request.inpCountry~` which has a value of `'Australia','Sweden'`

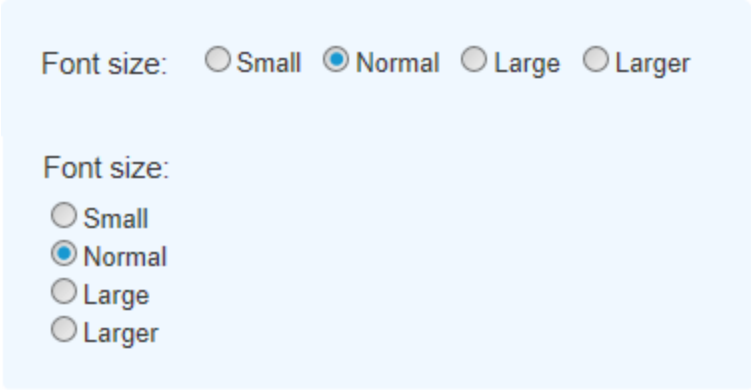
Default values for an Input Combo List element can be provided using its **Default Values** attribute, which can be a comma-separated list.

More Information

For additional information, see the Element Reference entry for Input Combo List.

Input Radio Buttons

The purpose of this element is to provide at least two "radio buttons", small circles with captions. They are "mutually exclusive": only one radio button in a group can be selected at a time. The buttons can be arranged horizontally or vertically.



Using It

This element requires a datalayer to provide the items that will appar as button options. One button will be displayed for each record in the datalayer. If list of options is short and unchanging, the **DataLayer.Static** element can be used to "hard code" the options. If the list is longer, the **DataLayer.XML** element can be used to read the data from an XML file. If the options are dynamic, elements such as **DataLayer.SQL** can be used to retrieve the items from a datasource, possibly with grouping to provide unique values.

If the datasource provides one, the value passed to the next Report or Process task can be different from the displayed text. The control automatically sizes itself to accommodate the displayed text.

The element has many of the attributes common to all Input elements and also these additional attributes:

Attribute	Description
Caption Column	(Required) Specifies the name of the datalayer column that contains the text to be displayed beside the radio buttons. This text is always displayed *to the right* of the button. In order to display text to the left, leave this attribute blank and precede the element with Label and Space elements.
Value Column	(Required) Specifies the name of the datalayer column that contains the value to be passed to the next report or process task. For example, if the "Larger" radio button is selected in the list, the value "4" might be passed to the next report. This allows the flexibility to display and pass different values and even different types of values, such as numbers.
Default Value	Specifies which radio button will be selected by default; enter the actual the value (from the "Value" column), not the displayed text, in order to set this. Can be used with @Request tokens passed from another report or a process task, or other tokens.
Radio Button Direction	Specifies the physical arrangement of the radio buttons: in a horizontal (*Across*) or vertical (*Down*) layout, as shown in the image above.
Tooltip Column	Specifies the name of an optional column in the datalayer that contains explanatory text to be displayed as a tooltip unique to each radio button.

This element can be used by itself or within an **Input Grid** element to make alignment with other Input elements easier.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

 This element *cannot* be used beneath a **Data Table** element, so you cannot display radio buttons inside table columns.

Getting Its Data

The value (from the "Value" column) associated with the selected radio button (from the "Caption" column) is available in the next Report or Process with an @Request token. For example, if the element's ID is set to *radFonts* then the token `@Request.radFonts~` will equal that value.

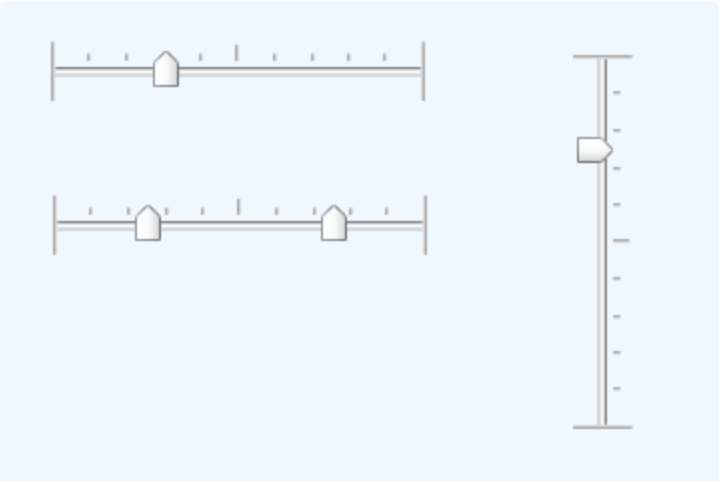
If there is no default selection and no radio button is selected and the page is submitted, then this @Request token *will not exist*. This is an important distinction to note when creating conditional processing that evaluates this token.

More Information

For additional information, see the Element Reference entry for Input Radio Buttons.

Input Slider

The purpose of this element is to provide one, or a pair of, sliding "thumbs" that can be used to adjust numeric value inputs. The thumbs are clicked and dragged to change the input value and the thumb scale orientation can be horizontal or vertical.



Using It

The slider can be oriented horizontally or vertically. It *does not* have a **Caption** attribute, so a caption for this element must be provided separately. It features several standard attributes; other attributes of interest include:

Attribute	Description
ID	(Required) Specifies the unique element ID associated with this control. This ID is used with an @Request token after the page is submitted to identify the data value for the first sliding thumb.

Attribute	Description
Background Image	Specifies the filename of an image for the background of the slider; if left blank, a default image is supplied. An image provided here is not stretched to fit; it must exactly fit the size of the slider; tick marks are automatically generated.
Decimal Points	Specifies the number of decimal places in the data values. The default is *0*, which causes the slider to return only integer values.
Default Value	Specifies the default value associated with the first sliding thumb, shown when the report page is first displayed.
Minimum Value Maximum Value	Specifies the range of numeric values represented by the slider scale. The default values are *0* and *100*, respectively.
Second Default Value	Specifies the default value associated with the second sliding thumb, if used, when the report page is first displayed.
Second Slider ID	Specifying a value here causes a second sliding thumb to be displayed; leave this blank to display just one sliding thumb. This ID is used with an @Request token after the page is submitted to identify the data value for the second thumb.
Slider Length	Specifies the width or height of the slider, in pixels, depending on the Slider Orientation attribute. The value represents the distance that the thumbs can move. Default: *200* pixels.

Attribute	Description
Slider Orientation	Specifies the orientation of the slider: horizontal or vertical. Default: *Horizontal*.
Slider Thumb Offset Left Slider Thumb Offset Top	Allows fine adjustment, in pixels, of the thumb image positioning, so that it aligns well with the slider.
Thumb Image	Specifies the filename of an image for the thumbs; if left blank, a default (shown above) is used.
Thumb Separation Value	When there are two sliding thumbs, specifies the minimum separation allowed between them. The value is based on the values of the Min Value and Max Value attributes, not in pixels.
Tick Count	Specifies the number of "detents", or stopping points, at which the thumbs will stop when dragged. Using this value, the detents are evenly distributed across the length of the slider; tick marks are automatically created.

This element *cannot* be used within an **Input Grid** element. If multiple Input Slider elements are used on a page, ensure that each thumb has a unique ID.

This element causes two **DHTML events** to occur. These can be used with Event Handler elements to provide user feedback, to refresh the current page, or to launch another report. The *onDrag* event is fired as a thumb is dragged, and the *onChange* event is fired when the dragging operation finishes.

This element incorporates a JavaScript function that can be called from script to set the slider value. The function syntax is:

```
rdInputSliderSet('sliderID', value)
```

This element *will not* appear in exported reports.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

The numeric value selected using this element will be available in the next Report or Process task by using an @Request token. For example, if the element's **ID** attribute is set to "Thumb1" and its **Second Slider ID** attribute is set to "Thumb2", then the selected values will be available in the next page or process task as the tokens `@Request.Thumb1~` and `@Request.Thumb2~`.

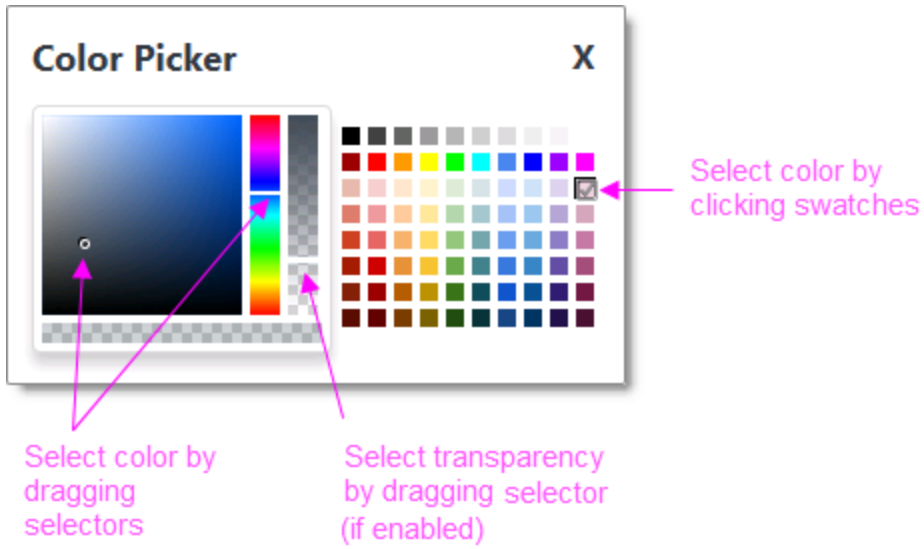
More Information

For additional information, see the Element Reference entry for Input Slider.

Input Color Picker

The Input Color Picker was significantly improved in this release. Users of earlier Info v12 releases will see a subset of the features described here.

This element provides a visual interface for color selection. The current color selection appears as an icon next to a palette icon.



Clicking either icon displays the Color Picker pop-up panel, as shown above, from which new selections can be made.

Using It

This element shares many of the standard Input element attributes and includes these special attributes:

Attribute	Description
Allow Transparency	Specifies whether the Color Picker pop-up panel will include a slider for selecting the transparency of the selected color. The default value is *False*.
Color Picker Caption	Specifies a custom caption for the Color Picker pop-up panel. The default value is *Color Picker*.
Default Value	Specifies the selected default color, by name, decimal RGB or RGBA value, or hex RGB value. Prefix hex values with the pound sign, like *#112233*.
Popup Modal	Specifies whether or not the Color Picker pop-up panel will be "system modal". Modal pop-ups disable the rest of the page until the panel is closed and the disabled area behind the panel get shaded in transparent gray.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

The value associated with the color selected from the palette is available in the next Report or Process task with an @Request token. For example, if the element's ID is set to *inpColors* then the token `@Request.inpColors~` will equal that value. This value will always be in hex RGB format: *#112233*.

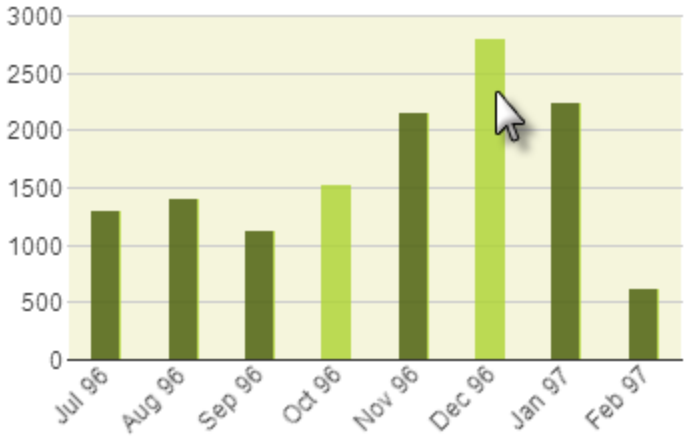
More Information

For additional information, see the Element Reference entry for Input Color Picker.

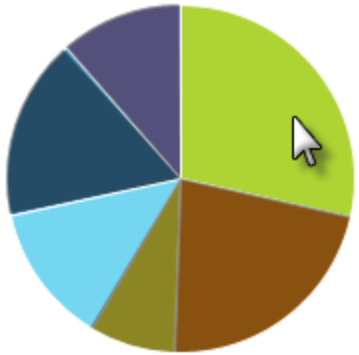
Input Chart.List

A number of Classic Charts have been deprecated; they are still supported and will work, but their elements are no longer available in Studio. Use Chart Canvas Charts instead for all new development.

This element enables a **Classic Chart** to become an input control. Once a **static Bar** or **Pie** chart has been made the child of this element, users can select one or more bars, or one or more pie regions, by clicking on them. When the page is submitted, the Label values of the selected chart items are passed to the next report or process.



Bar chart with two bars selected



Pie chart with two wedges selected

As shown above, the selected bars or wedges are displayed in a distinctly different color from the other bars or wedges by default. If the data has been grouped so that it's displayed as groups of several bars each, the entire group will be selected when clicked, rather than individual bars within the group.

Using It

This element shares many of the standard Input element attributes and includes these special attributes:

Attribute	Description
Input Chart Value Column	Specifies the data column whose values which will be returned, such as the IDs for each selected bar in a Bar chart. The chart's Chart Label Column (aka: Label Data Column X-axis) is the default.
Selected Color	Specifies the selected bar or wedge's color, by name, decimal RGB value, or hex RGB value. Prefix hex values with the pound sign, such as *#112233*.
Selected Transparency	Specifies the transparency level of the selected bar or wedge. The lowest possible value of 0 indicates that the region is opaque, with no transparency. The other end of the scale, 15, indicates a completely transparent bar or wedge.
Unselected Color	Specifies the color of unselected bars or wedges, by name, decimal RGB value, or hex RGB value. Prefix hex values with the pound sign, such as *#112233*.
Unselected Transparency	Specifies the transparency level of the unselected bars or wedges. The lowest possible value of 0 indicates that the region is opaque, with no transparency. The other end of the scale, 15, indicates a completely transparent bar or wedge.

This element *cannot* be used within an **Input Grid** element. A child validation element, **Validation.Required**, is available to ensure that a selection is made by the user, if desired.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

The **Label** values of the selected bars or wedges are available in the next Report or Process task with an @Request token. For example, if the element's **ID** is set to *inpMyPie*then the token `@Request.inpMyPie~` will equal that value.

If no chart item is selected and the page is submitted, then this Request token will exist but it will be empty ("").

If multiple chart items are selected, the resulting Request token will contain a **comma-separated list** of selected values. When working use with SQL statements, it may be desirable to surround each individual value in the list with single quotes. This can be done using the SingleQuote token modifier. Thus

`@Request.inpCLPie~`which has a value of `1,2,3`becomes
`@SingleQuote.Request.inpCLPie~`which has a value of `'1','2','3'`

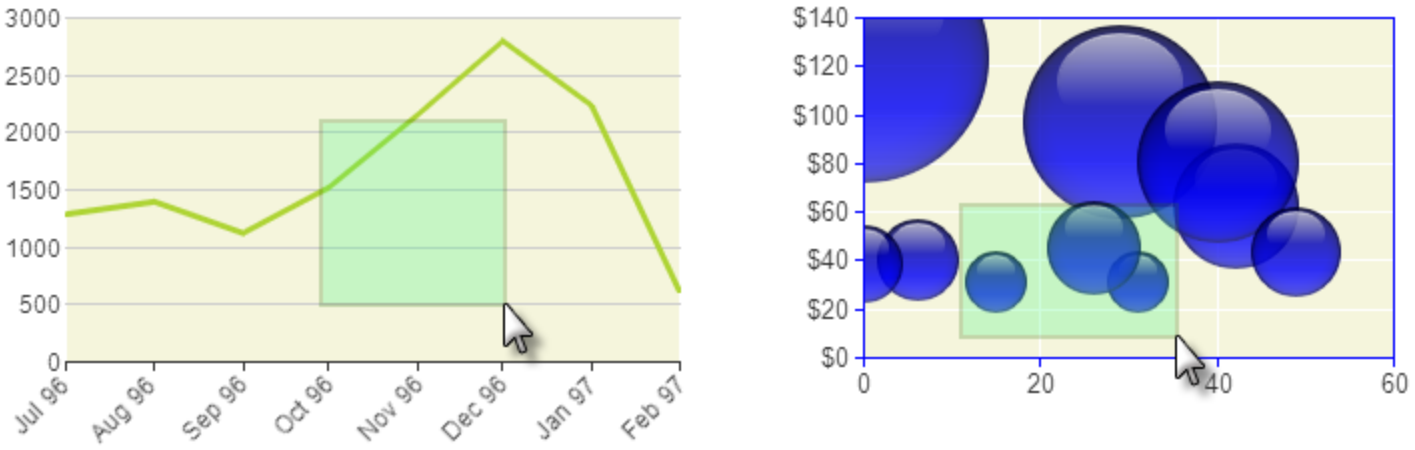
More Information

For additional information, see the Element Reference entry for Input Chart List.

Input Chart.Range

A number of Classic Charts have been deprecated; they are still supported and will work, but their elements are no longer available in Studio. Use Chart Canvas Charts instead for all new development.

This element enables a **Classic Chart** to become an input control. Once a static Line, Spline, or Scatter chart has been made the child of this element, users can select a region in the chart by drawing a selection box (by dragging the mouse) on it. When the page is submitted, the maximum and minimum Label and Data values of the selected chart region are passed to the next Report or Process task.



Mouse used to "draw" a rectangular selection area

The selected region is displayed in a distinctly different color from the rest of the chart by default.

Using It

The attributes of this element include:

Attribute	Description
ID	(Required) A unique element ID.
MaxXaxisID	Specifies the name of the Request token that will provide the upper X-axis (Label) value in the next report or process.
MaxYaxisID	Specifies the name of the Request token that will provide the upper Y-axis (Data) value in the next report or process.
MinXaxisID	Specifies the name of the Request token that will provide the lower X-axis (Label) value in the next report or process.
MinYaxisID	Specifies the name of the Request token that will provide the lower Y-axis (Data) value in the next report or process.
Region Border Color	Specifies the border color of the region drawn on the chart, by name, decimal RGB value, or hex RGB value. Prefix hex values with the pound sign, like *#112233*.
Region Color	Specifies the color of the region drawn on the chart, by name, decimal RGB value, or hex RGB value. Prefix hex values with the pound sign, like *#112233*.
Region Trans-	Specifies the transparency level of the region drawn on the chart. The lowest possible value of *0* indicates

Attribute	Description
parency	that the region is opaque, with no transparency. The other end of the scale, *15*, indicates a completely transparent bar or wedge.

This element *cannot* be used within an **Input Grid** element.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

Getting Its Data

The minimum and maximum **Label** and **Data** values of the region drawn on the chart are available in the next Report or Process task with @Request tokens. For example, if the element's MaxXaxisID attribute is set to *EndDate* then the token `@Request.EndDate~` will equal the maximum Label value.

If no selection region is drawn and the page is submitted, then the Request tokens will exist but they will be empty ("").

Special Event Elements

Two special, child event elements, **Area Drawn** and **Area Cleared**, are available for use with this element; both take Action elements as their children.

Area Drawn triggers its child Action element as soon as the user finishes drawing the area on the chart.

After an area has been drawn on the chart, if the user clicks somewhere on the chart again, the drawn area will be removed and **Area Cleared** will then trigger its child Action element.

More Information

For additional information, see the Element Reference entry for Input Chart Range.

Input Selection Point and Range

The **Input Selection Point** and **Input Selection Range** families of elements are available as children of some Chart Canvas chart Series elements. They provide the ability to interactively select portions of charts using the mouse and to initiate actions using the selected data. These input elements are discussed in *Input Selection for Chart Canvas Charts*.

Adding **HTML Attribute Params** to your User Input elements enables you to apply your application to work with other frameworks or libraries easier. Depending on the type of HTML Attribute Params you add, there will be different parameters available to use, or you can define your own parameters. If an attribute was set in both Element and HTML Attribute Params, the one set in the HTML Attribute Params will be ignored.

More advanced information relating to many of input elements can be found in "Working with User Input Elements" on the next page.

Working with User Input Elements

User Input elements allow users to interact with applications, most often to provide reporting criteria.

The following topics discuss a variety techniques for working with these elements:

- Working with Change Flags
- Aligning Input Elements
- Using AutoComplete
- Wrapping in a Fieldset Box
- Inclusion in Data Tables
- Populating Input Element Collections
- Setting Default Values
- Re-displaying Entered Values
- Validating Input
- Using Validation Elements
- Using Event Handlers
- Using Event Handlers with JavaScript

When Are Input Values Available?

Logi reports use the basic HTML FORM-submission mechanism for posting input control values. This means that a page *has to be submitted* before any of the values in its Input elements will be available for use in the next or reloaded page. Attempts to access @Data tokens for those values in the page *before* it's submitted will not work. A common technique is to display a page, let the user enter values and submit the page, and then re-display the same page with dynamic content displayed based on the input values. Another common technique is to display a page, let the user enter values and submit the page, run a Process task that does something with the entered values, and then re-display the original page. Some script-based techniques can be used to initiate

local actions on the input values, without submitting the page. For example, if text is entered in an input control, itcan be capitalized when the mouse leaves the control.

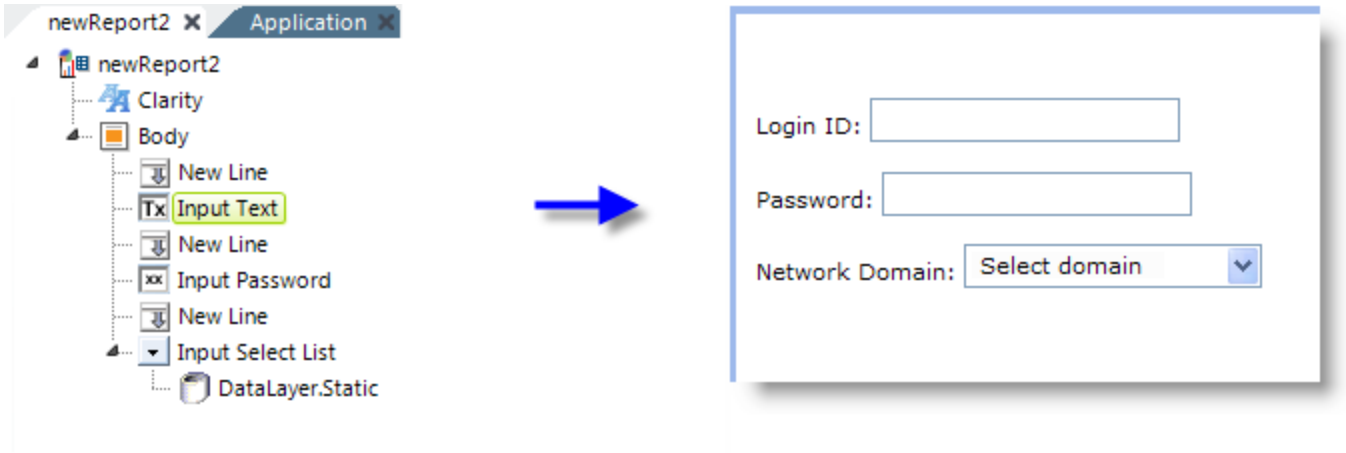
Working with Change Flags

You may find it useful to know, after a page with User Input elements has been submitted, whether or not any changes were made to their default values - did the user make any changes? This, for example, might let your code decide whether or not to take some action, such as updating a database record. Logi Info provides a "change flag" mechanism to let you make this determination. A "change flag" is simply a Request variable with a *True* or *False* value; an Input Hidden element is added to a Report definition to generate that value. Other User Input elements specify the Input Hidden element's ID in their **Change Flag Element ID** attributes. When the page is submitted, in the next Report definition or Process task, the request variable associated with the Input Hidden element will have a value of *True* if any changes to Input elements occurred. This change flag functionality relies on the DHTML "onChange" event supported by all browser. Prior to Logi Info v12.1, this event was used to the exclusion of other events, so you could not also use your own onChange event handler with elements implementing a Change Flag Element ID.

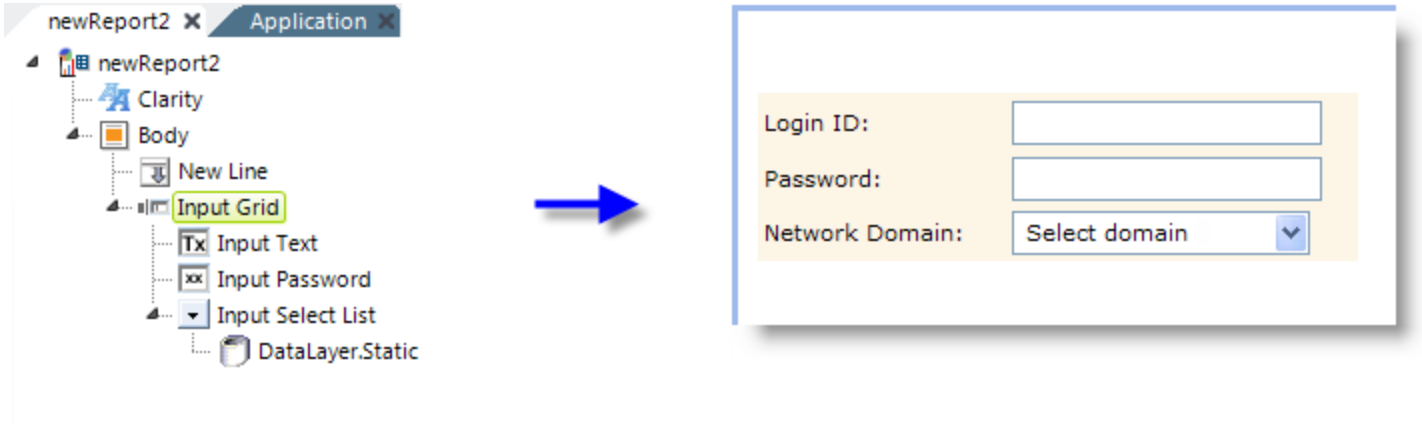
Support for multiple events is now available. When using a change flag, you may also add Event Handlers to User Input elements and they will be processed *after* the onChange events associated with setting the Input element change flags.

Aligning Input Elements

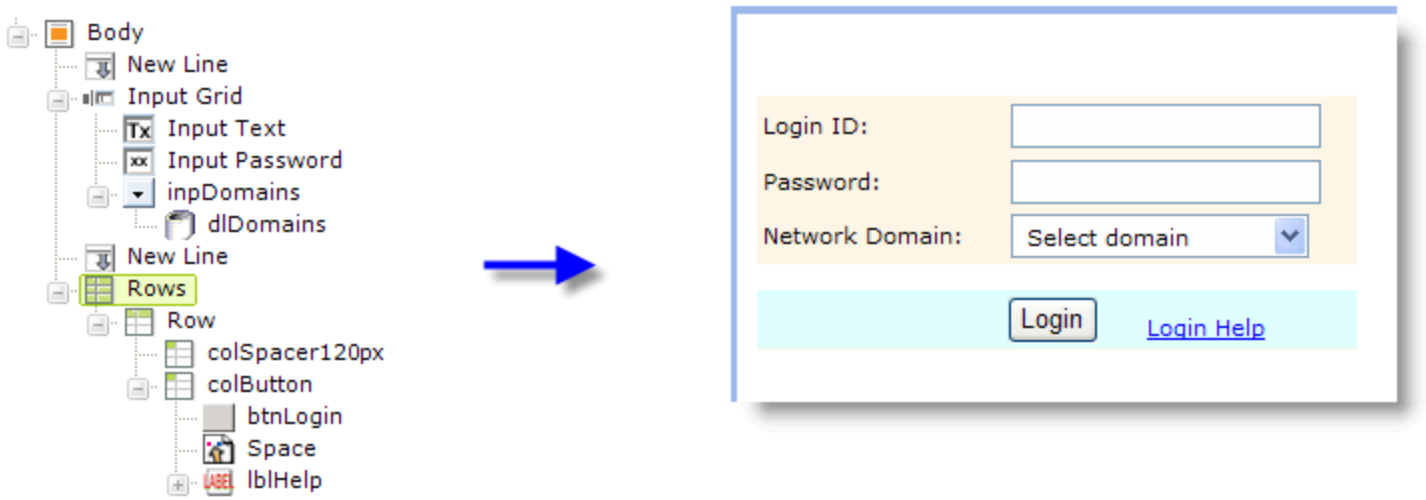
One challenge when working with Logi Studio is to get user input elements neatly aligned. The **Input Grid** element has been provided to assist you in doing this. The Input Grid is not an actual Input element but is instead a *container* for Input elements that lets you align them. It can be thought of as a two-column table with the first column containing the caption describing an Input element and the second column containing the Input element itself. Adding multiple rows allows a data input form to be created. The overall width of the table and the width of the caption column can be set using attributes.



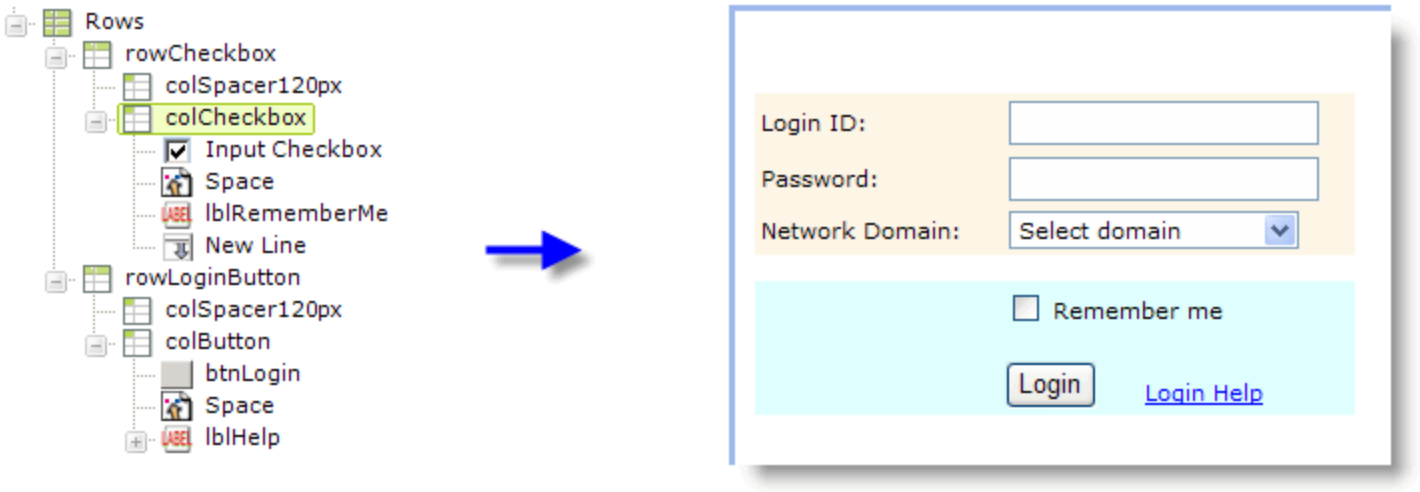
In the example shown above, user input elements are placed in a report definition *without* using the Input Grid element. As you can see, the **alignment** of the elements on the page is poor and the captions are not vertically centered on the controls.



In the second example, above, the user input elements have been made the children of an **Input Grid** element and the resulting, more precise, layout on the page is evident. For the purpose of this example, the Input Grid element was assigned a style class that colored its background, so you can see it more clearly. Attribute settings have made the entire grid 300 pixels wide and the caption space 120 pixels wide. In this definition, **New Line** elements between the input elements are no longer necessary.



There are some limitations, however. No other elements, such as Buttons or Labels, for example, can be placed within the Input Grid. These elements need to be carefully placed *outside* the grid, in their own HTML table using **Rows**, **Row**, and **Column Cell** elements, as shown above, if similar alignment is desired. Once again, for clarity in the example, this table has been assigned a style class to give it a colored background.



There are also times when you may wish to have the caption *follow* the control, rather than precede it. This can't be done within the Input Grid. Instead, as shown above, another row has been added to the table below the Input Grid and an **Input check box** element placed in it. The Input check box's **Caption** attribute has been *left blank* and a separate **Label** element provides the "caption" following the check box. The Input Grid and HTML tables are useful elements for alignment. We've seen how you can combine them and there are a variety of combinations that can work. For example, if you create a one-row, two-column table first and then put an Input Grid into each column, you can even make a two column (or more) data entry form.

Alignment Using Style Classes

The user input elements typically have two attributes related to style sheet classes: one for the entire element and one for the caption. This allows, for instance in an Input Select List, the font for the caption *and* the list items to be set to a certain font and size and the caption only to be set to bold.



It also allows other style selectors, such as **padding**, to be applied to the caption to align it correctly. For example, in the horizontal arrangement of radio buttons shown above, `padding-right: 3px`can be applied to the **Caption** to increase the space between each caption and the next button.

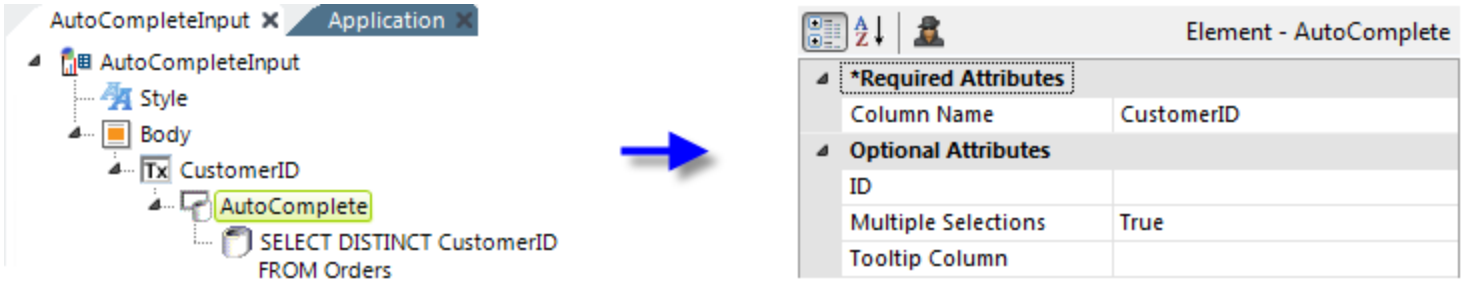


The default alignment for the caption of a Input Text Area element is *centered vertically*, as shown above left. This may not look very good in a stack of input elements, so the `vertical-align: top;`style selector can be applied to position the caption at the top of the input box. These are just a few examples of the many things that can be done using style classes. Be sure to distinguish

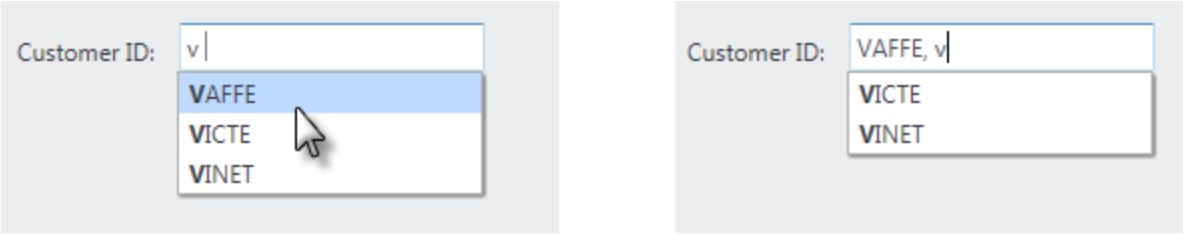
between situations where style classes need to be applied to the *elements* themselves and to the *objects* that contain the elements, such as an Input Grid or HTML Table.

Using AutoComplete

The **AutoComplete** element can be used to provide users with data-driven assistance when using Input Text and Input Email elements.

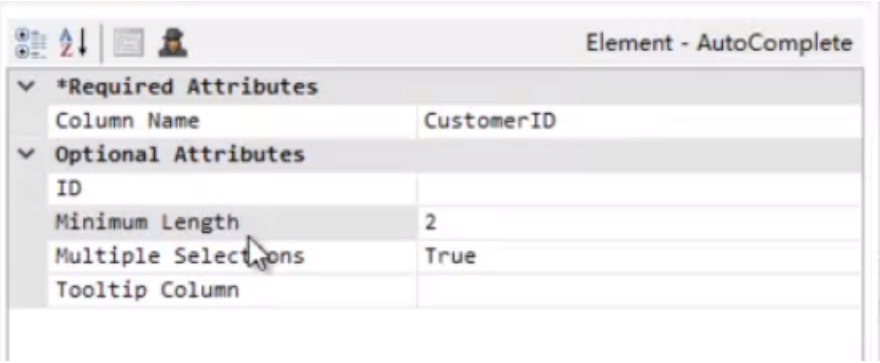


As shown above, the AutoComplete element is added as a child of the Input element and has its own child datalayer; all datalayer types are supported.



When the user begins typing into the Input element, a drop-down list of values, shown above, will appear. The list will consist of values from the datalayer column identified in the **Column Name** attribute. Items in the list will be progressively filtered with each letter typed, and can be selected at any time. Users can configure the number of characters that they want to generate the drop-down list of values by entering a value in the **Minimum Length** attribute, shown below. In this example, a search would not

return any values until 2 characters are entered into the Auto-Column box. This feature helps to increase performance and make searches more efficient by returning a limited set of search results.



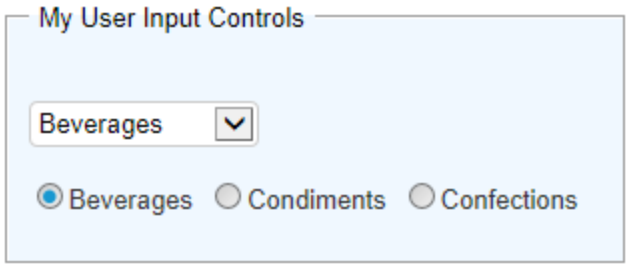
When the AutoComplete element's **Multiple Selections** attributes is set to *True*, the user may enter or select multiple text items, which will appear in the Input element separated by commas. Items that have already been added to the Input will be automatically filtered out of the remaining choices. The default separator character for multiple selections is a comma. You may change this to another character by manually editing the element's XML source code, in Logi Studio's Source tab, to include an **InputValueDelimited** attribute and value. For example, this will set the delimiter to a semi-colon:

```
<AutoComplete InputValueDelimiter=";"/>
```

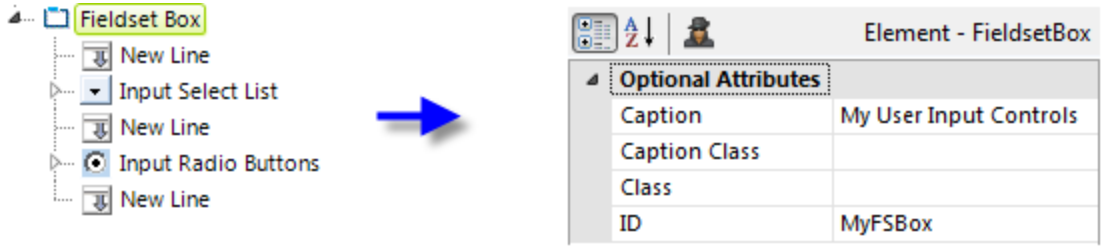
The resulting @Request variable created when the page is submitted will contain an individual value, or if multiple items have been entered/selected, a comma-separated list of values. Trailing commas will be removed automatically. For more information on parsing comma-separated lists for SQL queries, see "Input Select List" on page 606. The AutoComplete element includes a **Tooltip Column** attribute. Enter the name of a data column here to display a unique tooltip value for each data item.

Wrapping in a Fieldset Box

The **Fieldset Box** element provides a visual container for other elements, including User Input elements.



The example above shows a Fieldset Box, with a caption of "My User Input Controls" surrounding Input Select List and Input Radio Buttons elements. The box has been styled to have a pale blue background.



The elements used to produce the previous example are shown above, and this style sheet code:

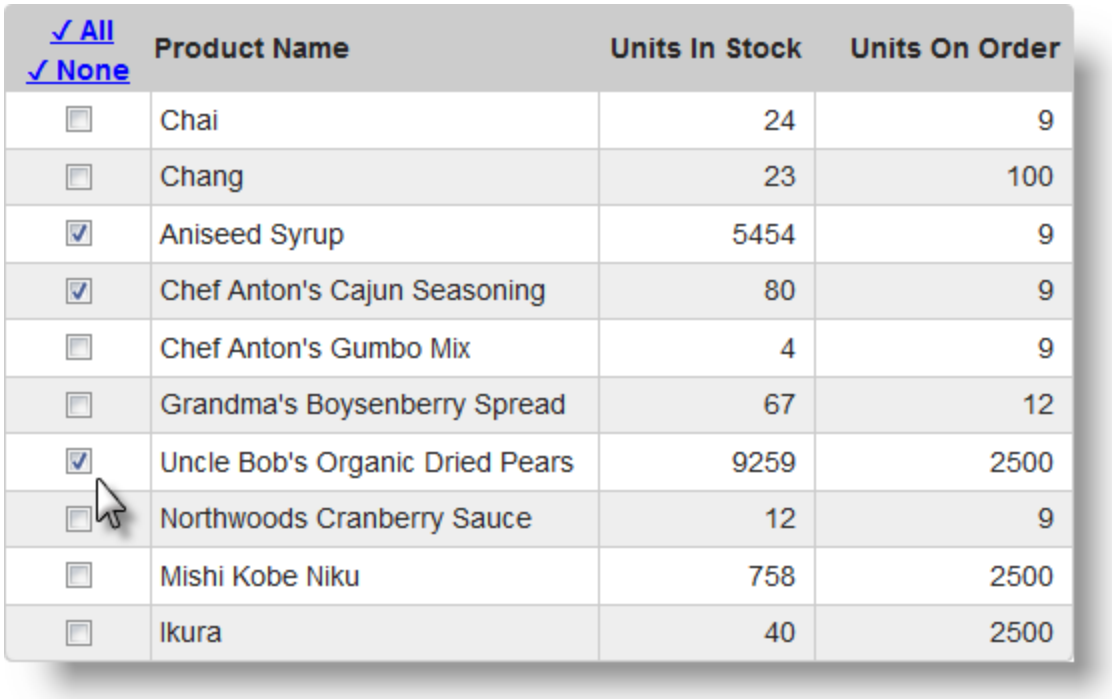
```
#myFSBox {
background-color: AliceBlue;
}
```

```
#myFSBox legend {
padding: 0px 5px 0px 5px;
}
```

was used to style it. The second class sets the space before and after the box caption text.

Inclusion in Data Tables

Some applications display data in a Data Table and include **user input** elements in each row of the table. A common example is a "Delete" check box for every record.

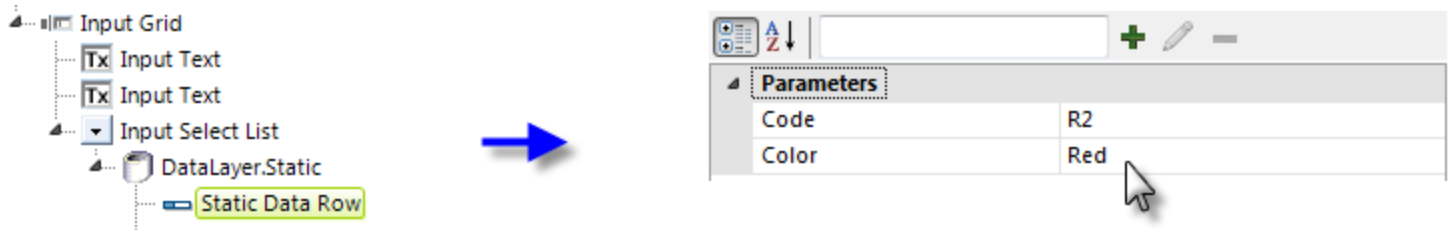


✓ All ✓ None	Product Name	Units In Stock	Units On Order
☐	Chai	24	9
☐	Chang	23	100
☑	Aniseed Syrup	5454	9
☑	Chef Anton's Cajun Seasoning	80	9
☐	Chef Anton's Gumbo Mix	4	9
☐	Grandma's Boysenberry Spread	67	12
☑	Uncle Bob's Organic Dried Pears	9259	2500
☐	Northwoods Cranberry Sauce	12	9
☐	Mishi Kobe Niku	758	2500
☐	Ikura	40	2500

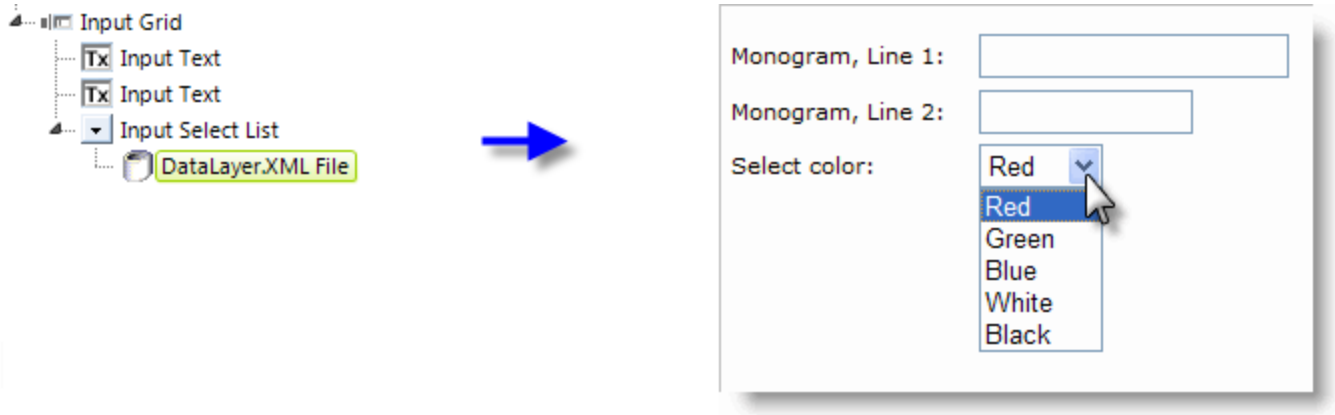
This can be done in Logi applications, but generally requires use of a Process Task to process the input values. DevNet offers a sample application that shows you the coding techniques for doing this. In addition, this technique is *not recommended* when hierarchical data is being used. The complex data relationships involved make it impossible to uniquely identify the associated user input elements. This prohibition applies to obvious hierarchical representations, such as the **Data Table**, and also to elements like the **Data Tree**, which use hierarchical grouping internally to create their parent-child data representations.

Populating Input Element Collections

Input Select List and **Input Radio Buttons** elements differ from the other UI elements in that they require a *datalayer* to provide their selection items. The use of a datalayer allows some interesting flexibility in presenting the data. Here are several examples of this in action:



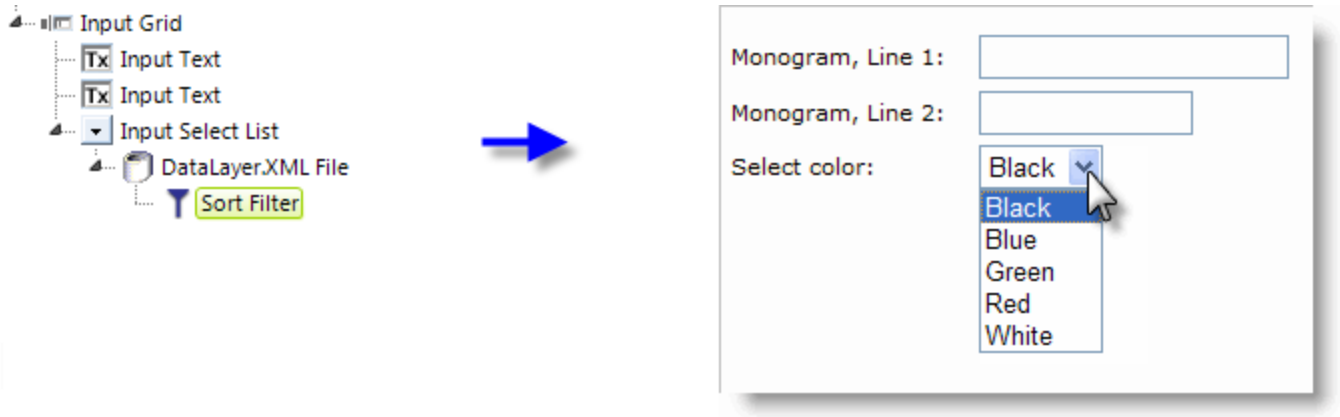
If the selection items are never going to change, then they can be "hard coded" into the definition using **DataLayer.Static** and child **Static Data Row** elements, as shown above.



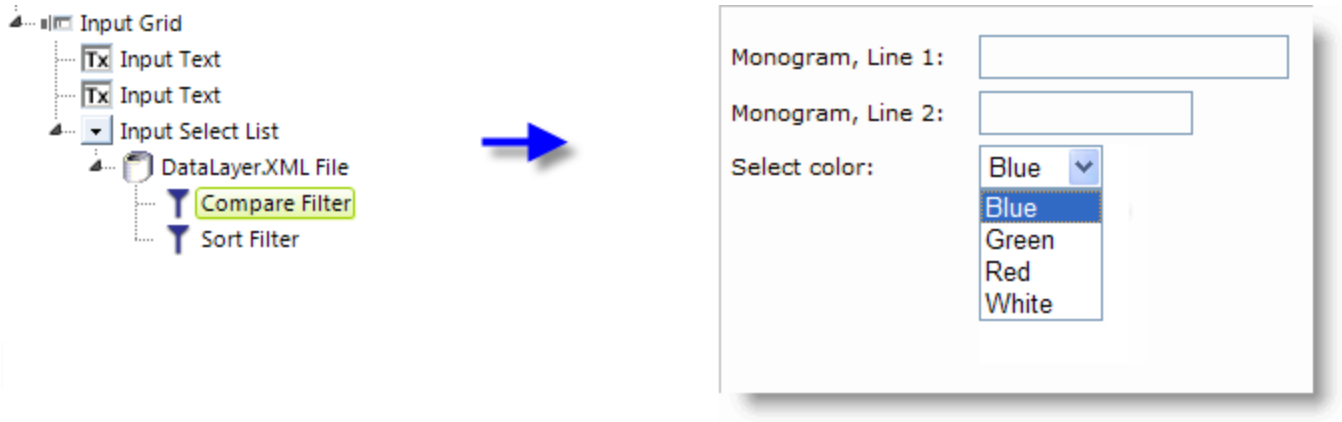
It may make sense to place the selection items in an **XML file**, which has the advantage of being able to be edited outside of your Logi application. In the example shown above, a **DataLayer.XML File** element is used to provide the selection list items,

```
<MyColors>
<Option Color="Red"/>
<Option Color="Green"/>
<Option Color="Blue"/>
<Option Color="White"/>
 <Option Color="Black"/>
 </MyColors>
```

and an example of the supporting XML data file is shown above.



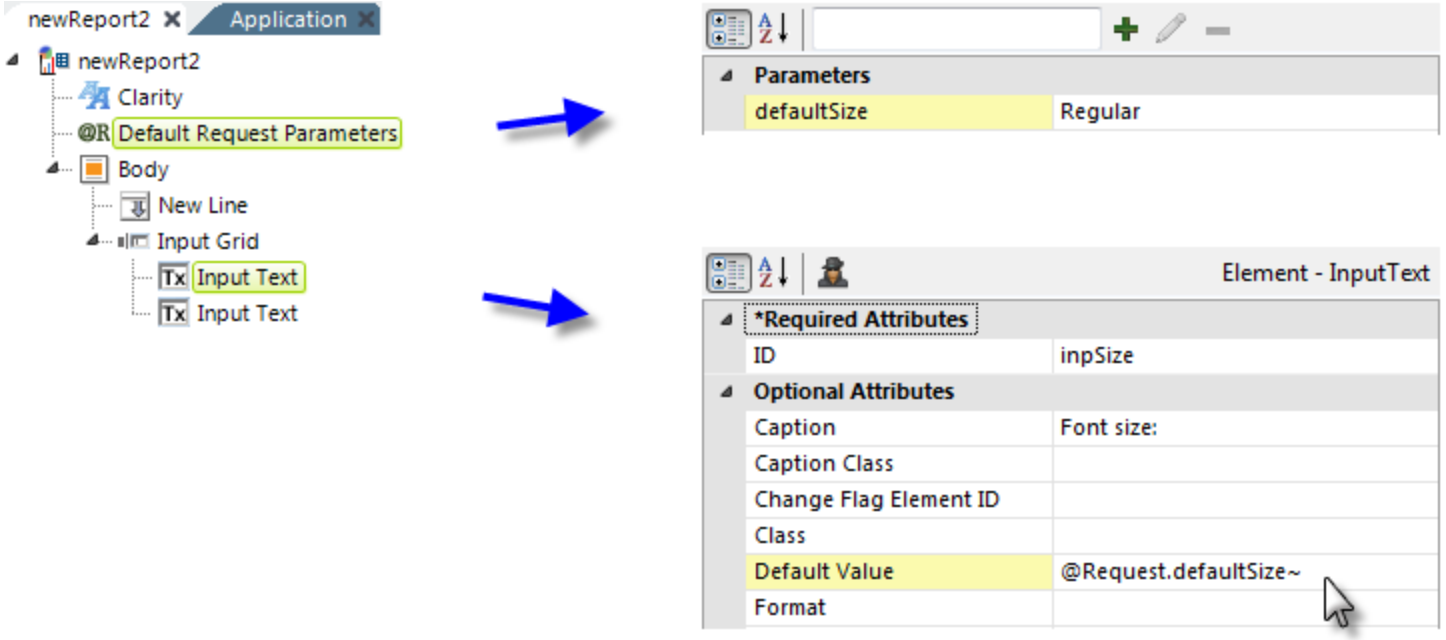
In the next example, above, a **Sort Filter** has been added to the datalayer, causing the list items to be sorted alphabetically.


Input Grid
Input Text
Input Text
Input Select List
DataLayer.XML File
Compare Filter
Sort Filter
Monogram, Line 1:
Monogram, Line 2:
Select color:
Blue
Blue
Green
Red
White

Setting Default Values

All of the user input elements, except Input File Upload, have a **Default Value** attribute, which lets you set the value that will be shown or selected when the page is first displayed. Making the desired value appear or be initially selected can be as simple as entering that value into the attribute - "hard coding" what will appear each time the page appears. Note that in the case of the **Input check box**, **Input Select List** and **Input Radio Buttons** elements, this value must be the one passed to the next page (from the Value column) not the displayed text. Tokens can also be used in the **Default Value** attribute. In a situation where the user enters data then submits the page, for example to a Process task, and then task calls the data input page again, it's common to want to *redisplay* the values the user entered. To do this, use @Request tokens in the **Default Value** attributes of your Input elements and ensure that the calling Process task sends the values as Request variables.

 This *cannot* be done by just setting the **Request Forwarding** attribute for a **Target.Report** element to *True*; you must use **Link Params** instead. If you've used @Request tokens as described above, how can you get some default values to appear the very *first time* the page is displayed?



As shown above, the **Default Request Parameters** element can provide Request variables, for use as your *default values*, when none are actually passed in the query string. The parameter and its value are created as shown above. Then the **Default Value** attribute of the first **Input Text** element is set to `@Request.defaultSize~`. This ensures that a value ("Regular") will appear in the Input element when the page is first loaded. If the query string that calls this page includes a Request variable of the same name, the Default Request Parameters value is ignored. There's another option for populating the default values in an **Input Select List** element. The **Default Values** element, added as a child of the Input Select List, allows you to provide multiple default values when the Input Select List's **Multiple Selections** attribute is set to *True*. The default values are retrieved by a child datalayer beneath the Default Values element. If a Default Values element is present, then the Input Select List's Default Value attribute is ignored.

Re-displaying Entered Values

After a user has entered values in Input elements, a report refresh or additional processing of some kind usually takes place. When the report is redisplayed, it's often desirable to redisplay the last values the user entered in the Input elements. There are three ways to do this: setting the default value attribute, saving the value in a cookie, and saving the value in local storage.

Setting the Default Value Attribute

Most Input elements have a **Default Value** attribute and you can set its value to either the @Request token passed when the report was refreshed (e.g. `@Request.theInputElementID~`), or an @Request token passed as a Link Param from the calling report or process task.

Saving the Value in a Cookie

The following elements have a **Save In Cookie** attribute:

- Input Chart List
- Input check box
- Input check box List
- Input Color Picker
- Input Combo List
- Input Date
- Input Email
- Input Number
- Input Password
- Input Radio Buttons
- Input Select List

- Input Telephone
- Input Text
- Input Text Area
- Input Time

Setting this attribute to "True" causes the entered value to be saved in a cookie, named for the element ID, when the page is submitted. This cookie survives the user session, preserving the value. You can then cause the saved cookie value to be used again in the next session by setting the Input element's **Default Value** attribute to the proper cookie token, such as: `@Cookie.theInputElementID~`

Saving the Value in Local Storage

The elements listed above were also given a **Save In Local Storage** attribute. Setting this attribute to "True" causes the entered value to be stored in the browser's "local storage" (if the browser supports this HTML5 technology). Data stored this way is preserved between sessions and is *automaticallyrestored* as the Input element's default value when the page is redisplayed. The local storage size limit is approximately 5 MB total and all the values are stored as strings.

Save In Local Storage is ignored if there's a value in the Default Value attribute. *Do not use them at the same time.* When *should* you save values in local storage instead of in cookies? Local storage is the "cleaner", recommended method. However, you should only use it when you're reasonably sure your users will be using newer browser versions that support HTML5 and local storage (Internet Explorer 7, for example, does not). In addition, all cookie values are passed with *every* request from the browser to the web server, increasing network traffic, while local storage values aren't passed to the server at all. Finally, some developers may have a policy against working with cookies, making local storage a good choice. Data is *not* stored in an encrypted form in cookies or local storage. The data can easily be found and read or edited on a user's machine, so don't store any sensitive information using these techniques.

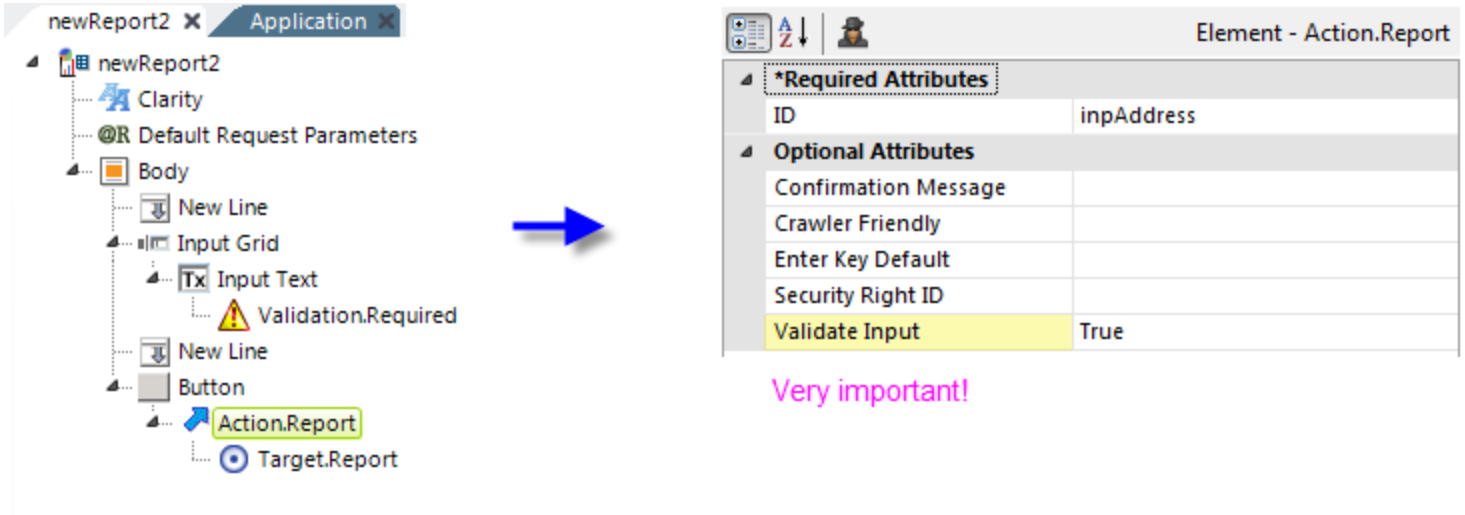
Validating Input

When it comes to validating data entered by users, we *highly recommend* it! It takes some additional development work up front, but can save so many heartaches (and time) later. Validating data is one of the most important tasks associated with processing user input. The browser environment presents some validation challenges and there are generally three opportunities to do validation. The first is related to events and occurs when the focus changes from one Input element to another element or when the Input element contents change. Validation in this case is accomplished using Event Handlers. The second opportunity occurs when the page is submitted. If the validation fails, then the submission is aborted. Logi Studio provides a variety of **Validation** elements for this purpose. This topic concerns itself with these first two validation opportunities but there is a third: it occurs in the *next* Report page or Process task, *after* submission has completed. That falls outside the scope of this topic but is highly encouraged, especially if you're updating a database. The **Action.Pre-Action Javascript** element, a child of many Action elements, can be used to validate input and control whether its parent Action element executes. For more information, see *Work with Scripting*.

Using Validation Elements

Validation elements are provided to allow you to validate the data users enter. All of them have some behaviors and attributes in common. For example, if validation *fails*, the page submission is aborted, an error message is displayed, and, optionally, the offending user input element can be highlighted by applying a different style class to it (coloring its background, for example). Multiple validation elements can also be used in combination under a single Input element.

You *must* give each Input element being validated a **unique ID**. This allows the validation process to correctly identify the element in case validation fails; failing to do this will result in validation being ignored. Some validation elements are not appropriate (nor available in Studio) for some user input elements. For example, it makes no sense to try to validate a numeric entry for the **Input Radio Buttons** element.



 A *very important step*, one that developers often overlook when working with Validation elements, is *enabling input validation*. As shown above, the **Action** element used to submit the page has a **Validate Input** attribute which must be set to *True* or else none of the Validation elements will do anything.

Validation.Date

The **Validation.Date** element allows the developer to validate several things about dates entered in an Input Date element. The Validation.Date element's attributes include those discussed above for the Validation.Required element and also adds **Range Start** and **Range End** attributes. Validation.Date will ensure that the date was entered in a format that matches the mask in the Input Date element's **Format** attribute. So, if the Input Date element's Format value is "MM/dd/yyyy" and the user enters "12/25/15", an error message will be displayed because a 4-digit year was not entered. Validation.Date can also ensure that the date entered falls within the date range specified by its **Range Start** and **Range End** attributes. To be validated, the date entered must be *on* or *after* the Range Start date, and be *on* or *before* the Range End date. For this validation to work correctly, the format of the Range Start and Range End values entered *must match* the format described in the Input Date element's Format attribute. If you use @Date tokens for the Range attributes, be aware that their default format, "yyyy-M-d", is not very user-friendly and might not match a more normal format you establish for Input Date. To reconcile this, in your _Settings definition, add a **Globalization** element and set its **Default Input Date Reformat** attribute to the same user-friendly format as your Input Date element. This will change the format of the @Date tokens, they'll match the Input Date element's Format attribute, and the validation will work correctly.

Validation.Email

The **Validation.Email** element allows the developer to validate the value entered in an Input Email element. Validation.Email ensures that a properly formatted email address value has been entered. The address is validated for the "@" symbol and a

domain name. Certain special characters forbidden in email addresses are not allowed. The validation elements provide a quick, easy method of implementing simple validation and can catch many data entry errors.

Validation.JavaScript

The **Validation.JavaScript** element allows you to validate the value entered by using a custom JavaScript function.
To use this element, enter a call to a custom function (found in the report definition) in its **Javascript Function** attribute. Usually this call will include the ID of the Input element as argument. Here's an example of the complete attribute value:

```
validateMustBeX(document.getElementById('inpText'));
```

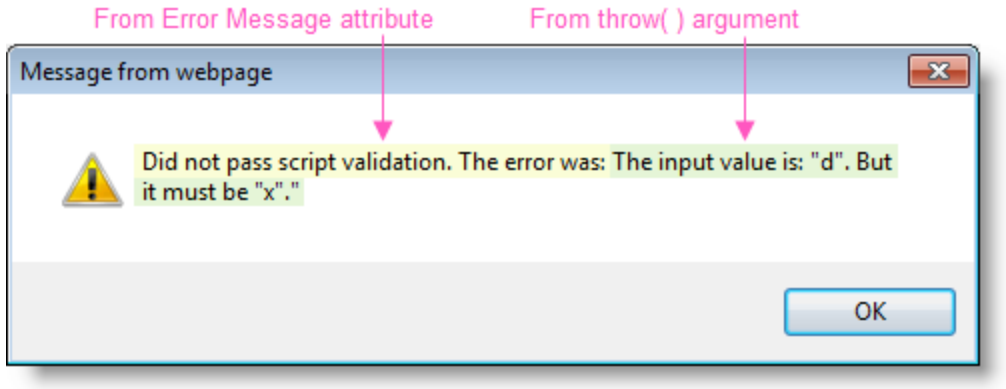
The function itself can be included in your report definition using any of the elements that run script in the browser, such as **Include Script** or **Include Script File**. When the page is submitted, the custom function will be run by the browser. If the function detects an error, it should call `throw(<errorMessage>)` to trigger display of an error message. Here's an example of a custom validation function that requires the letter "x" to be entered:

```
function validateMustBeX(eleInput){
 if (eleInput.value != 'x') {
 throw('The input value is: "' + eleInput.value + '". But it must be "x".')
}
}
```

If the validation fails, the text in the Validation element's **Error Message** attribute gets displayed in a message box. If the attribute's value contains the special string variable `{error}` it will be replaced with the custom function's throw( ) argument, providing a customizable error message. For example, the attribute value could be:

```
Did not pass script validation. The error was: {error}
```

When used with the custom function shown earlier, the resulting message box would look like this (assuming a the letter "d" was entered):



Here's a particularly good example of Validation.JavaScript in use: when using **Input Date** with a date range, ensuring that the Start date is not later than the End date. Assuming the input element is configured so that its ID is *inpDateStart* and its End Date Range ID attribute is set to *inpDateEnd*, here are the Validate.JavaScript element's attributes: Error Message: `Did not pass validation! {error}` JavaScript Function: `validateMe(document.getElementById('inpDateStart'), document.getElementById('inpDateEnd'));` As you can see, the function call above uses the element IDs to pass the input objects as parameters to the validating function.
And, in an Include Script element at the end of the definition, here's the code for the "validateMe" function:

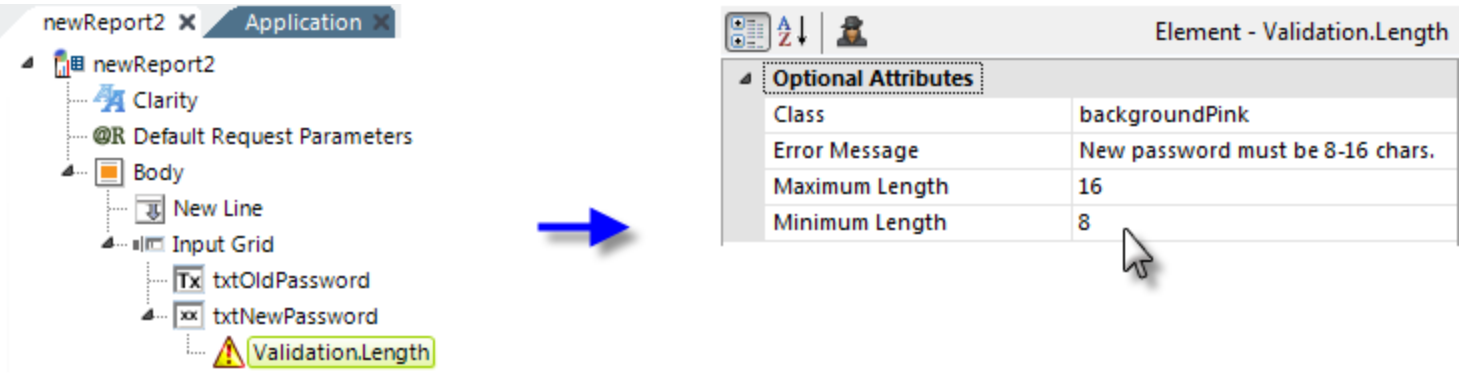
```
function validateMe(objStart, objEnd) {
var sDate = new Date(objStart.value);
var eDate = new Date(objEnd.value);

/* development only */
alert(sDate);
alert(eDate);
```

```
if (sDate > eDate) {
throw('The Start Date must be earlier than the End Date');
}
} JavaScript makes it easy to compare the Date objects directly, without having to parse them.
```

Validation.Length

The purpose of the **Validation.Length** element is to ensure that a *minimum* and/or *maximumnumber* of characters has been entered.

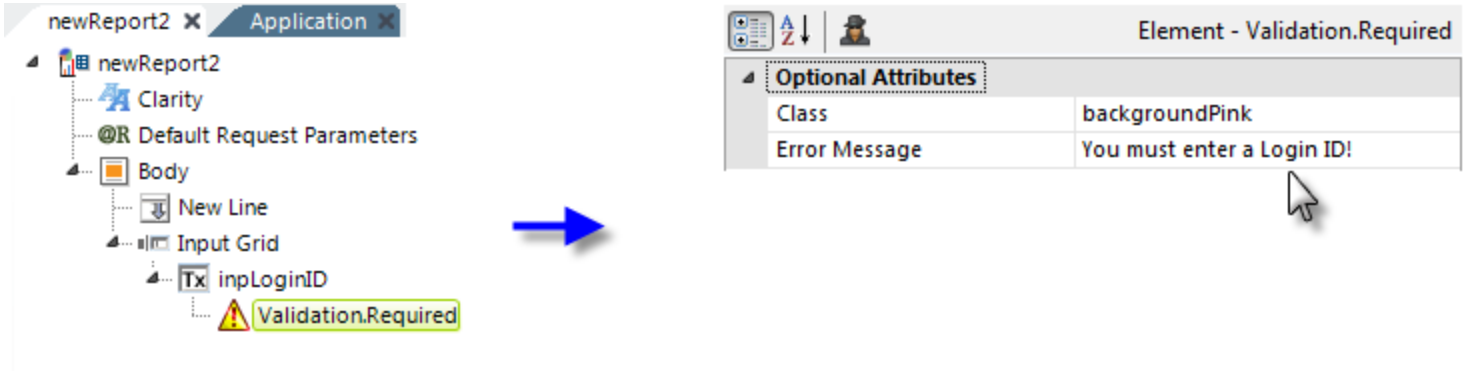


The Validation.Length element's attributes include those discussed above for the Validation.Required element and also adds **Maximum Length** and **Minimum Length** attributes, which are optional (though you *should* enter something for at least one of them or why bother with the element at all?)

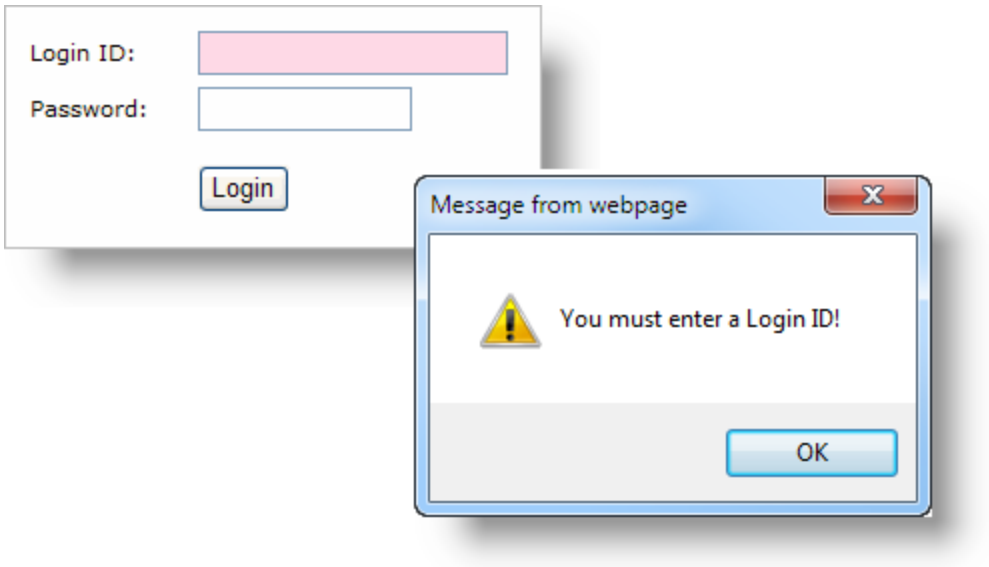
Validation.Numeric

The purpose of the **Validation.Numeric** element is to ensure that all of the characters entered in the text box are *numbers*. If any of the characters entered are letters or symbols, the validation fails, the error message is displayed, and the text box can have an optional style class applied to it. Page submission is aborted. The Validation.Numeric element's attributes are the same as those discussed above for the Validation.Required element.

Validation.Required



The purpose of the **Validation.Required** element, as shown above, is to ensure that the user *enterssomething* in the control. If the text box is left blank and the page is submitted,



a popup error message window will appear displaying the text set in the **Error Message** attribute. If the optional **Class** attribute is set, then the offending user input element will be set to that style class (allowing the developer to color its background). After the OK button in the popup error message window is clicked, submission of the page is *aborted*.

Validation.Time

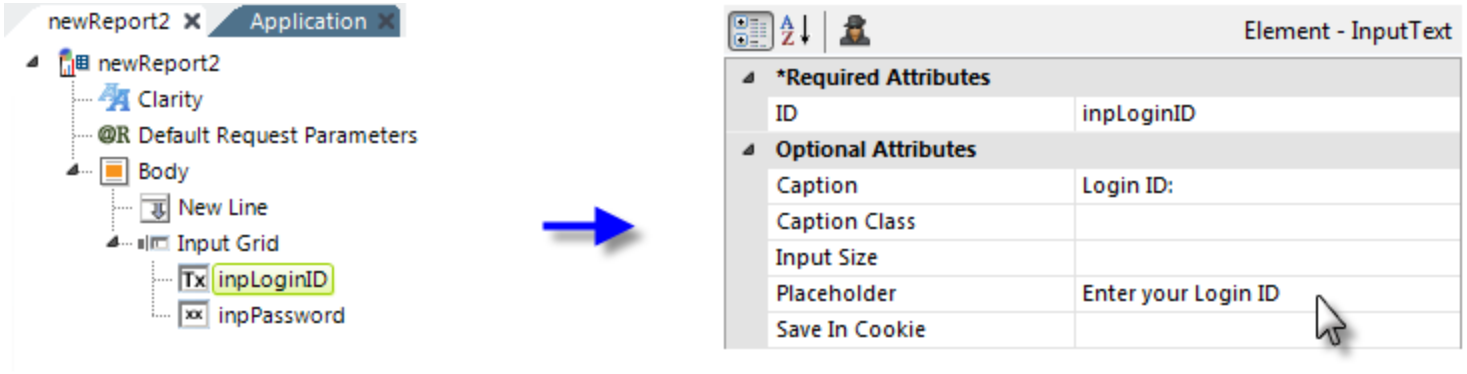
The **Validation.Time** element allows the developer to validate the time value entered in an Input Time element. The Validation.Time element's attributes include those discussed above for the Validation.Required element and also adds **Range Start Time** and **Range EndTime** attributes. Validation.Time ensures that a time value has been entered. The format of the time entered must conform to that in the **Format** attribute of the parent Input Time element. Validation.Time can also ensure that the time falls within a time range, specified with the Range Start Time and Range End Time attributes. To pass the validation, the time entered must be *on* or *after* the Range Start Time, and be *on* or *before* the Range End Time.

Using Event Handlers

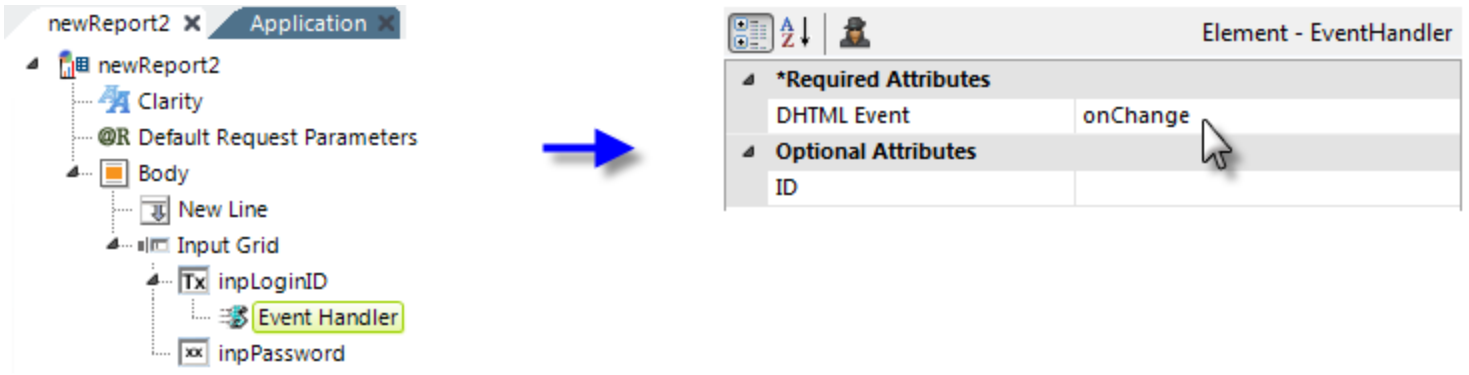
Another popular method of validating user input is through the use of *event handlers*. The Validation elements use this same technique but shield you from the underlying details. For better control and more validation variety, you can create your own Event Handlers, as shown in the following examples.

Event Handler Mechanics

The **Event Handler** element is used for this purpose and it responds to DHTML events. If you're unfamiliar with these events, information about them is available here from Microsoft.



The example above shows a typical arrangement and configuration of elements. The **Placeholder** text is displayed inside the Input control but disappears if text is entered (it also re-appears if the text entered is deleted).



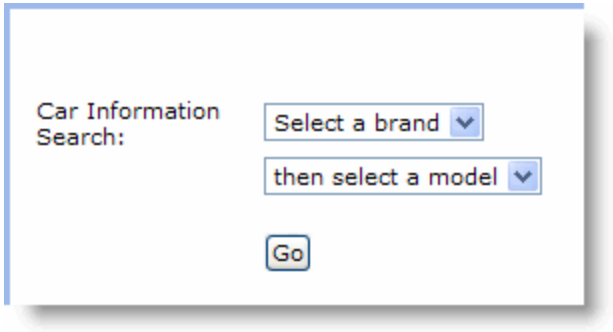
Next, as shown above, an **Event Handler** element has been added beneath the first Input Text element ("inpLoginID"). Its **DHTML Event** property has been set to "onChange". The **onChange** event only runs ("fires") when the cursor moves to another control *and* if a change has been made to the data in inpLoginID. In that case, processing is passed to any elements beneath the Event Handler. The **onChange** event fires at different times for different Input elements. For **Input Text** and related elements, it fires if a change has been made and the cursor leaves the control. For **Input Select List**, it fires when the selection is changed. For the **Input Radio Buttons** and **Input check box** elements, use the **onClick** event, which fires when a radio button or check box is clicked.

The use of Event Handlers that "intercept" the onChange event will prevent the **Change Flag Element** scheme, described with Input Hidden (see "User Input Elements" on page 566) and available to all user input elements, from working so you can't use both. Now let's see some specific Event Handler use-case examples.

Using Event Handlers and Dependent Controls

Developers often encounter requirements to have the contents of one Input Select List change dynamically based on user selections in another Input Select List. The second control is said to be "dependent" on the first one. Event Handlers provide an easy

method of enabling this dependency. The following example depicts a web page that allows users to get information about automobiles. This is a classic example of Input Select List dependency: once users select from a list of car brands, then a second list, of car models, is populated with choices based on the brand selection.

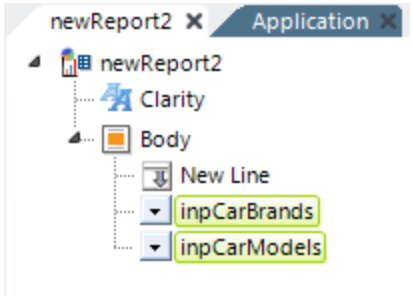


Here's how it works: when a car brand is selected, the page is submitted and the next page is called. The selected brand is passed to the next page as a request variable. The "next" page, however, will be the *same* report definition (the page calls itself or "refreshes") but now the selected brand will be available to it in a @Request token. This token is then used to populate the list of car models that match the selected brand. Here's the example data, a simple XML file, that we'll use:

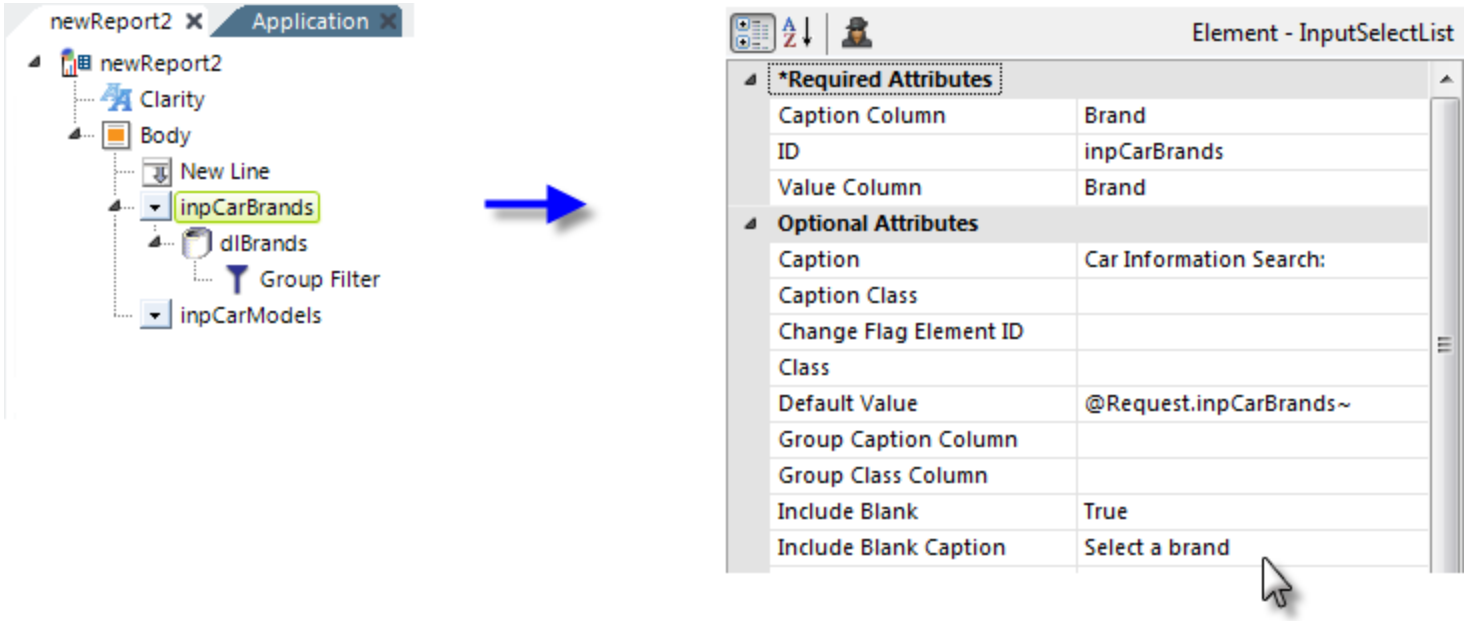
```
CarInfo>
<Car Brand="Alfa Romeo" Model="174 Hatchback" Code="1"/>
<Car Brand="Aston Martin" Model="Spider Open" Code="2"/>
<Car Brand="Audi" Model="Spider Open" Code="3"/>
<Car Brand="Bentley" Model="Spider Open" Code="4"/>
<Car Brand="BMW" Model="3 Series Convertible" Code="5"/>
<Car Brand="BMW" Model="3 Series Coupe" Code="6"/>
<Car Brand="BMW" Model="5 Series Saloon" Code="7"/>
<Car Brand="BMW" Model="5 Series Touring" Code="8"/>
```

```
<Car Brand="BMW" Model="7 Series Touring" Code="9"/>
<Car Brand="BMW" Model="M3 Coupe" Code="10"/>
<Car Brand="Cadillac" Model="Spider Open" Code="11"/>
</CarInfo>
```

For the purposes of this example, BMW is the only manufacturer that's been given multiple models.



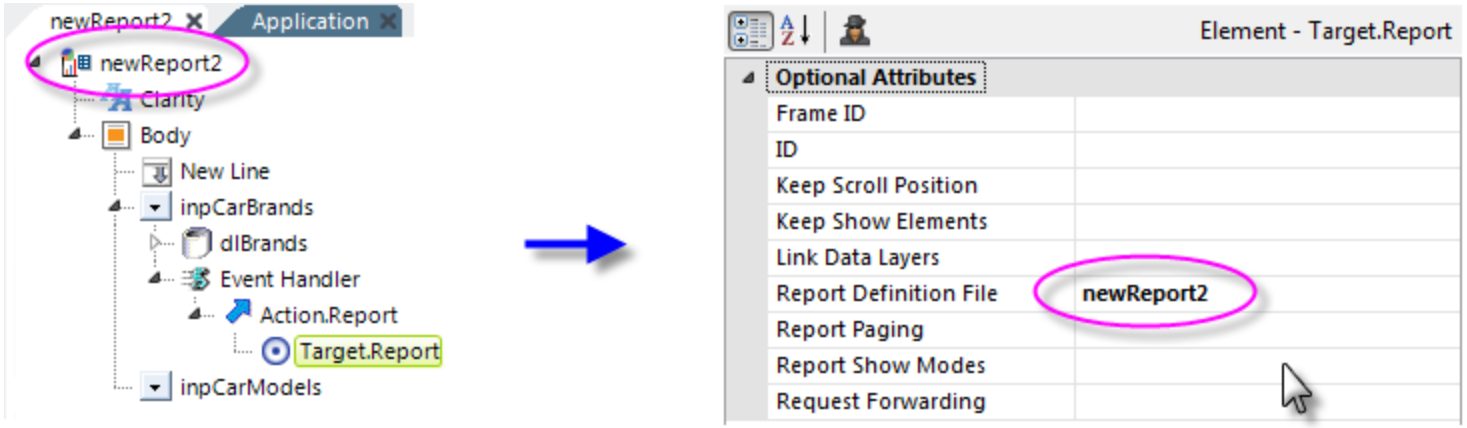
The example report definition, shown above, contains two **Input Select List** elements.



The attribute values for the car brands Input Select List are shown above. Note these two key attributes:

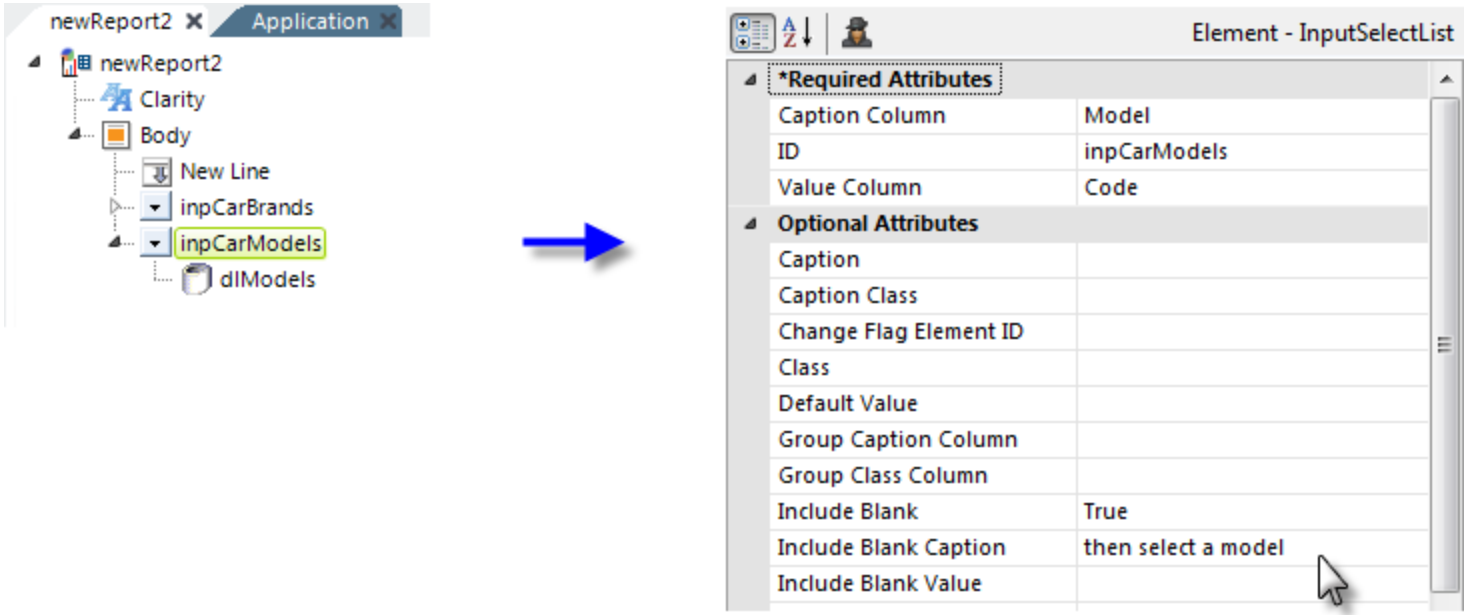
- **Value Column** - This attribute has been set to the same value as the **Caption Column** attribute, ensuring that the selected brand name is passed as a request variable.
- **Default Value** - This attribute is set to the request variable with the same **ID** as the Input Select List element, ensuring that the selected brand name will remain selected when the page refreshes.

In addition, datalayer and **Group Filter** elements have been added below the Input Select List. The Group Filter's **Group Column** attribute is set to the "Brand" column, ensuring that there are no duplicate entries in the list of brands.

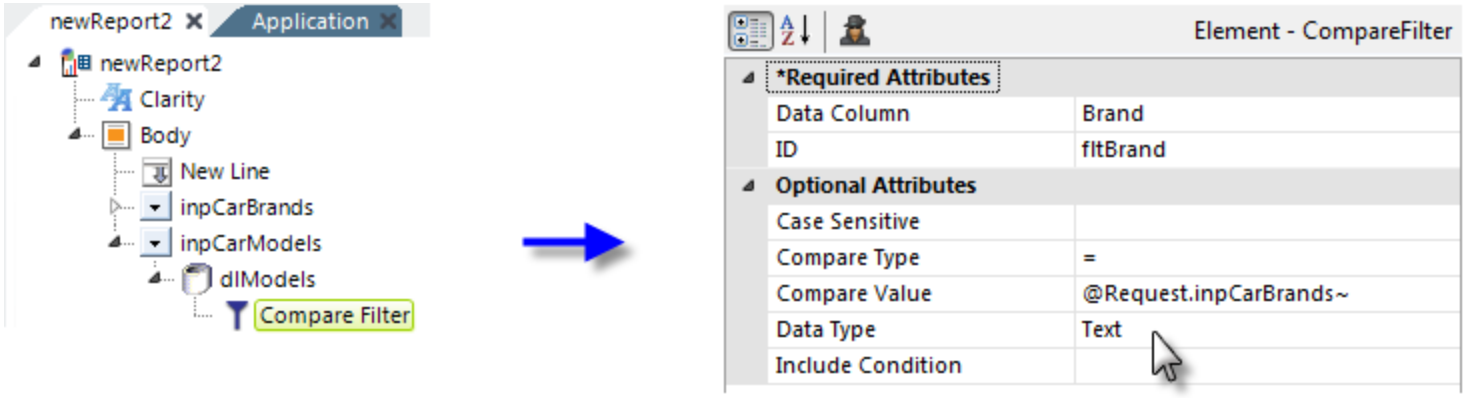


As shown above, an **Event Handler** element has also been added beneath the car brands Input Select List. Its **DHTML Event** attribute has been set to "onChange". In the case of an Input Select List, this event will fire as soon as a selection has been made. **Action.Report** and **Target.Report** child elements have also been added and the target definition file has been set to the name of the current report definition, "newReport2".

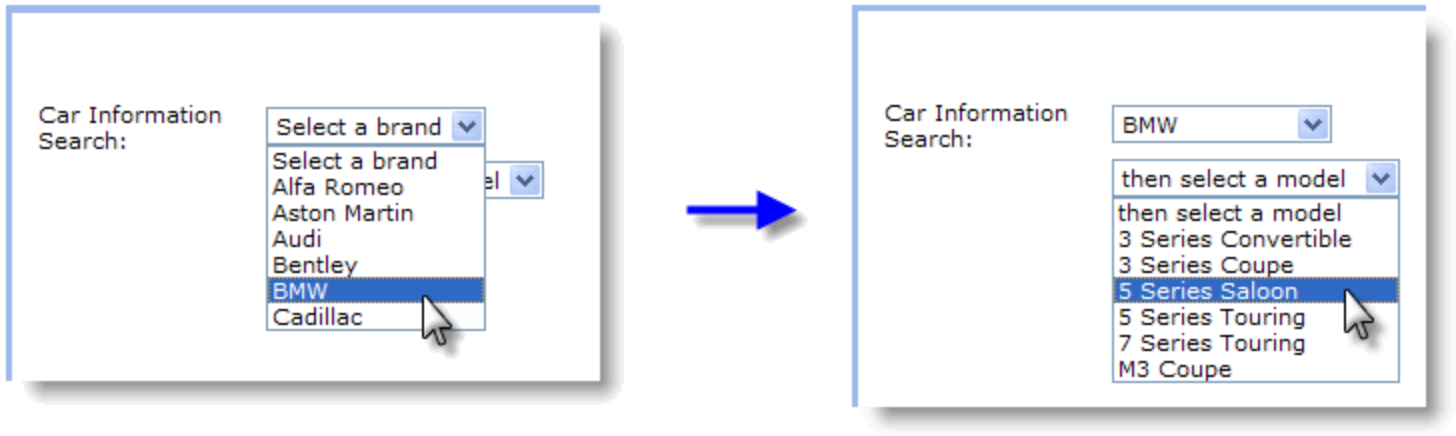
Do not use the *CurrentReport*option available in the list of choices for this attribute - Request variables are not passed when this is used.



The attribute values for the car models Input Select List are shown above; its attributes are pretty straightforward.



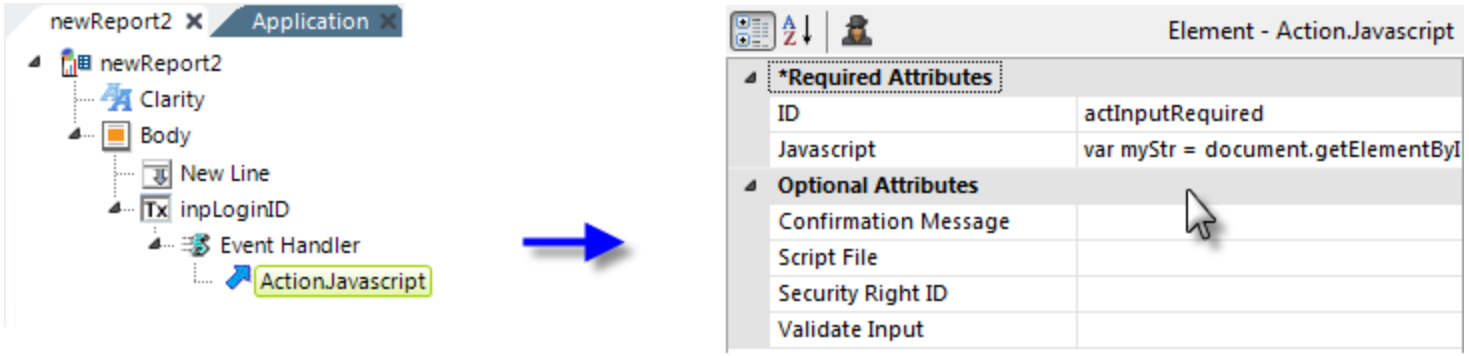
As shown above, datalayer and **Compare Filter** elements have been added below the Input Select List. The filter is the key to setting the second Input Select List to the models of the chosen brand: its attributes are set to filter the Model data based on the @Request token containing the Brand selection.



Finally, the interaction between the two select lists is shown above. The entire page will not repaint itself when a brand selection is made, so there'll be no annoying page flicker, due to the use of AJAX technology in Logi products. In summary, this example has shown how an event handler can be tied to a change in a user input control and how it can trigger other actions based on that change.

Using Event Handlers with Javascript

A very powerful feature of Logi Info is the ability to execute **JavaScript** when an event occurs.



Developers can use the **Action.Javascript** element, shown above, beneath an Event Handler element. JavaScript code can be entered directly into the **Javascript** attribute; multi-line code is supported. Now we're able to use the power of JavaScript and the DHTML object model for activities such as validation. The Attribute Zoom window comes in handy when entering code (double-click the Attribute name). Here are some use-case examples:

Example 1: Ensure that some data is entered

It's beyond the scope of this topic to delve too deeply into JavaScript or the object model, but essentially the following code *assigns* the data in the inpLoginID control to a variable, then *tests* the length of that variable's value, and *displays* a message box if the length is zero.

```
var myStr = document.getElementById('inpLoginID').value;
if (myStr.length == 0 ) alert('You must enter some message text.');
if (myStr.length == 0 ) return false;
```

`return true;` This is very similar to the **Validation.Required** element's functionality. The Document object's getElementById method is very handy in this situation. Remember: JavaScript is *case-sensitive*.

While you may enter multi-line JavaScript, you *may not* include code blocks that are set off by curly brackets. This means structures like If...Then...Else blocks are not usable. As shown in these examples, you can get around this by using multiple lines of code, each with a single If statement.

Example 2: Ensure some data is entered but not more than 4,000 characters

The example script that follows does something special: it *limits the length* of data that can be entered into an **Input Text Area** element.

```
var myStr = document.getElementById('inpTextArea1').value;
if (myStr.length > 4000) alert('Your message is too long (' + myStr.length + ' chars) - maximum length is 4,000 characters.');
if (myStr.length > 4000 ) return false;
if (myStr.length == 0 ) alert('You must enter some message text.');
if (myStr.length == 0 ) return false;
return true;
```

Input Text Area does not have a "maximum length" attribute. The code above sets a limit and if the text entered is too long, it reports the actual number of characters currently entered. It also requires that *some* data be entered.

Example 3: Replace all occurrences of '@' with '#'

This example demonstrates a method for *replacing* certain characters in the data entered, using a JavaScript *Regular Expression*.

```
var myStr = document.getElementById('inpTextArea1').value;
myStr = myStr.replace(/[@]/g, '#');
```

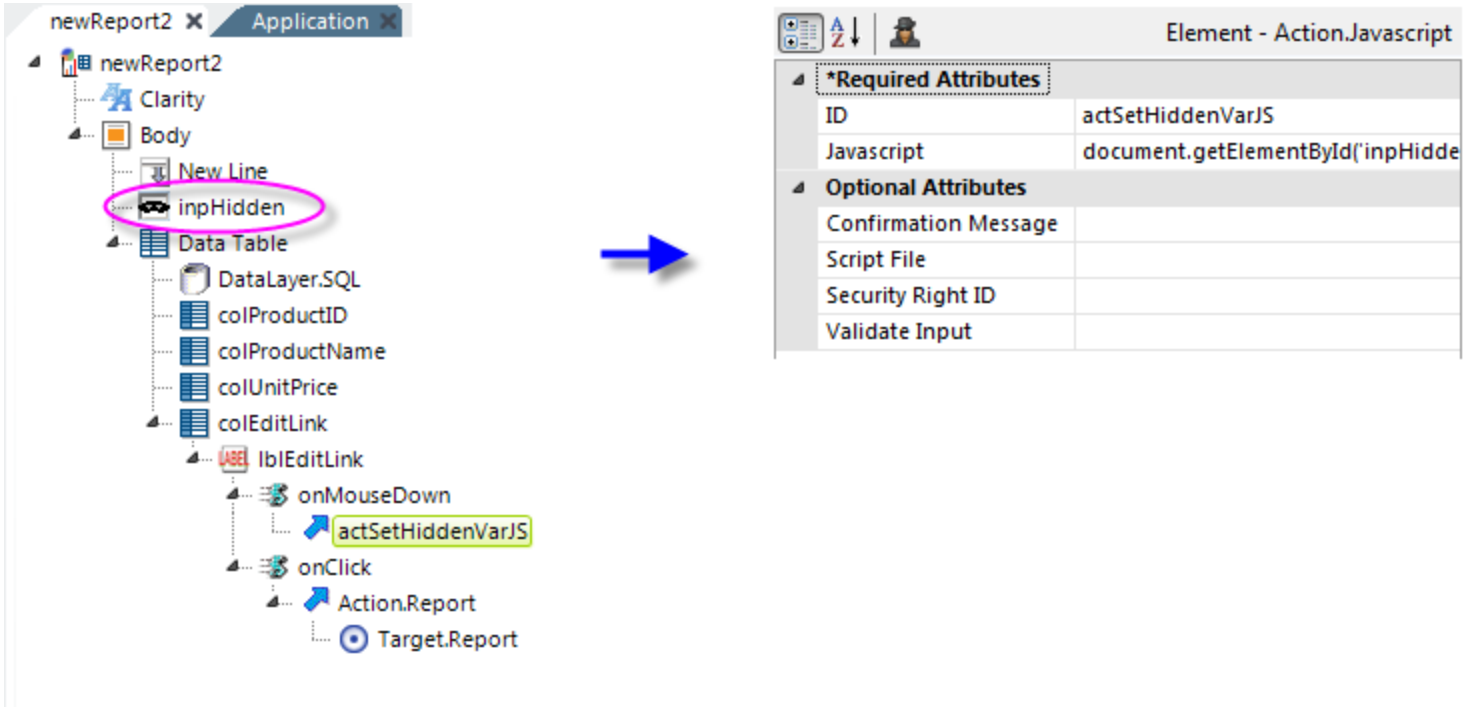
```
document.forms[0].inpTextArea1.value = myStr;

return true;
```

In the example, any @ characters that were entered by the user are replaced with the # character, and then the altered data is *substituted* for the data that was entered. Regular Expressions are very powerful, can be quite complex, and can perform actions such as adding, replacing, or removing characters, changing character case, and expanding or contracting phrases. As you can see, the ability to run JavaScript whenever data changes in user input elements provides developers with a very powerful validation tool.

Example 4: Handling Links within Tables

This is a slightly different kind of example that deals with a common problem. Suppose you have a Data Table and each row has a column in it that contains a link or button; for instance, an "Edit" button. When the link or button is clicked, you want to send data from that row to another report or process task. Internally, the Logi engine adds a row designator - "_Row1, _Row2, etc." - to the datalayer column names, so addressing the correct data using tokens (@Request.myCol_Row#~) in the receiving Report or Process task is difficult. However, there's an easy solution: add an **Event Handler***and* an **Action** element beneath the link or button. The Event Handler will assign the data from the current table row to one or more Input Hidden elements (depending on how many columns you need to send) and the Action element submits the page and calls the next Report or Process task. The next report or task can then work with the values in the @Request token for the Input Hidden elements.



The example above shows the elements that might be used in this situation. First, an **Input Hidden** element is added at the top of definition, *outside* of the table and its columns. We'll put the value we want to work with in this element, getting around the @Request.myCol_Row# issue. Inside one of the table columns, beneath the Label element (which is the "Edit" link) an **Event Handler** is added and its **DHTML Event** attribute is set to *onMouseDown*. Beneath the Event Handler, ActionJavascript is added and its code set to:

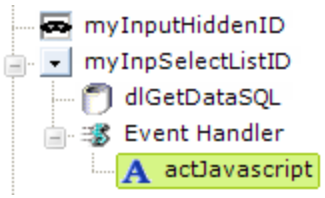
```
document.getElementById('inpHidden').value = @Data.ProductID~;
return true;
```

So, when the mouse button is depressed on the link, the data from the ProductID column for the current row will be assigned to the Input Hidden element. And then another **Event Handler**, for the actual mouse-click, is added with child **Action.Report** and **Target.Report** elements. These submit the page and calls the next Report or Process task, where the Product ID value will be

available as `@Request.inpHidden~`. This is a common and useful technique that allows you to perform multiple actions based on what seems like one mouse click.

Passing Input Select List Captions

When a page containing an Input Select List (ISL) is submitted, the value from the designated Value Column in the selected row is passed to the next definition as a @Request token, using the ISL's element ID. However, developers may want to also, or instead, pass the **Caption** for the selected item. In this release, the Input Select List and Input check box List elements were enhanced to incorporate the ability to pass the captions of the items selected. For more information, see "User Input Elements" on page 566. This functionality is not available in earlier versions, so you must use JavaScript to make the text available. Here's how:



1. Include an **Input Hidden** element in your definition (users will not see it), as shown above.
2. Use an **Event Handler** to capture the *onChange* event beneath your **Input Select List.**
3. Beneath the Event Handler add **Action.Javascript.**
4. Enter the following JavaScript code in the **Source** attribute (keeping in mind that it's case-sensitive):

```
/* get index of selected item in Input Select List (ISL) */
var ndx = this.selectedIndex;
/* get the text in ISL of selected item */
var selText = this.options[ndx].text;
/* development only: popup to show us the text */
```

```
        alert('Text: ' + selText);
        /* stuff the selected text into an Input Hidden element */
        document.getElementById('myInputHiddenID').value = selText;
        /* uncomment the following to also submit the page */
        /* document.forms[0].action =
'http://localhost/myApp/rdPage.aspx?rdReport=nextDef'; */
        /* document.forms[0].submit(); */
        return true;
```

If you wish to submit the page right after the selection is made in the ISL, un-comment the next-to-last two lines and provide the definition name. As you can see, the code stuffs the Caption (text) of the ISL's selected item into the Input Hidden and you can then read it in the next definition using @Request.myInputHiddenID~.

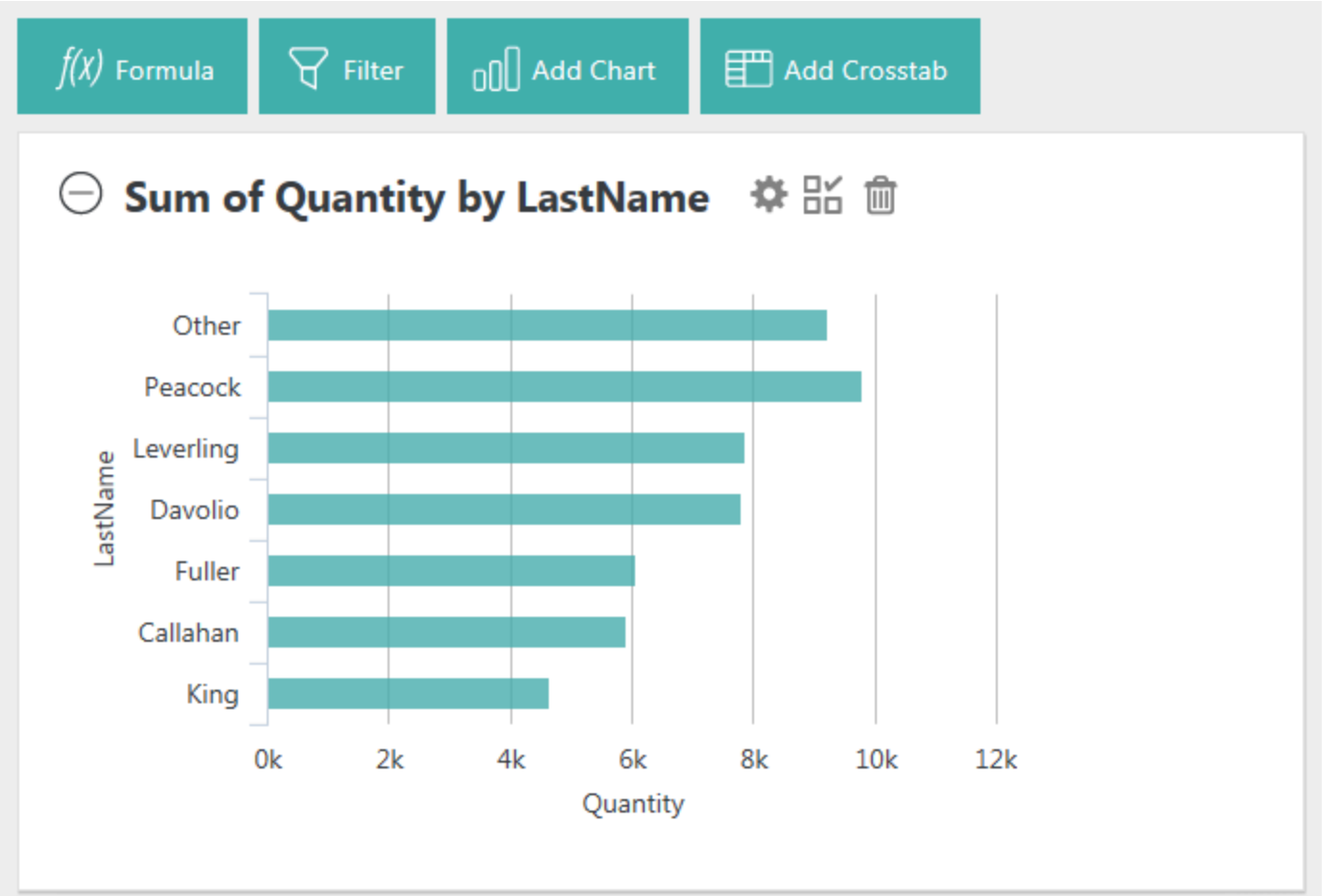
Super Elements

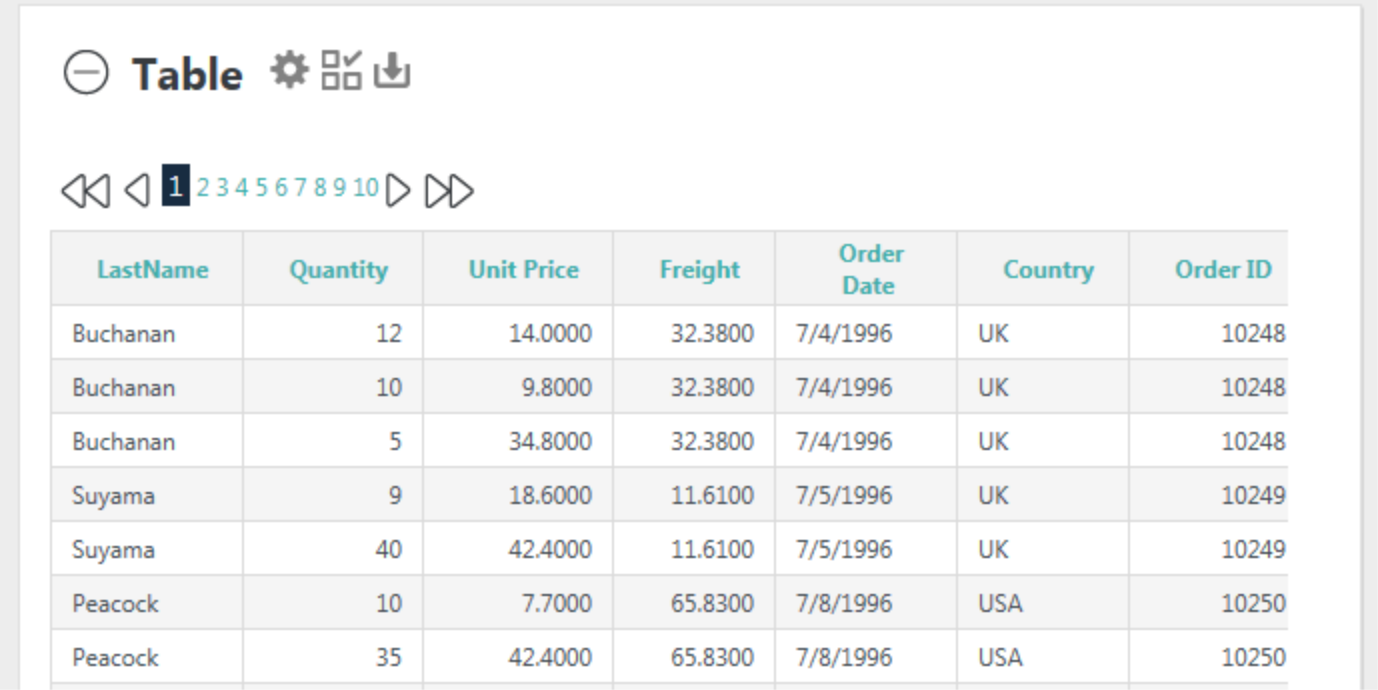
Logi Info includes several pre-built, interactive "mini apps" that provide rich functionality to end users. Classified as "Super-Elements", they provide developers with instant functionality, and end-users with interactive controls that allow them to manipulate their data visualizations at runtime. This topic introduces the Super-Elements.

- About Super-Elements
- Table of Super-Elements

About Super-Elements

Super-Elements are special collections of other elements, bound together as a single element, which produce a sophisticated "mini-application" that a developer can include in a Logi report. At runtime, users can be given access to controls that allow them to select data and construct visualizations on the fly. Developers have complete control over what data is available for manipulation and which features are available to users.


Formula
Filter
Add Chart
Add Crosstab
Sum of Quantity by LastName
Other
Peacock
Leverling
Davolio
Fuller
Callahan
King
LastName
0k
2k
4k
6k
8k
10k
12k
Quantity



LastName	Quantity	Unit Price	Freight	Order Date	Country	Order ID
Buchanan	12	14.0000	32.3800	7/4/1996	UK	10248
Buchanan	10	9.8000	32.3800	7/4/1996	UK	10248
Buchanan	5	34.8000	32.3800	7/4/1996	UK	10248
Suyama	9	18.6000	11.6100	7/5/1996	UK	10249
Suyama	40	42.4000	11.6100	7/5/1996	UK	10249
Peacock	10	7.7000	65.8300	7/8/1996	USA	10250
Peacock	35	42.4000	65.8300	7/8/1996	USA	10250

An **Analysis Grid**, shown above, illustrates the kinds of features available in a Super-Element. It includes user interface controls in a toolbar at the top and both a Data Table and a chart below. At runtime, the user can filter the data and customize the table and charts. Export controls are also available, if the developer allows them. Links for drill-downs into detail reports are supported.

In addition, most Super-Elements allow the customizations made by users to be saved between sessions and recalled later.

Developers should note that you can only use one Super-Element, such as an Analysis Grid, per report definition. Because of internal identifiers within them, using multiple Super-Elements on the same page would cause identity confusion. You can, how-

ever, place Super-Elements in several separate report definitions and then include the definitions, as SubReports in IncludeFrame mode, in a single master definition, resulting in the appearance of multiple Super-Elements in the same "page".

Customizing Super-Elements

Super-Elements make it very easy for developers to provide a lot of functionality with a minimum of development effort, while retaining control and security. Because they present an interface to the user, techniques are available for internationalizing the interface. Interface text, date formats, currency symbols, etc. can be translated into the proper culture and language dynamically, if necessary.

Almost all of the Super-Elements are built using their own "template" definitions, which are distributed with the product. Developers can modify these templates to internationalize or otherwise create custom variations of the Super-Elements. The techniques used for this purpose are described in *Template Modifier Files*.

Table of Super-Elements

The following table enumerates the Super-Elements. Click the element name for additional information and the View icon to run a sample application using the element.

Super-Element	Description
Analysis Chart	The Analysis Chart allows users, at runtime, to display non-animated charts, based on a discrete data set and limited to chart types the developer chooses to include. For more information, see "The Analysis Chart" on page 130.

Super-Element	Description
Analysis Filter	The Analysis Filter allows users to apply filtering, at runtime, to the datalayers that support their visualizations. Changes made to the filtering immediately causes the visualizations to be updated. For more information, see "The Analysis Filter" on page 168.
Analysis Grid	An Analysis Grid consists of separate panels for controls, configuration, Data Tables, charts, and cross-tab tables. At runtime, users can manipulate the controls, creating data analysis and presentation on the fly. For more information, see *The Analysis Grid for Developers*.
Chart Grid	The Chart Grid displays a tabular arrangement of bar, line, or scatter charts, with each table cell containing a separate chart. All of the charts in the grid will be of the same type. Users can manipulate the display at runtime. For more information, see *The Chart Grid*.
Dashboard	A "Dashboard" is a collection of panels containing Logi reports, which in turn contain tables, charts, images, etc. At runtime, the user can customize the Dashboard by rearranging these panels on the browser page, by showing or hiding them, and even by changing their contents using adjustable reporting criteria. Dashboards can also include tabbed pages of panels. For more information, see *Logi Info Dashboard* .
Dimension Grid	The Dimension Grid element is a Data Table with interactive controls and charting options that allows the user to "slice and dice" data from a Logi XOLAP cube, at runtime. For more information, see *Dimension Grid Wizard*.
Logi Scheduler	The Logi Scheduler provides the ability to execute reports and other activities on a scheduled basis. The Schedule element presents a "console" for managing scheduled events. For more information, see *Logi Scheduler*.

Super-Element	Description
OLAP Grid	The OLAP Grid element is a Data Table with interactive controls and charting options that allows the user to "slice and dice" data from an OLAP cube, at runtime. For more information, see "Logi OLAP" on page 487.
Report Author	The Report Author element allows end-users to work with a Dashboard-like interface to create their own basic reports at runtime. For more information, see *Report Author*

Data Menu

The **Data Menu** element is used to define a menu, with optional sub-menus, for use in Logi mobile reports.

The following topics provide guidance for using the Data Menu, with simple and complex examples:

- Data Menu Attributes
- Example: Simple Menu
- Example: Complex Menu (with sub-menus)
- Changing Appearance with Template Modifiers

About the Data Menu

Logi developers can use the **Data Menu** element to create menus and sub-menus for their mobile reports:



A typical menu is shown above. Each menu item appears in its own horizontal bar, followed by a right-justified ">" symbol. An optional image can be included to the left of the menu item. When the menu item is tapped, the application navigates to a sub-menu, a new mobile report, or a different URL.

The Data Menu can be used to create a simple menu or more complex menus that include a hierarchy of sub-menus.

The menu items, links, and all other required information are retrieved into a datalayer, so the menu can be data-driven and security can be applied. The *attributes* of the Data Menu element are used to identify the *columns* in the datalayer whose values comprise the menu items, image file names, actions, etc. that make up the menu.

The examples below assume that you're using a Logi Info build that includes the Data Menu element and that you're working in a Mobile Report definition.

Data Menu Attributes

The Data Menu element has the following unique attributes:

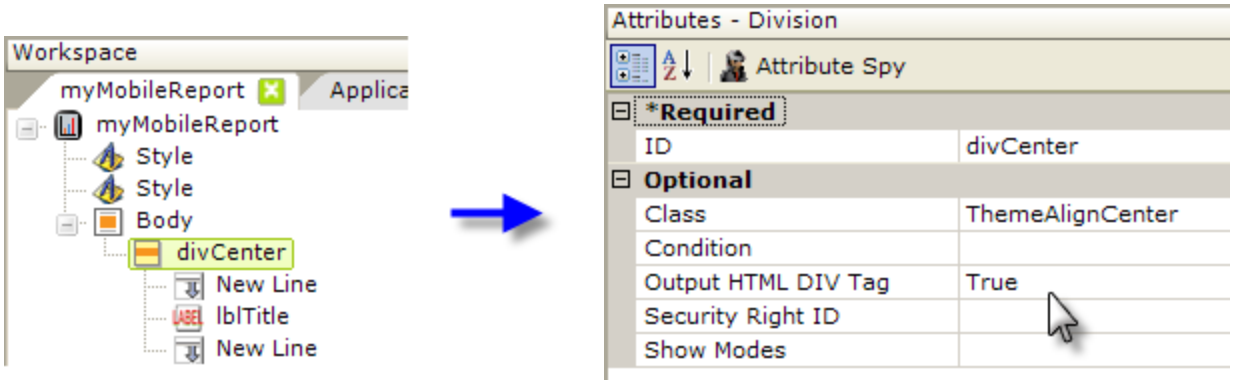
Attribute	Description
Caption Column	(Required) The name of the column in the datalayer whose values will be displayed as the menu items.
Action ID Column	The name of the column in the datalayer whose values are the ID of a child Action element that will be executed when the menu item is tapped. This is used for menu items that call another report definition, a URL, etc., not for navigating to sub-menus.
Image Column	The name of the column in the datalayer whose values are the names of image files. These images are displayed to the left of the menu item. If the image file is stored in the _SupportFiles folder, no file path is required. Relative file paths may be used. Example: *abcnews.png*
Sub Menu ID Column	The name of the column in the datalayer whose values are the names of the sub-menus, if any, that each menu item *resides in*. Used in data rows for sub-menus only.
Target Sub Menu ID Column	The name of the column in the datalayer whose values are the names of the sub-menus to *navigate to* when each menu item is tapped. Used only when sub-menus exist.

When working a hierarchy of sub-menus with **DataLayer.Static**, it is possible to construct and use **Static Data Rows** that do not necessarily all have the same number of columns. Different data rows contain data for different purposes and the Logi Server Engine won't object if columns exist in one row but don't exist in another row where they serve no purpose.

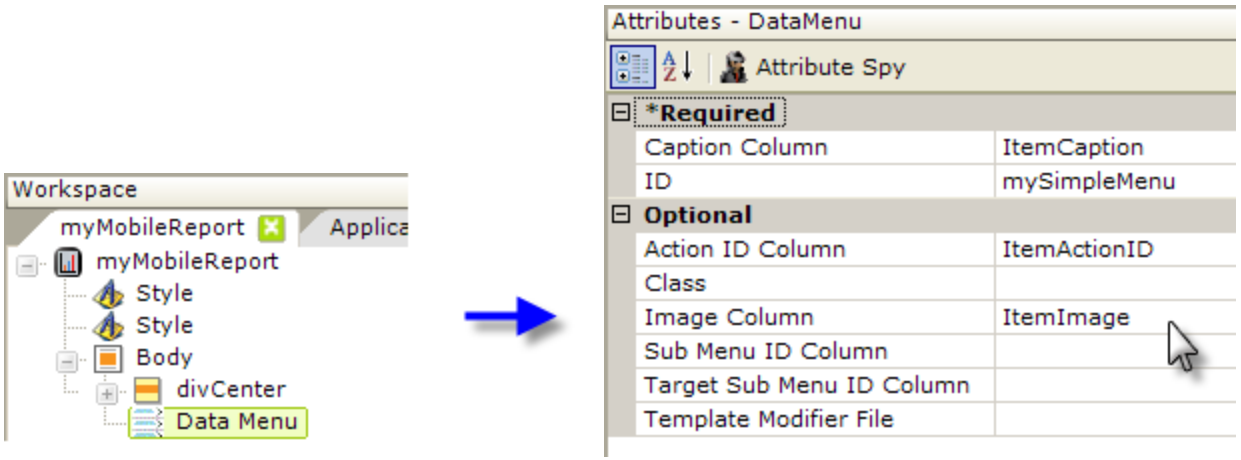
Obviously, if you use a database to provide your menu data, then all rows returned by a query will have the same number of columns. In that case, you'll need to ensure that the query returns *all* of the columns needed for *all* purposes.

Data Menu Example: Simple Menu

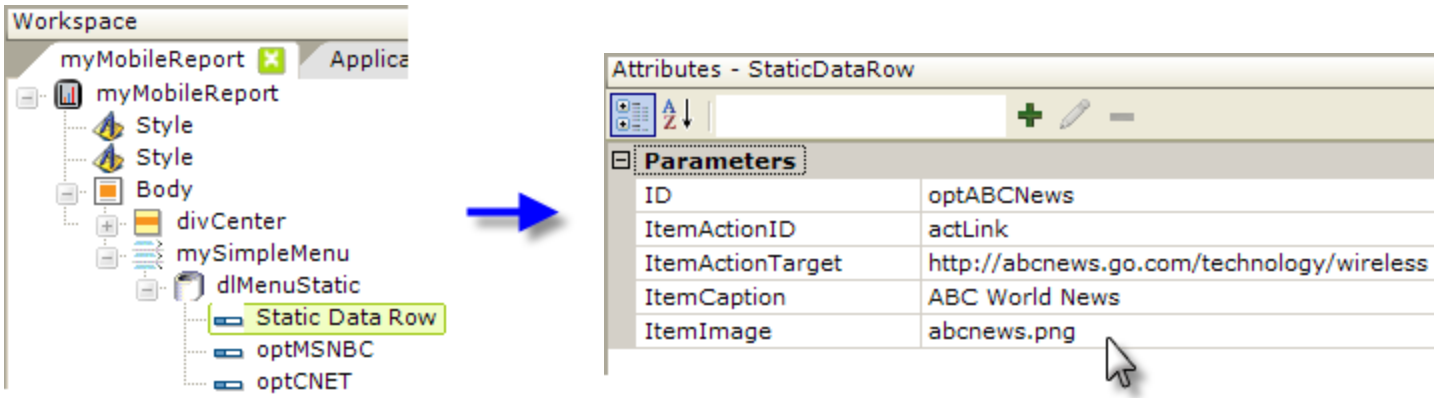
The following example demonstrates how to construct a simple, one-level main menu in a Mobile Report.



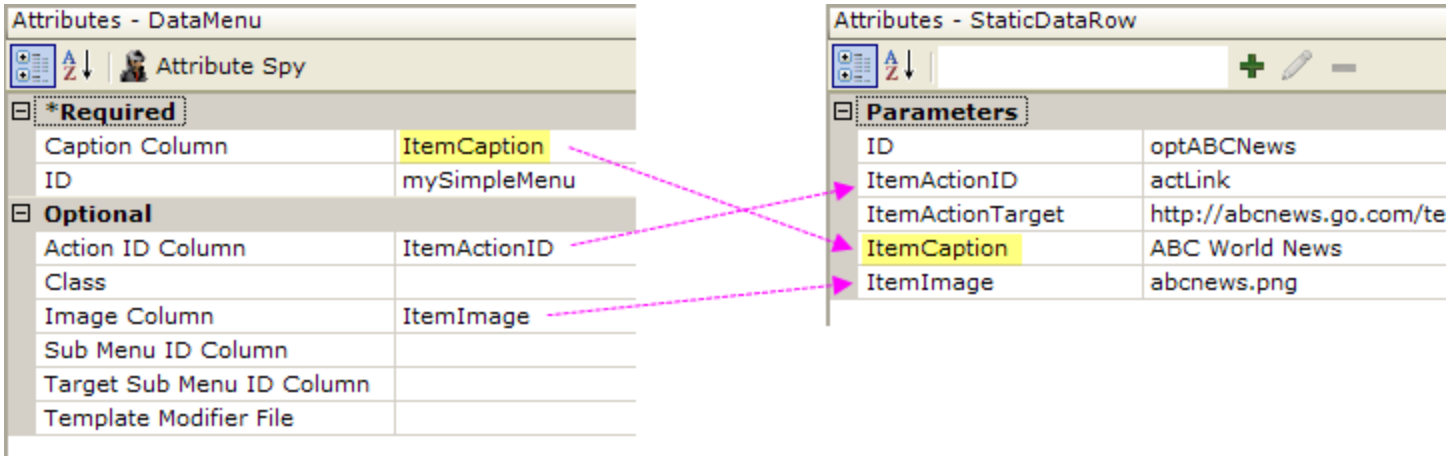
1. Begin by configuring a theme and adding a title. Add a **Style** element and select a theme, as shown above, and, if desired, add a second Style element and select the Rounded theme.
2. Add a **Division** element and set its attributes as shown above. Beneath it add **New Line** and **Label** elements to provide a menu title.



3. Add a **Data Menu** element beneath the Body element and set its attributes as shown above. Although we recommend you use the values in this example for now, the text that's used in these attributes is completely arbitrary; you may use any value that makes sense to you.

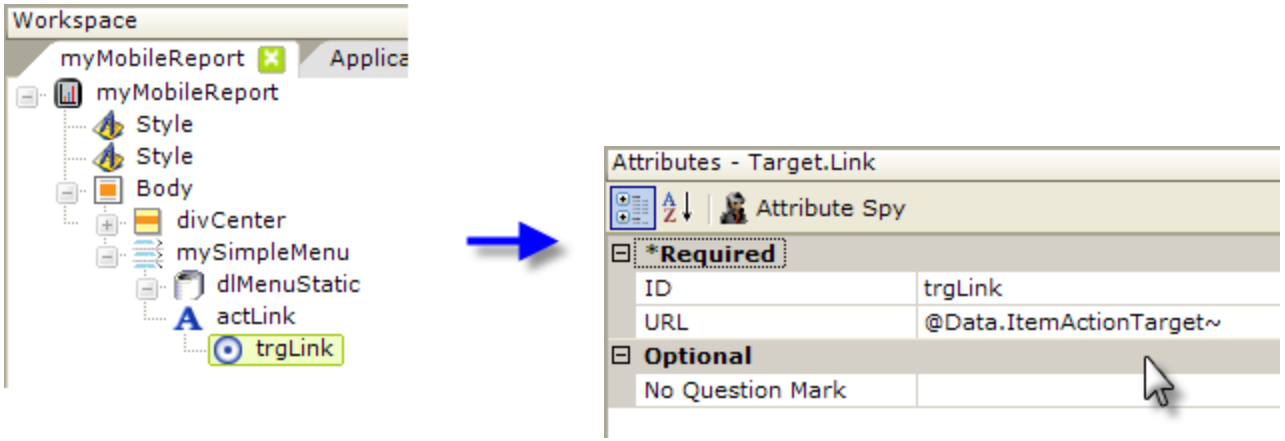


4. Next, add a **DataLayer.Static** element beneath the Data Menu, as shown above.
5. Add a **Static Data Row** element beneath the datalayer, and set its dynamic attributes as shown above.



The Data Menu element's attributes match the data row's "column" names. One additional column not found in the Data Menu attributes has also been added: **ItemActionTarget**. This is used by Action elements we'll add later.

6. Repeat the previous step a few times to create additional menu items.

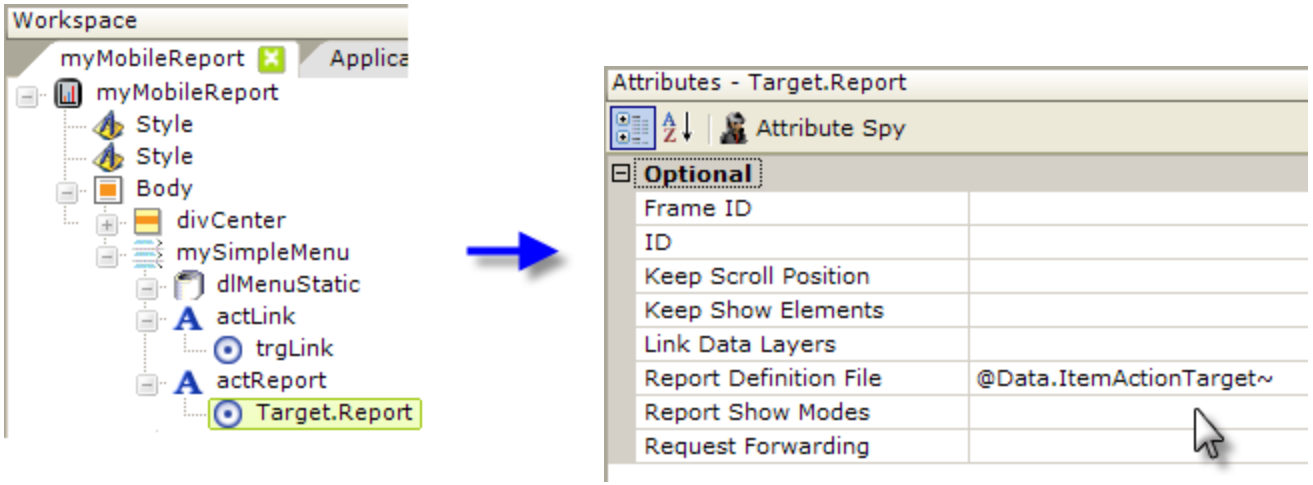


7. Add an **Action.Link** element beneath the Data Menu element, and give it an element ID that matches the value of the ItemActionID attribute in the Static Data Rows (i.e. "actLink"). This identifies the action to be executed when the menu item is tapped.
8. Finally, add a **Target.Link** element beneath the Action element and set its URL attribute as shown above. This is where that "extra" Static Data Row attribute is used. In general you can add any number of extra attributes and access them using @Data tokens in this manner. This might be useful, for example, for passing data to the target URL or report using Link Parameters.

Your menu is now ready. Click the Preview button in Studio and an image similar to the one at the top of this topic should be displayed.

Navigating to Another Report

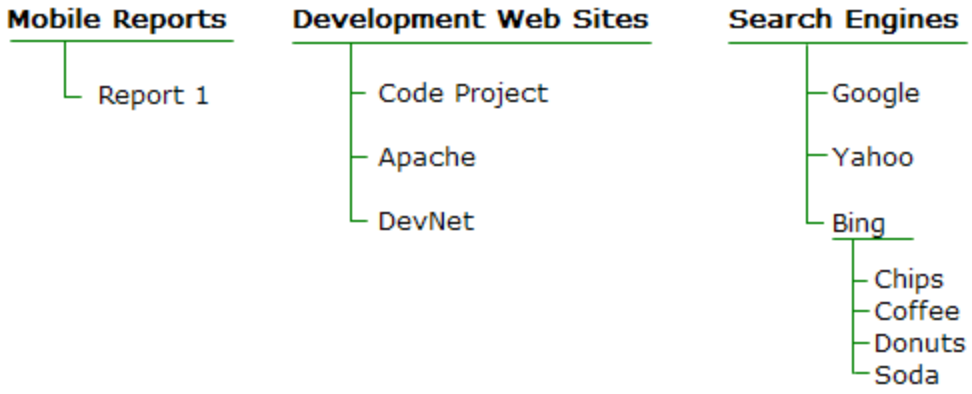
If you want your users to tap a menu item and load a different Mobil Report definition, you can substitute **Action.Report** and **Target.Report** for Action.Link and Target.Link in our example. But, what if some menu items need to point to report definitions and some to URLs? In that case, you can "mix and match":



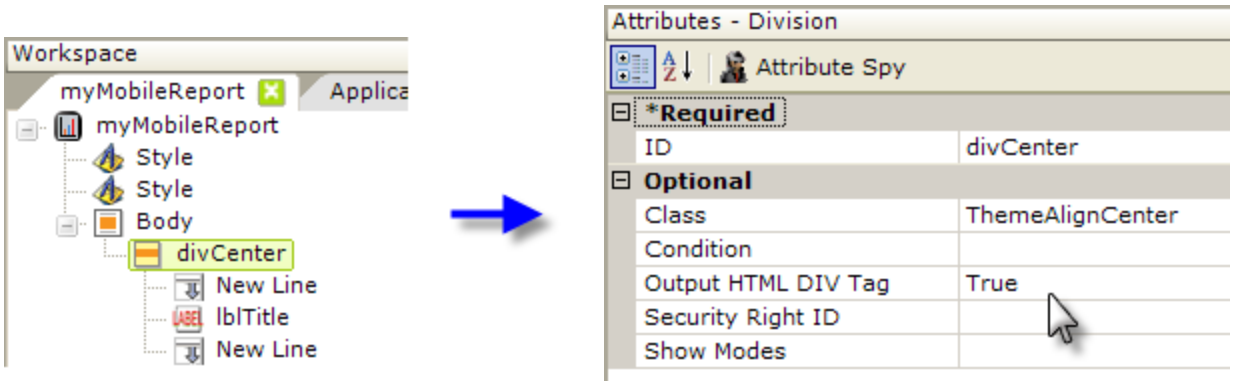
Add *both* Action.Link and Action.Report elements beneath the Data Menu, as shown above, then set the ItemActionID attribute value for each menu item to use the desired action element.

Data Menu Example: Complex Menu (with sub-menus)

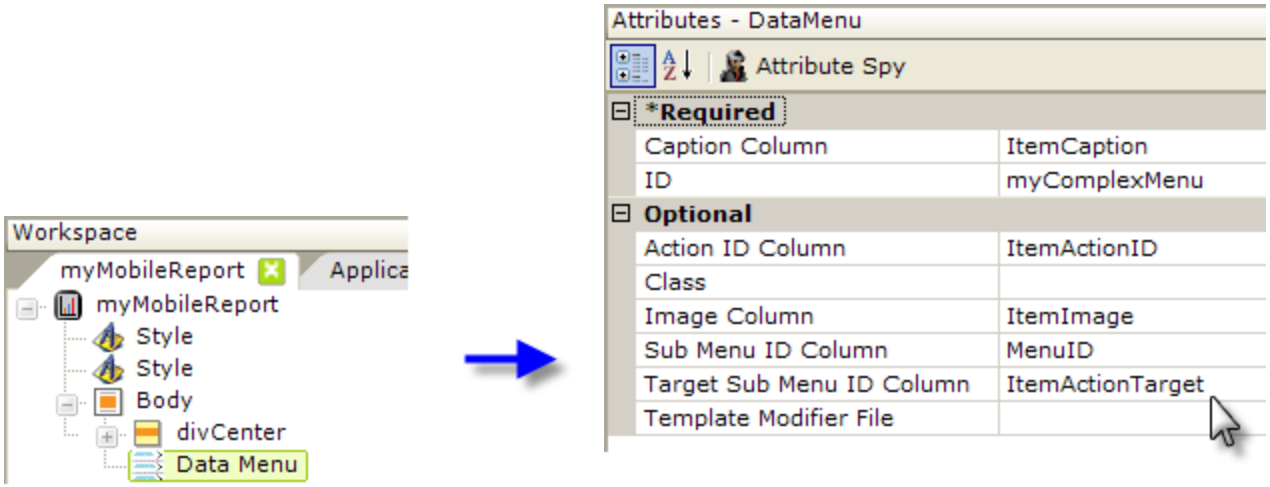
The following example demonstrates how to construct a more complex menu, with sub-menus, in a Mobile Report. Here's the menu hierarchy we'll build:



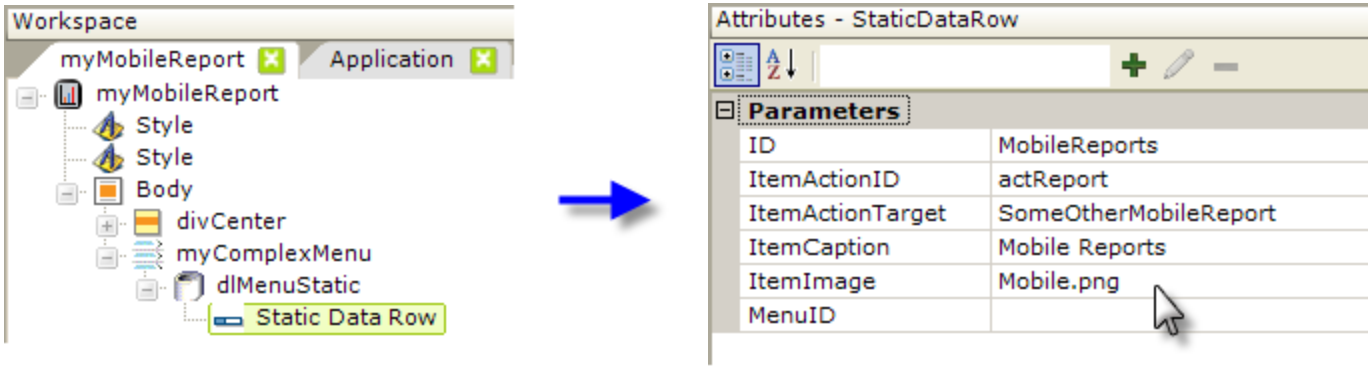
The first main menu item will point to a different mobile report, and the second to a sub-menu of web sites. The last main menu item will point to a sub-menu that will have an item that points to another sub-menu.



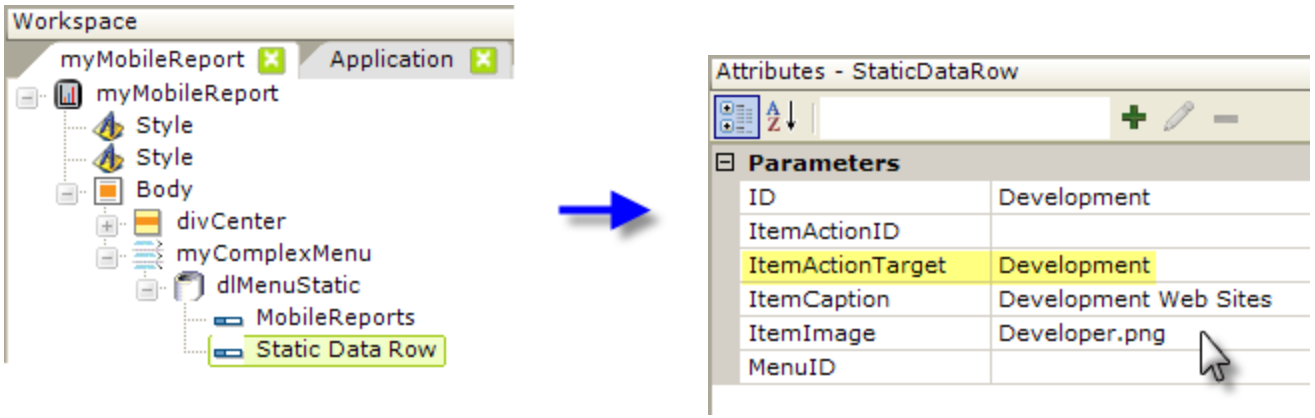
1. As before, begin by configuring a theme and adding a title. Add a **Style** element and select a theme, as shown above, and, if desired, add a second Style element and select the Rounded theme.
2. Add a **Division** element and set its attributes as shown above. Beneath it add **New Line** and **Label** elements to provide a menu title.



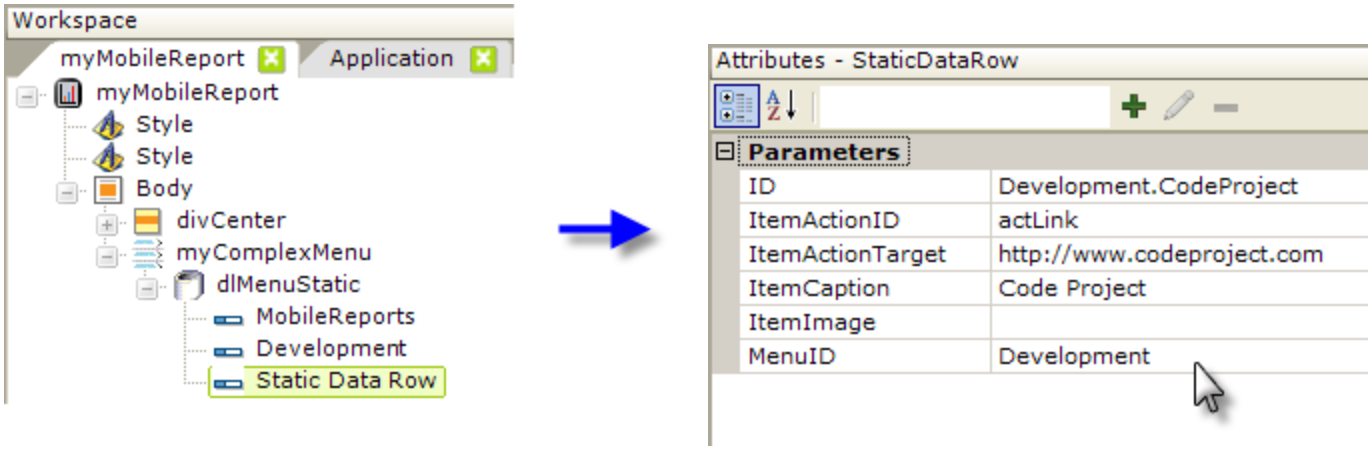
3. Add a **Data Menu** element beneath the Body element and set its attributes as shown above.



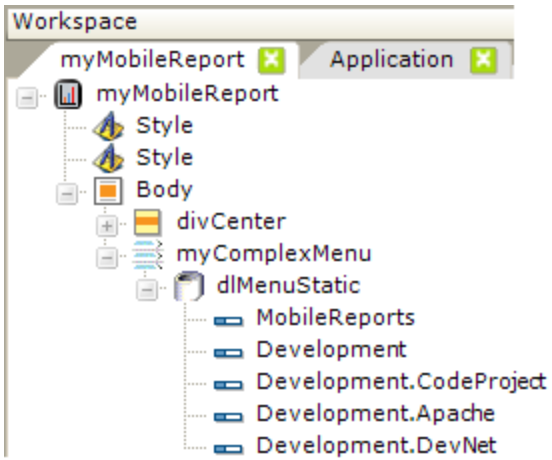
4. Next, add a **DataLayer.Static** element beneath the Data Menu, as shown above.
5. Add a **Static Data Row** element beneath the datalayer, and set its dynamic attributes as shown above. The data for this menu item is similar to that used for the Simple Menu example. This completes the first main menu item.



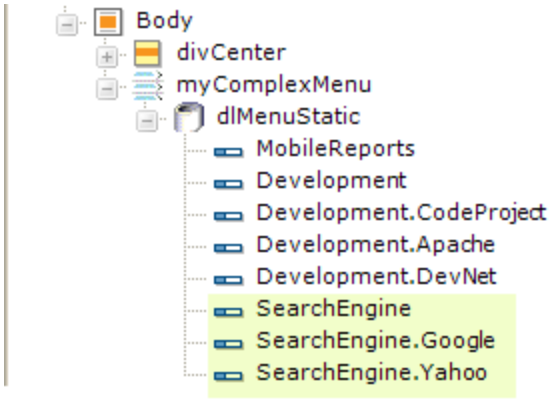
6. Add a second Static Data Row and set its column data as shown above. This item will point to a sub-menu, so note that its **ItemActionTarget** column is set to a value, "Development", that we'll use in the next steps to identify the sub-menu.



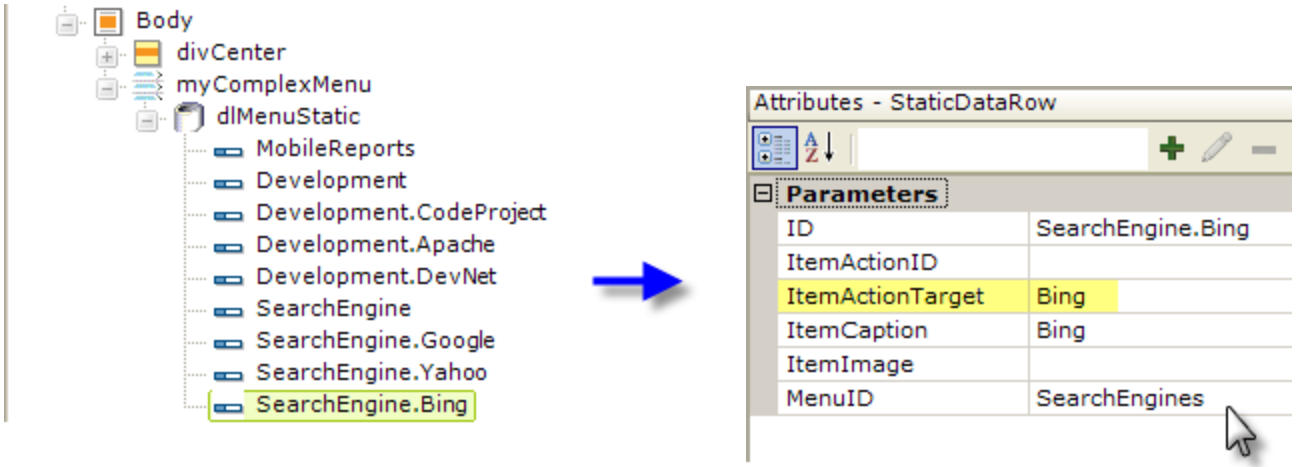
7. Add another Static Data Row and set its column data as shown above. The **ID** column contains a "dot" in it - this will make things easier to read in the Element Tree - and that the **MenuID** column is used to indicate that this item is in a sub-menu called "Development", which relates to the ItemActionTarget discussed in the previous step.



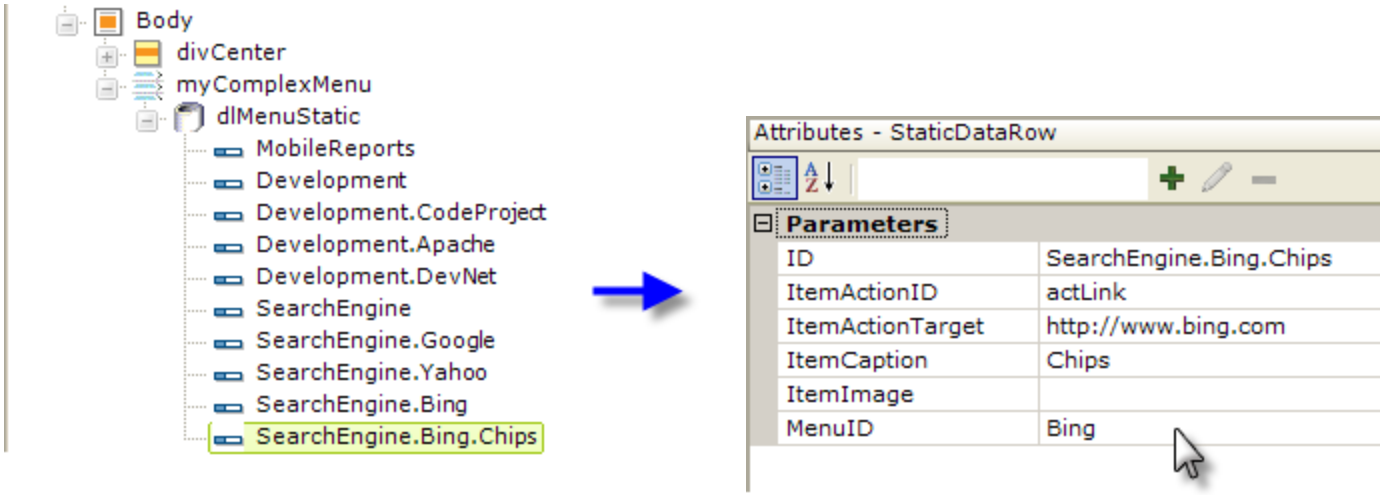
8. Add two more Static Data Rows and set their columns data in the same fashion as the previous row. These become the menu items in the sub-menu. This completes the second main menu item and its sub-menu.



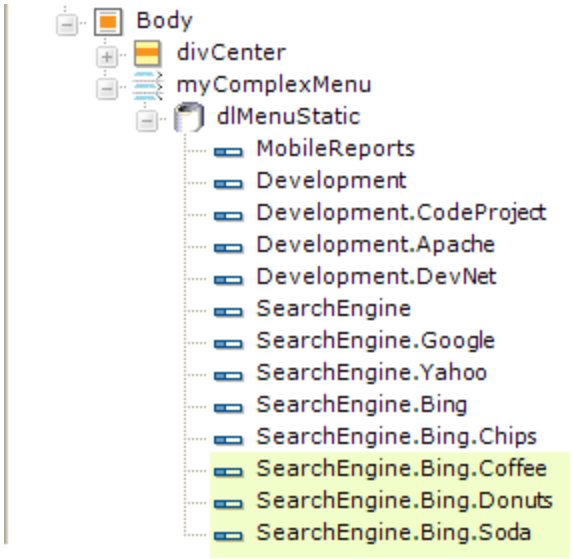
9. Repeat the last three steps to add three Static Data Rows for the Search Engines main menu item and two of its sub-menu items. Their column data values will be similar to those set previously. At this point, you can see the value of the "dotted" element IDs.



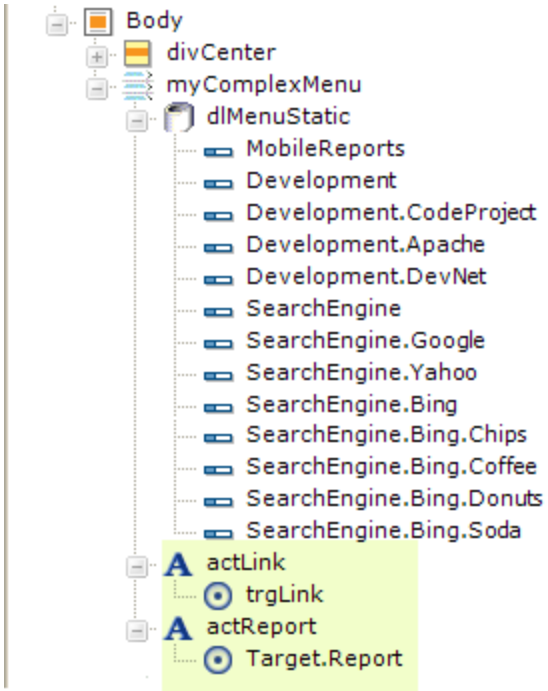
10. Add another Static Data Row and set its column data as shown above. This sub-menu item is going to point to another sub-menu, so note that its ItemActionTarget column value is "Bing", the ID of the next sub-menu.



11. Add another Static Data Row and set its column data as shown above. This is the first menu item in the second-level sub-menu. Its ItemActionID and ItemActionTarget column values will be used by the Action elements we'll add in a few steps.



12. Repeat the last step to add Static Data Rows for three more Bing sub-menu items. Their column data values will be similar to those set previously.



13. Finally, add the **Action.Link** and **Action.Report** elements, with child **Target** elements, as shown above, to provide an action when a menu item (which does not point to a sub-menu) is tapped. Be sure to set the Action elements' ID attribute to match the ItemActionID column value in the Static Data Rows, as appropriate (see Step #7 in the Simple Menu example).

Your menu is now ready. Click the **Preview** button in Studio and test out the menu, sub-menus, and menu items.

Changing Appearance with Template Modifiers

The Data Menu element uses a "template file" to define certain element properties that are not otherwise available as attributes to the developer for modification. These include language- and culture-specific attributes that you may want to change for locale-based reasons (or you may simply want to change the elementto better suit your application).

To do this, you can use the element's **Template Modifier File** attribute to identify a custom XML file you create containing custom elements that will override the same elements in the template file.

The Data Menu template file is:

```
<yourAppFolder>\rdTemplate\rdPopup\rdDataMenuTemplate.lgx
```

More detailed information about template modifier files can be found in *Template Modifier Files*.

The Wait Panel

Logi applications interact with the web server, generally, in a serial or synchronous manner: one operation has to complete before the next one begins. In cases where operations may be time consuming, developers can include the **Wait Panel** element, which displays a transparent, modal, "please wait" overlay on the current page.

- About the Wait Panel Element
- Using the Wait Panel

About the Wait Panel Element

The **Wait Panel** element is associated with the start of an action and displays a modal panel, with animated image, until action processing is complete. The Wait Panel element can be added as a child of a number of elements, including:

- Action.Link
- Action.Process
- Action.Refresh Element
- Action.Run Bookmark
- Analysis Chart
- Analysis Grid
- Chart Grid
- Dimension Grid
- General
- Target.CSV
- Target.Export Word Or Excel
- Target.Google Spreadsheet
- Target.Native Excel
- Target.Native Word

- Target.PDF
- Target.Report
- Target.Template
- Target.XML

You can see, from the list, that it covers a variety of potentially time-consuming operations. What happens is that, when the parent Action occurs, the current web page is overlaid with a transparent, modal page, with an animated "please wait" icon, which remains in place until it's replaced by whatever page follows it.

The Wait Panel is not available for use in **SubReports** with a SubReport Mode attribute setting of *Embedded*.

Logi Info comeswith a standard HTML "wait" page which is called by the Wait Panel element. This file and its supporting files are located in your application's rdTemplate folder.

Global Wait Panel

The **Wait Panel** element is a child of the _Settings definition's **General** element and is automatically added to all new applications. When present in the _Settings definition, it's global in effect and causes a Wait Panel to be enabled anywhere one can be used, providing a default for the entire application. The Wait Panel element attribute values become the defaults for the entire application.

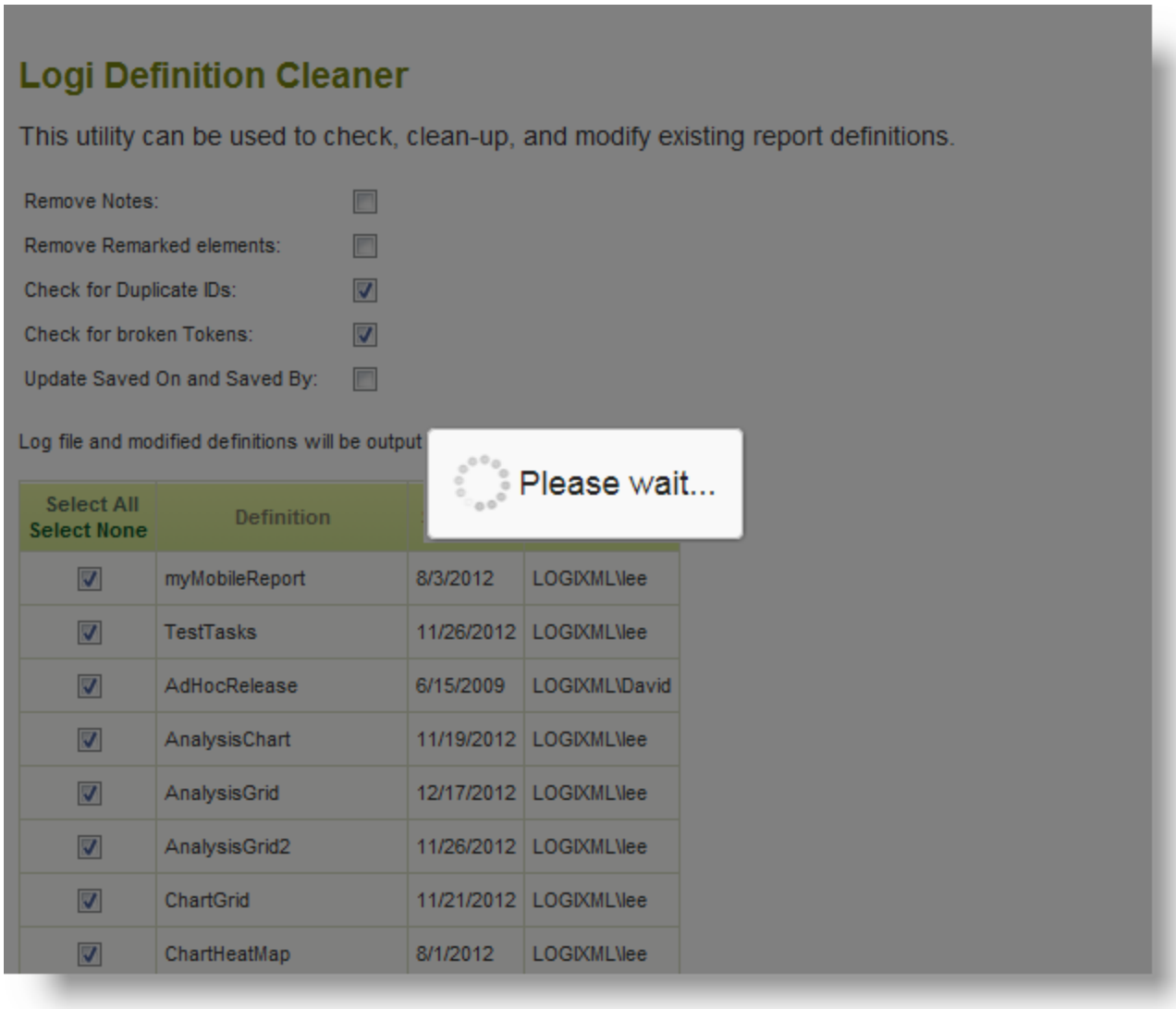
If a "local"Wait Panel element exists in a report definition, the attributes of that element will be used when displaying a Wait Panel. However, any empty local element attribute values, such as Caption, Class, or Caption Class, will default to the values set in the global Wait Panel element in _Settings.

For all Wait Panel elements (except in the scenario in the previous paragraph):

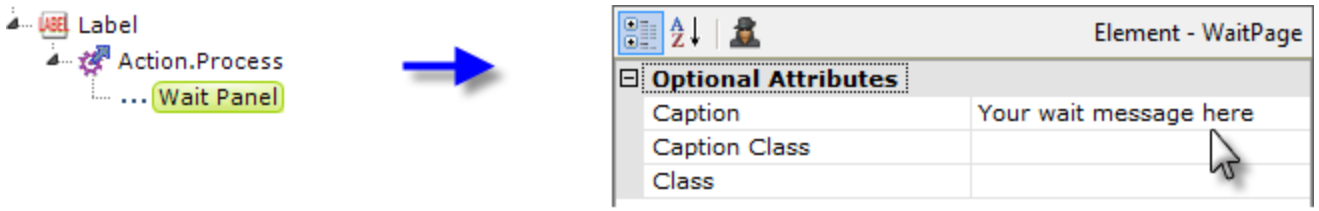
- If there is no specific Caption attribute value specified, the value defaults to "Please Wait...".
- If there is no specific Class value specified, the valuedefaults to "rdThemeWaitPanel".
- If there is no specific Caption Class value specified, the valuedefaults to "rdThemeWaitCaption".

Using the Wait Panel

The Wait Panel element displays a transparent, modal page over the current page and includes an animated icon, as shown below:



To implement it, the **Wait Panel** element is added beneath one of its parent elements:



As shown above, all of its attributes are optional, and you do not need to configure them to display the page shown earlier.

However, you can enter a customized message in the **Caption** attribute or apply style classes to the caption or the display in general using the **Caption Class** and **Class** attributes.

The Caption attribute *does not* support formatting, formulae, or multiple lines, but will evaluate tokens.

Once displayed, the page remains visible until the browser receives another page from the server to replace it, or the user otherwise navigates to another page.

Tabbed Panels

Tabs and **Tab Panel** elements in Logi Info provide developers the ability to visually group charts, reports, and other elements using panels that can be selected by tabs.

The following topics discuss the Tab and Tab Panel elements:

- Basic Tab Elements
- Working with the Tabs Element
- Working with the Tab Panel Element

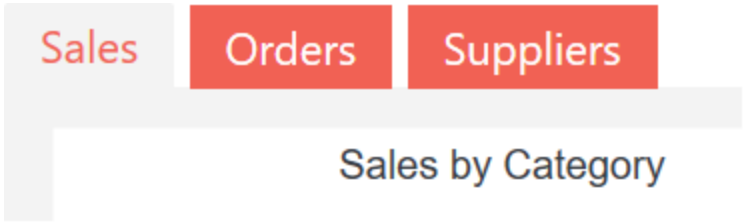
About Tabbed Panels

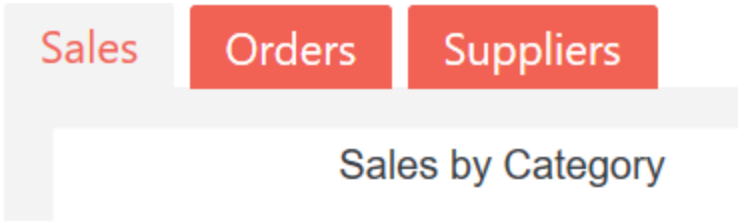
Tabbed panels provide a neat method of grouping things together and are a commonly-understood mechanism in user interfaces.



As shown above, each panel has a "tab" that extends beyond the panel, either at the top or bottom or the sides. The selected panel is presented as being "in front" of the unselected panels. The tabs themselves can have a different appearance depending on whether they're the selected tab or not.

Tab panels may contain just about any other element, although the use of super-elements, such as the **Analysis Grid**, within a tab panel is *not* recommended. In addition, *do not* use tab panels within an *Embedded*-type subreport (within an *IncludeFrame*-type subreport is OK). All of these elements use AJAX, and nesting them in these ways causes unpredictable AJAX behavior.



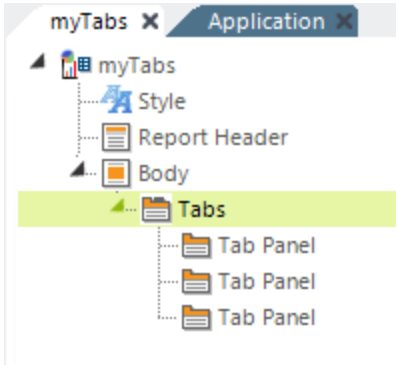


The appearance of the tabs and tab panels can be controlled both with style classes and through the use of themes. As shown above, the theme can be used to round all the square corners (availability may vary by browser).

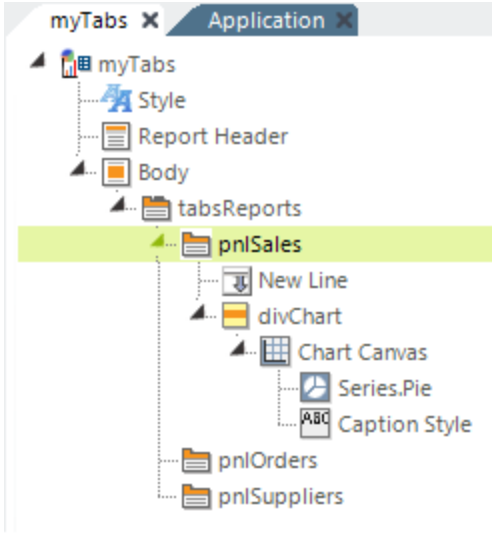
You may use multiple Tabs and Tabbed Panel sets in a single report definition, however, they will all have the same style-related appearance.

Basic Tab Elements

Two elements, **Tabs** and **Tab Panel**, are used to implement tabbed panels in Logi reports:



As shown above, the **Tabs** element is the parent of one or more **Tab Panel** child elements.



As mentioned earlier, Tab Panel elements can contain a wide variety of other elements, including those for tables, charts, user input, images, labels, and more. In the example above, a Chart Canvas pie chart has been placed in the first tab panel.

Notice that the example shown above uses a Division element ("divChart") in the Tab Panel, and it contains the pie chart. This is the proper way to *align* content in the panel. The division's Class attribute can be set to a style class that will, for example, center its contents, and you must also set its Output HTML Div Tag attribute to *True* to make it work.

Division elements also include "HTML Attribute Params", enabling you to apply your application to work with other frameworks or libraries easier. With the HTML Attribute Params, you have the option to include "style" parameters. If an attribute was set in both Element and HTML attribute params, the one set in the HTML attribute params will be ignored.

You can specify an exact size for Tab Panels but, if you do not, they will automatically size themselves to their contents.

Working with the Tabs Element

The **Tabs** element is the parent element for one of more Tab Panels elements. The attributes of the Tabs element are:

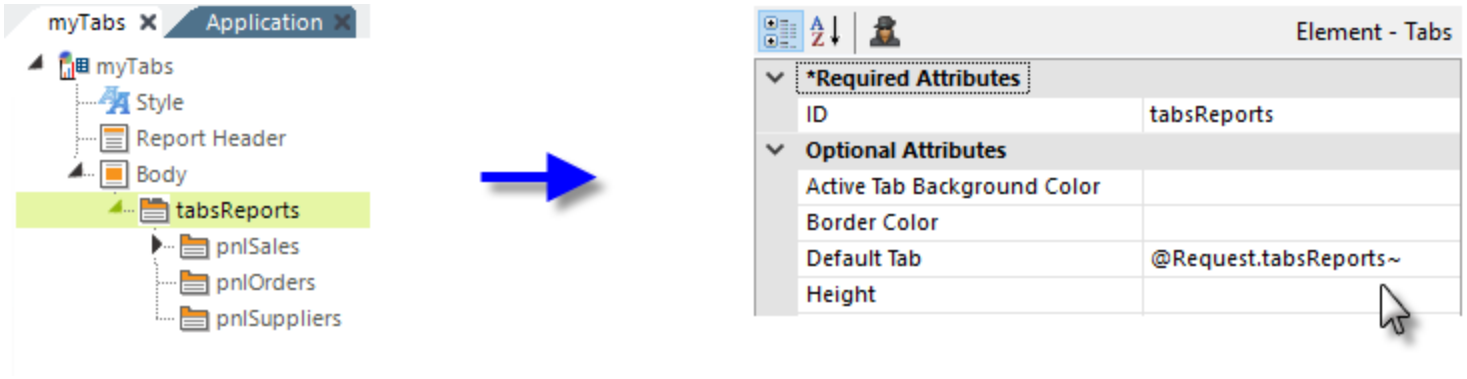
Attribute	Description
ID	(Required) Specifies a unique element identifier.
Active Tab Background Color	Specifies the background color for the active tab panel. Enter a color by name, decimal RGB value, or hex RGB value. Prefix hex values with the pound sign, such as #112233.
Border Color	Specifies the color of the border lines for all tab panels. If border lines are not desired, you may select "Transparent". Enter a color by name, decimal RGB value, or hex RGB value. Prefix hex values with the pound sign, such as #112233.
Default Tab	Specifies the ID of the Tab Panel that will be initially selected when the report is displayed. *See the paragraph below concerning how to maintain the tab selection between page refreshes.*
Height Height Scale	Specifies a fixed height for the tab panels. Leaving this value blank will cause the panels to dynamically resize themselves based on their content. The adjacent **Height Scale** attribute specifies whether this value is in to be applied in pixels or as a percentage.
Inactive Tab Background Color	Specifies the background color for inactive tab panels. Enter a color by name, decimal RGB value, or hex RGB value. Prefix hex values with the pound sign, such as #112233.

Attribute	Description
Security Right ID	Controls access to this element using Logi Security. Provide the ID of a Right defined in the application's _settings definition and only users that have a Role referenced in the Right will be able to see the element. Multiple Right IDs, separated by commas, may be entered.
Tab Location	Specifies location of the tabs that the user clicks to switch between tab panels. Options include: *Top*, *Bottom*, *Left*, and *Right*. The default value is: *Top*.
Tabbing Style	Specifies when tab panel *content* will be refreshed and rendered. Possible values are: *Static* - (default value) The content of all tabs is retrieved and rendered when Tabs element is displayed. When the user selects a hidden tab, the tab panel's content is already at the browser and the selected tab and content are displayed immediately *Ajax* - The contents of tabs other than the default tab are *not* retrieved and rendered until each tab is selected. When the user selects a tab, an AJAX request is sent to the server to get its panel's contents. This can help reduce rendering time when there are many tabs, especially if those panels contain datalayers. If a tab is re-selected after other tabs have been selected, the panel is simply redisplayed without going back again to the server for content. *RefreshPage* - Unselected tab panel contents are not processed by the server until the user selects their tab. When selected, the panel page is always re-submitted and therefore displayed with refreshed content. As with the previous mode, this can help reduce rendering time when there are many tabs, especially if their panels contain datalayers.
Width Width Scale	Specifies the fixed width of the tab panels. Leaving this value blank will cause the panels to dynamically resize themselves based on their content. The adjacent **Width Scale** attribute specifies whether this value is in to be applied in pixels or as a percentage.

Maintaining the Tab Selection Between Page Refreshes

It may be desirable to maintain the current tab selection when the page is refreshed. For example, if a Data Table is displayed in a tab panel and its records can be sorted by clicking column headers, the current tab should remain selected after a column header is clicked, the table is resorted, and the page is refreshed.

When a page with tabs is submitted using Action elements, the ID of the currently selected Tab Panel is automatically sent out as the value of a **request variable**. The request variable name is set to the ID of the Tabs element. Referring to the example definition shown earlier in this topic and assuming that the Sales tab was selected, this request variable would be sent when the page is submitted: tabsReports = pnlSales.



So, by setting the **Default Tab** attribute of the Tabs element to the request variable token, as shown above, the current tab will be maintained when the page is refreshed.

Refreshing or Navigating to a Specific Tab

If you'd like to refresh your page, or navigate to it from another page, and ensure that a specific tab will be selected after it loads, it's easy to do.

First, set the Tabs element's **Default Tab** attribute to a Request token. In the previous example, this value was set to @Request.tabsReports~.

Then, in the URL used to call the report, include this query string parameter: &<*Tabs Element ID*> = <*Tab Panel Element ID*>

To do this when refreshing your report page or navigating to it from another Logi definition using Action.Report, assuming the element IDs from the previous example, use a **Link Parameters** element to add a request parameter with the name *tabsReports* and a value of *pnlOrders* to select the Orders tab.

To do this when navigating from an non-Logi HTML page or using Action.Link, the URL would be something like:

```
http://myLogiApp/rdPage.aspx?rdReport=ReportWithTabs&tabsReports=pnlOrders
```

Working with the Tab Panel Element

The **Tab Panel** elements are the visible component of tabbed panels and include the little tab that users click to change panels and the area that's displayed when they do so.

The attributes of the Tab Panel element are:

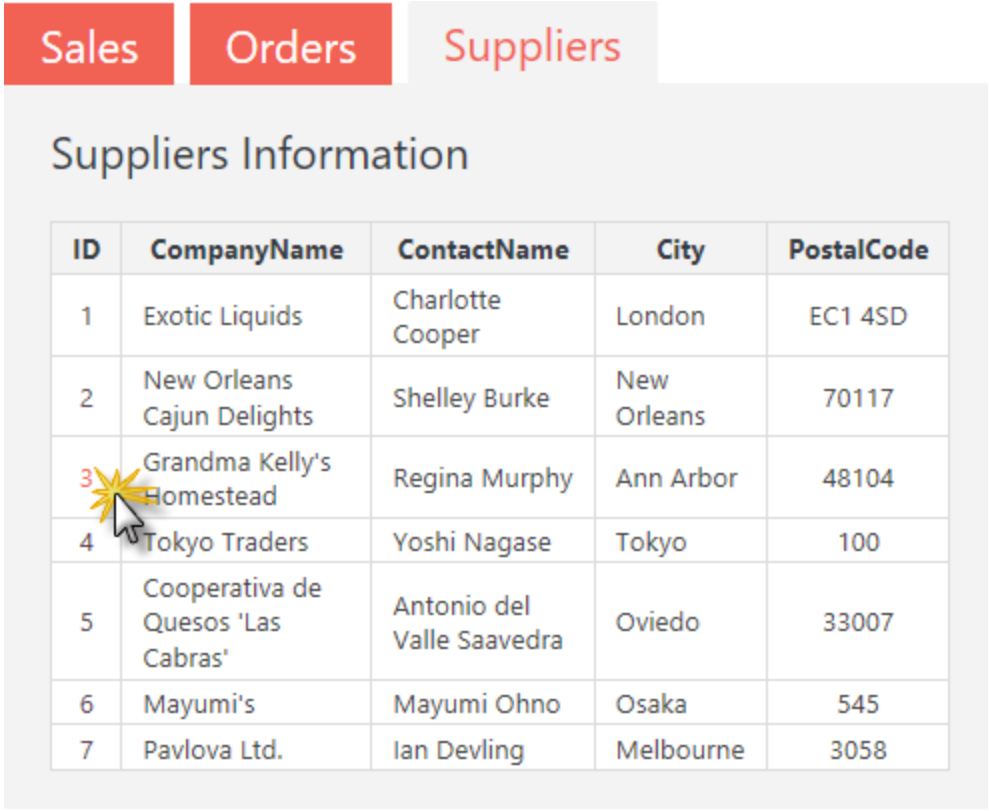
Attribute	Description
ID	(Required) Specifies a unique element identifier.
Caption	(Required) Specifies the text that will appear on the little tab that a user clicks (the selection tab) to change panels. This text appears as a link and, like all links, its appearance can be controlled using style sheet classes. If your caption seems too long, consider using an abbreviation and putting longer text into a Tooltip.
Class	Specifies the style sheet class to apply to the tab panel.  Classes are applied to the text in the selection tab link *and* to elements that appear within the Tab Panel, such as labels. For example, padding applied using classes will affect *both* the Caption text and elements in the Tab Panel. To isolate elements in the panel from the Caption text, consider adding a Division element within the panel, moving the panel contents within it, and then applying a class to it.
Security Right ID	Controls access to this element using Logi Security. Provide the ID of a Right defined in the application's _settings definition and only users that have a Role referenced in the Right will be able to see the element. Multiple Right IDs, separated by commas, may be entered.
Show	Specifies whether the tab panel is visible or hidden. This attribute can be used to dynamically increase or reduce

Attribute	Description
Modes	the number of tabs available for selection, based on runtime criteria. With this attribute set to an arbitrary string, such as "showMyPanel", you can control panel visibility by manipulating the *rdShowModes* request variable. For example, if you create a Default Request Parameter named rdShowModes and give it a value of showMyPanel, the panel will be displayed. If the value is something else, it will be hidden. Similarly, if instead you pass a request parameter named rdShowModes when calling or refreshing the report, its value will determine whether the tab panel is displayed.
Tooltip	Specifies text that appears when the mouse is **hovered** over a tab.  If this attribute is left blank, the word "active" will appear as a default tooltip.

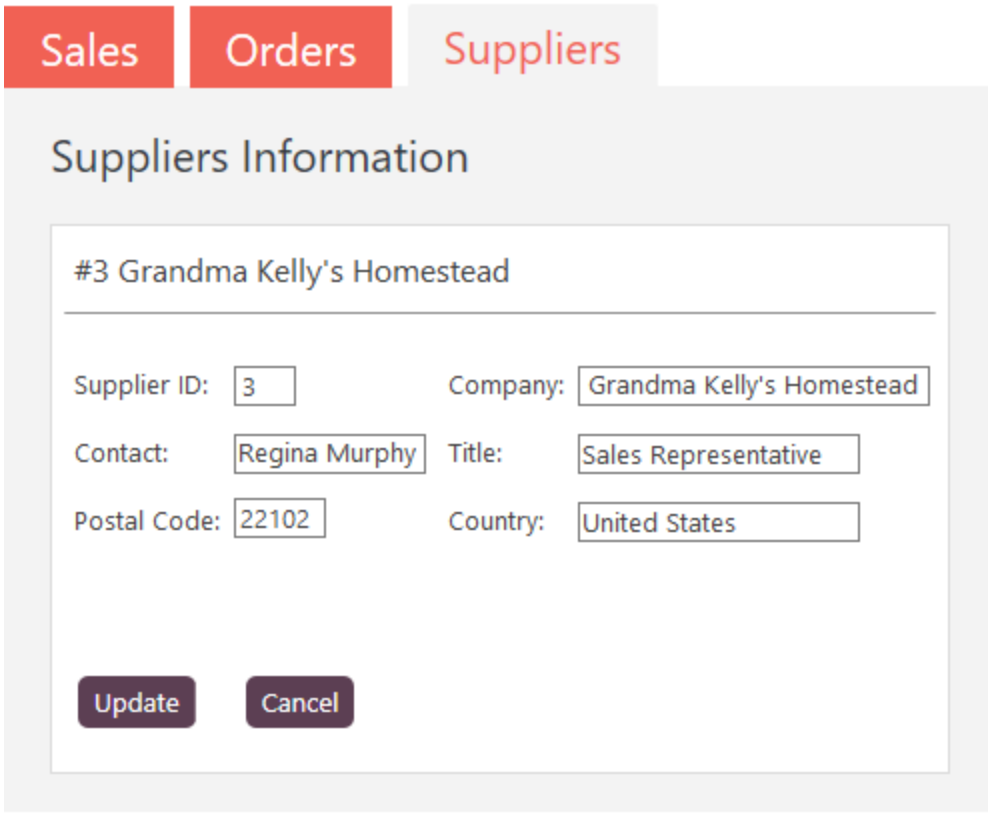
Changing Tab Panel Content Dynamically

This next example illustrates two more important ideas about working with Tabs. The first is that embedded subreports can be placed into tab panels, allowing you to develop panel contents as independent reports first, before integrating them into your tab panel. This can make debugging and report polishing much easier.

The second is that tab panel contents, through the use of **Division** elements, can be dynamic. By showing and hiding divisions, based on request variables, one tab panel can have multiple uses.

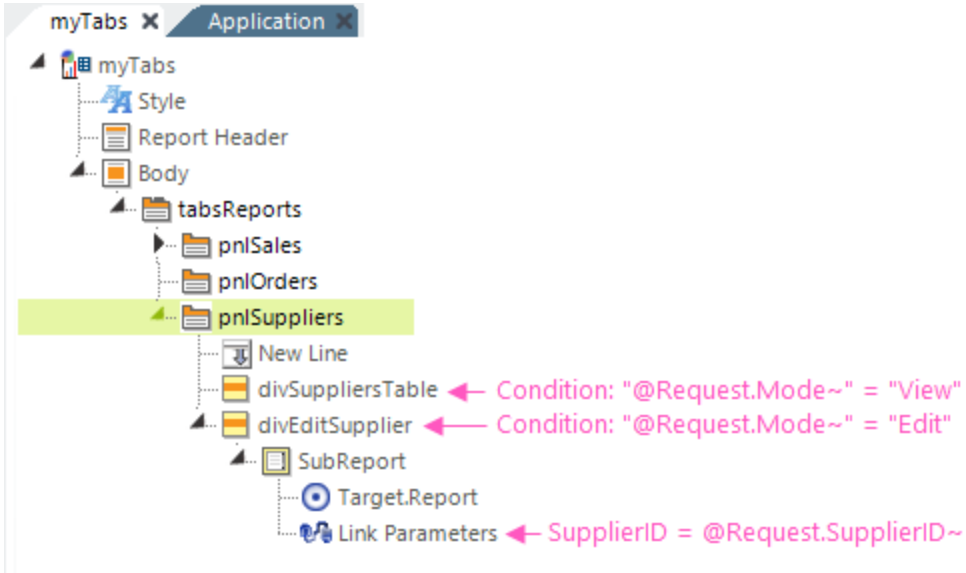

Sales
Orders
Suppliers
Suppliers Information
ID
CompanyName
ContactName
City
PostalCode
1
Exotic Liquids
Charlotte Cooper
London
EC1 4SD
2
New Orleans Cajun Delights
Shelley Burke
New Orleans
70117
3
Grandma Kelly's Homestead
Regina Murphy
Ann Arbor
48104
4
Tokyo Traders
Yoshi Nagase
Tokyo
100
5
Cooperativa de Quesos 'Las Cabras'
Antonio del Valle Saavedra
Oviedo
33007
6
Mayumi's
Mayumi Ohno
Osaka
545
7
Pavlova Ltd.
Ian Devling
Melbourne
3058

In "view" mode



In "edit" mode

Suppose we want a tab panel to show a Data Table and then be able to edit the data for a single row in that table, using a data entry form, in the same tab panel, as shown above.



To achieve this, we'll begin by placing two Division elements in the Tab Panel "pnlSuppliers", as shown above. The first division contains the elements that display the Data Table of supplier information. A column in this table, such as a SupplierID, is a link that refreshes the report, passing both the Supplier ID value and a variable indicating whether we're in "View" or "Edit" mode. This "mode" request variable is used in the divisions' Condition attribute to determine which division will be displayed.

The second division, as shown above, uses a **SubReport** element, in *Embedded* mode, to include a separate definition which contains the data entry form. The **Link Parameters** element is used to pass the request variable containing the Supplier ID to the subreport, so it knows which supplier record to look up and present in the data entry form.

Did you follow all that? It depends entirely on the fact that reports can call themselves ("refresh" themselves) and also pass request variables to the refreshed report in the process.

In summary, tabbed panels provide a useful, well-recognized, and convenient mechanism for arranging and displaying data without using separate report pages, and Logi Info provides an easy way to implement them.

Popup Panels

Logi Info includes the **Popup Panel** element, which opens a fully-functional, configurable window that can include Data Tables, charts, sub-reports, images, and other standard visualizations.

The following topis discuss the Popup Panel element, as well as the Tooltip Panel:

- Popup Panel Attributes
- Using a Popup Panel with a Link
- Using a Popup Panel with the onMouseOver Event
- Passing Parameters to a Popup Panel
- Data Entry in Popup Panels
- The Tooltip Panel

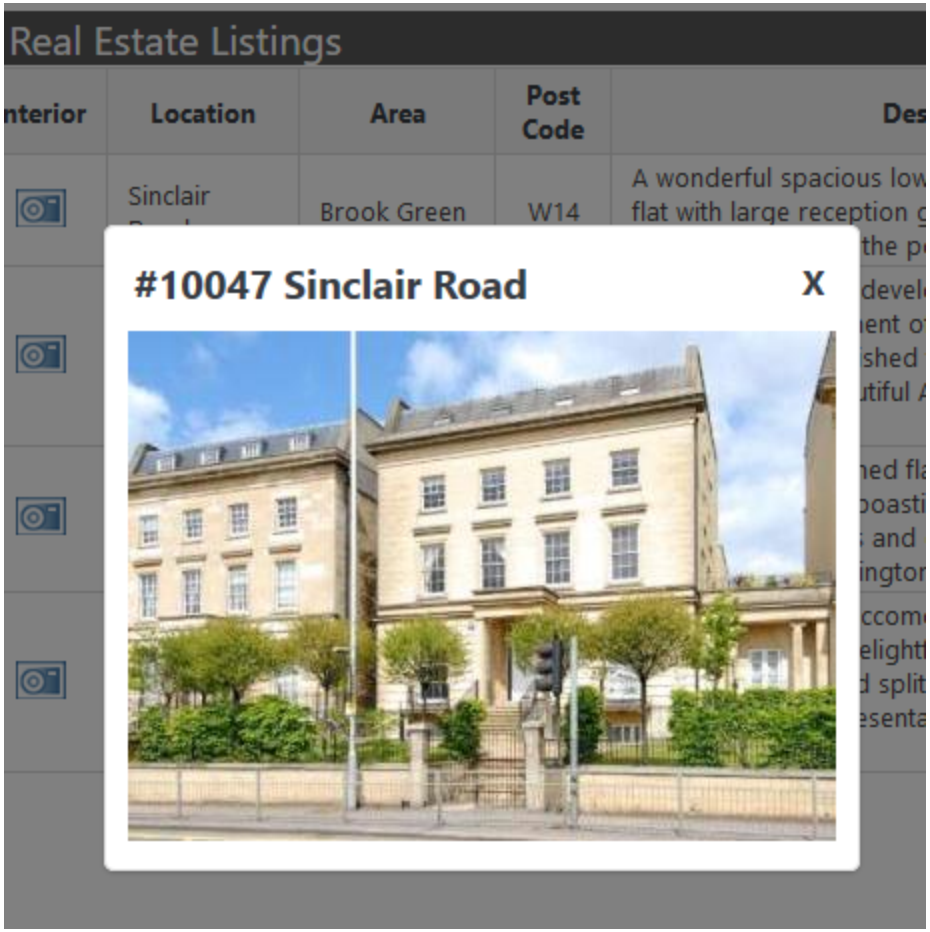
About the Popup Panel

A Popup Panel is a separate window that you can open "on top" of your existing report page. You can control its size, initial location, modal state, style, and, of course, its contents. It can contain almost every other element you would use in a regular report, with the exception of the super-elements (Dashboard, Analysis Grid, etc.). and a few others.

The visibility of a Popup Panel is controlled using the **Action.Show Element** element and therefore it can be shown or hidden when a user clicks a link, button, image, or data, or when a DHTML event occurs.



As you can see in the example above, Popup Panels include a **Title Bar**, where the element's **Caption** attribute value appears, and an optional"Close Button" that appears at the right end of the title bar.



Panels can either be displayed as *Modal* or *non-Modal*. A Modal window is one that requires users to interact with it before they can return to using the application. As shown above, when a Modal panel is displayed, the underlying page is disabled and shaded until the panel is closed.

The Popup Panel uses its own style classes to set colors, font sizes, spacing and other presentation values. These can be *over-ridden* by adding classes with the same names to your own style sheet. The panel's style sheet is:

```
<application folder>/rdTemplate/rdPopup/rdPopupPanel.css
```

In order for the style class override effect to work, your style sheet must be named in a Style Sheet element that is *lower* in your report definition's element tree than the Popup Panel element.

Popup Panel Attributes

The Popup Panel element has the following attributes:

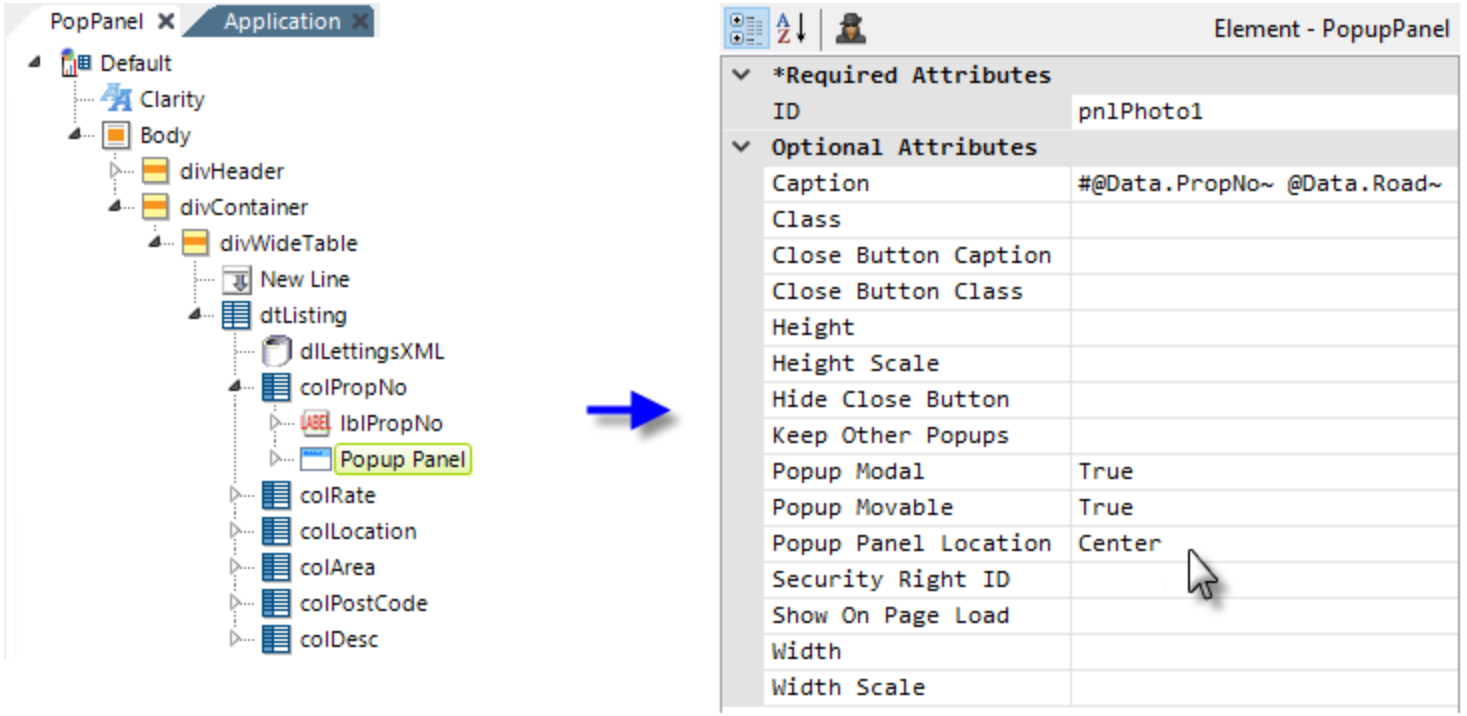
Attribute	Description
ID	(Required) Specifies a unique element identifier.
Caption	Specifies the text that will appear in the panel's Title Bar. You must provide a value here in order to display the Close button.
Class	Specifies the CSS class to be used by the element. When set, this class will also be used by all child elements that don't have their own class.
Close Button Caption	Specifies the "Close Button" text, which by default is an "X". This attribute allows you to specify different text instead, such as "Done". You must enter a value in the Caption attribute in order to display the Close button.
Close Button Class	Specifies an alternate class for the Close Button.
Height	Specifies the height of panel. If left blank, panel will automatically size to its contents.
Height Scale	Specifies the measurement units for the previous attribute: *Percent* or *Pixels*.
Hide Close Button	Specifies whether or not the Close Button will be displayed. Default: *False*

Attribute	Description
Keep Other Popups	Specifies behavior when one popup is already being shown and then a second popup is shown. The default behavior is to close the first popup. When this attribute is set to *True*, the first popup will remain open and subsequent popups will be layered in above it.
Popup Modal	Specifies if the panel will be shown in a Modal state, disabling the rest of the report page and shading it with a transparent gray overlay. Default: *False*
Popup Movable	Specifies whether the panel to be dragged and repositioned, using the mouse. Default: *False*.
Popup Panel Location	Specifies the location at which the panel will be shown. Choices include: *Mouse*- panel will appear below and to the right of the cursor position. *Center* - panel will appear at the center of page. *200,150* - panel will appear at these X,Y coordinates, relative to the upper left corner of the screen. Default: *Mouse*
Security Right ID	When using Logi Security, specifies one or more Security Right IDs, separated by commas. Users with these Security Right IDs will be able to view the Analysis Grid, users without them will not.
Show On Page Load	Specifies whether the panel will be automatically displayed when the report page is first loaded. Default: *False*

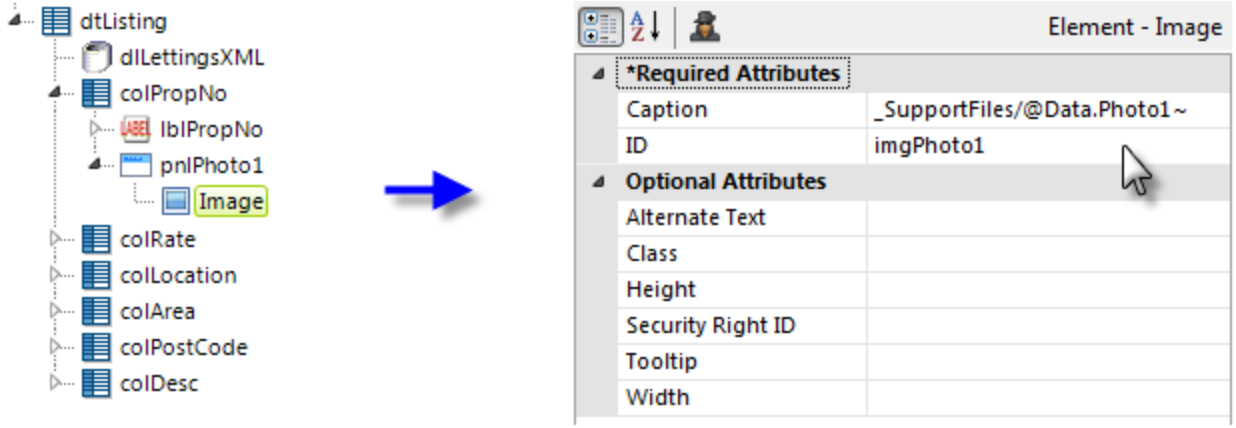
Attribute	Description
Width	Specifies the width of the panel. If left blank, panel will automatically size to its contents.
Width Scale	Specifies the measurement units applied to previous attribute: *Percent* or *Pixels*.

Using a Popup Panel with a Link

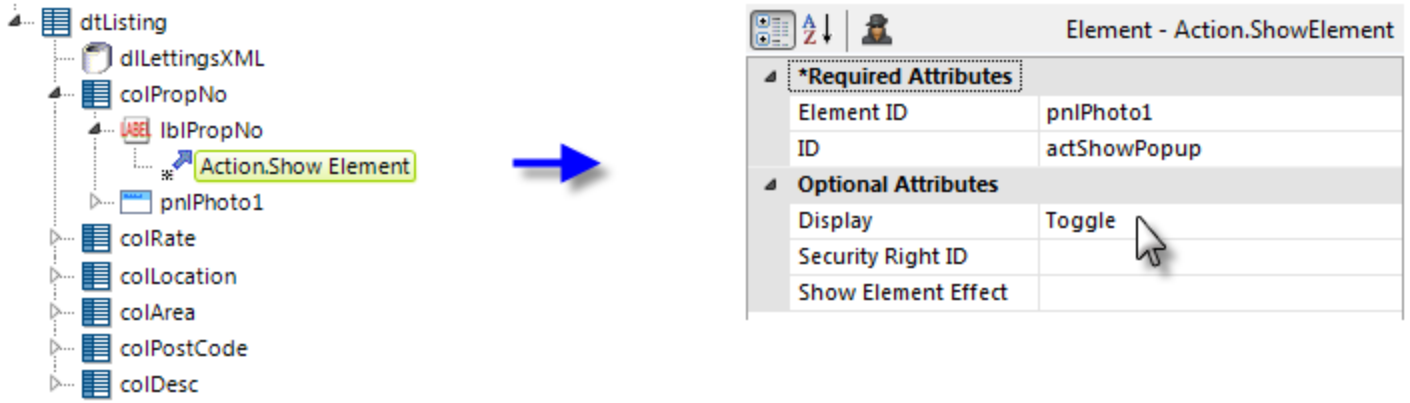
Adding a Popup Panel to your report definition is simple and straightforward:



In the example above, a **Popup Panel** element (named "pnlPhoto1") has been added to the definition in a Data Table column. Its attributes have been set to control its behavior. The panel's caption will display data from the datalayer, through the use of @Data tokens, which is why the Popup Pane has been placed as a child-element beneath a Data Table Column element. Otherwise it could have been placed anywhere in the definition.



Next, below the Popup Panel element, an **Image** element has been added, as shown above, and its **Caption** attribute set to data returned from the datalayer that specifies the image filename.



Finally, a link has been created, which shows or hides the Popup Panel, by adding an **Action.Show Element** element to the **Label** element that displays the data in the report's Property Number column. Its attributes are set to make it act upon the Popup Panel ("pnlPhoto1") and to cause the panel's visibility to *toggle*, or alternate, between shown and hidden.

London Flats to Let

11/10/2015

Prop #	Weekly	Location	Area	Post Code	Description
10047	£575	Sinclair Road	Brook Green	W14	A wonderful spacious lower ground two bedroomed flat with large reception good sized rooms and garden, located on the popular Sinclair Road.
10048	£550	Goldhawk Road	Stamford Brook	W6	Situated in a luxury development, this fabulous two bedroomed apartment offers spacious accommodation finished to the highest possible standards with beautiful Amtico wood floor and secure parking.
10049	£525	Blythe Road	Brook Green	W14	Smart two bedroomed flat, benefitting from recent refurbishment and boasting a large reception room, spacious bedrooms and great access to the amenities of High Street Kensington.
10050	£500	Kings Street	Ravenscourt Park	W6	Offering spacious accomodation throughout and benefiting from a delightful south-facing roof terrace, this two bedroomed split-level apartment boasts excellent neutral presentation, private entrance and garage

London Flats to Let

11/10/2015

Prop #	Weekly	Location	Area	Post Code	Description
10047	£575	Sinclair Road	Brook Green	W14	A wonderful spacious lower ground two bedroomed flat with large reception good sized rooms and garden, located on the popular Sinclair Road.
10048	£550	Goldhawk Road	Stamford Brook	W6	Situated in a luxury development, this fabulous two bedroomed apartment offers spacious accommodation finished to the highest possible standards or and secure parking.
10049	£525	Blythe R			nefitting from recent refurbishment and m, spacious bedrooms and great access to ensington.
10050	£500	Kings St			on throughout and benefiting from a delightful two bedroomed split-level apartment boasts private entrance and garage

#10048 Goldhawk Road

X

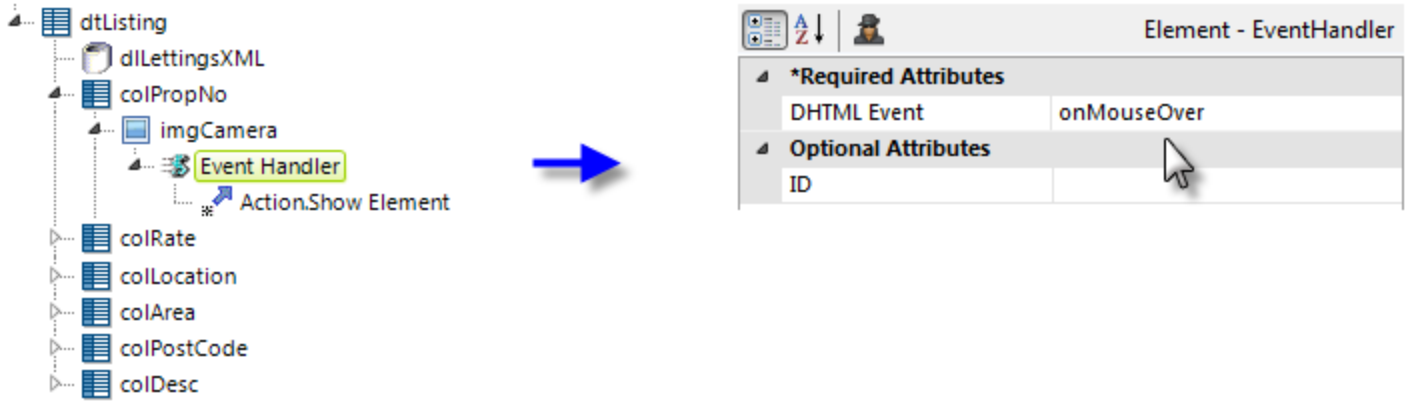


The examples above show the Data Table and what happens when one of the Property Number links is clicked. The report is disabled when the panel is displayed and the background is "grayed" out because the Popup Panel has been set to *Modal*.

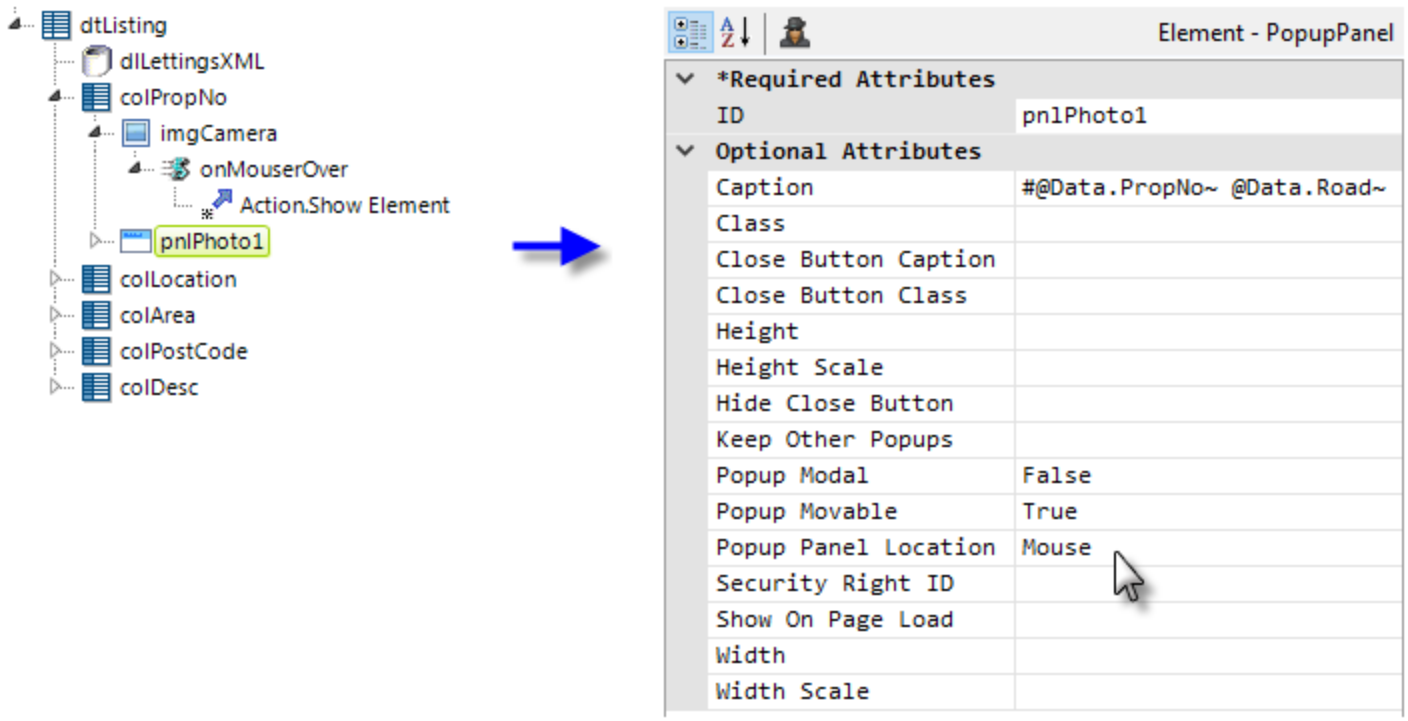
 When you place a popup panel in a Data Table column, an instance of the panel will be generated for *every* row in the Data Table. This could have performance implications when using large data sets. Another technique, placing the Popup Panel element *outside* of the Data Table, is discussed in "Passing Parameters to a Popup Panel" on page 745.

Using a Popup Panel with the onMouseOver Event

This example changes the definition in the previous example slightly so that the Popup Panel will appear when the mouse *hovers* over an icon in a Data Table column:



Starting with the example from "Using a Popup Panel with a Link" on page 736, in the definition above the Property Number data has been replaced with an **Image** element displaying a camera icon, with an **Event Handler** element below it. The Event Handler has been set, as shown, to fire when the *onMouseOver* event occurs. The **Action.Show Element** element from the previous example has been placed below the Event Handler.



For a little variety, a few changes have also been made to the Popup Panel's attributes. The panel is no longer Modal and its pop-up location has been set to be near the mouse cursor.

London Flats to Let

11/10/2015

Prop #	Weekly	Location	Area	Post Code	Description
	£575	Sinclair Road	Brook Green	W14	A wonderful spacious lower ground two bedroomed flat with large reception good sized rooms and garden, located on the popular Sinclair Road.
	£550	Goldhawk Road	Stamford Brook	W6	Situated in a luxury development, this fabulous two bedroomed apartment offers spacious accommodation finished to the highest possible standards with beautiful Amtico wood floor and secure parking.
	£525	Blythe Road	Brook Green	W14	Smart two bedroomed flat, benefitting from recent refurbishment and boasting a large reception room, spacious bedrooms and great access to enities of High Street Kensington.
					g spacious accomodation throughout and benefiting from a delightful acing roof terrace, this two bedroomed split-level apartment boasts nt neutral presentation, private entrance and garage

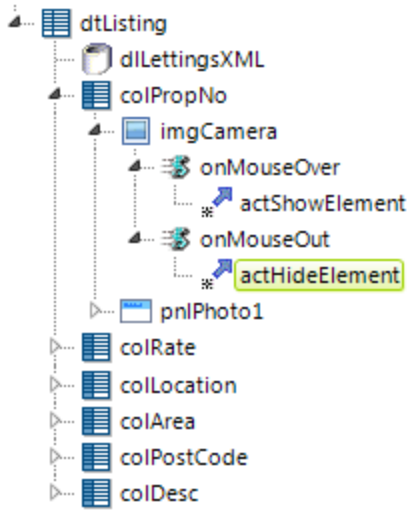
#10049 Blythe Road

X



As shown above, when the cursor passes over the camera icon, the Popup Panel with its image appears. Because the panel isn't Modal, the cursor can continue to be moved over the other icons in turn.

This example works as expected but still requires you to click the Close Button to hide the last photo displayed. Let's make the pop-up panel close when the mouse leaves the icon. To do this, we'll use the "dual Event Handler" technique:



In the example above, a second Event Handler element has been added beneath the Image element, set to fire on "onMouseOut" events. An copy of the Action.Show Element element we used to show the element is added beneath the second Event Handler (but here we're given it an ID of "actHideElement"). Because the Action elements use *Toggle* as their Display mode, one will show the pop-up and the other will hide it.

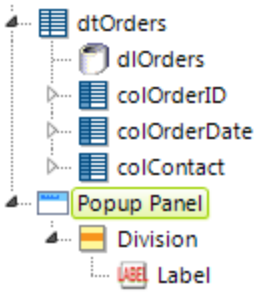
The only thing left to do is to set the Popup Panel element's **Hide Close Button** attribute to *True*.

Passing Parameters to a Popup Panel

As shown in previous examples, a common use of the Popup Panel is to provide drill-down into the details of data that's presented in a Data Table. Placing a Popup Panel element in a Data Table column will result in a pop-up being generated for *every* table row, which could have a negative performance impact.

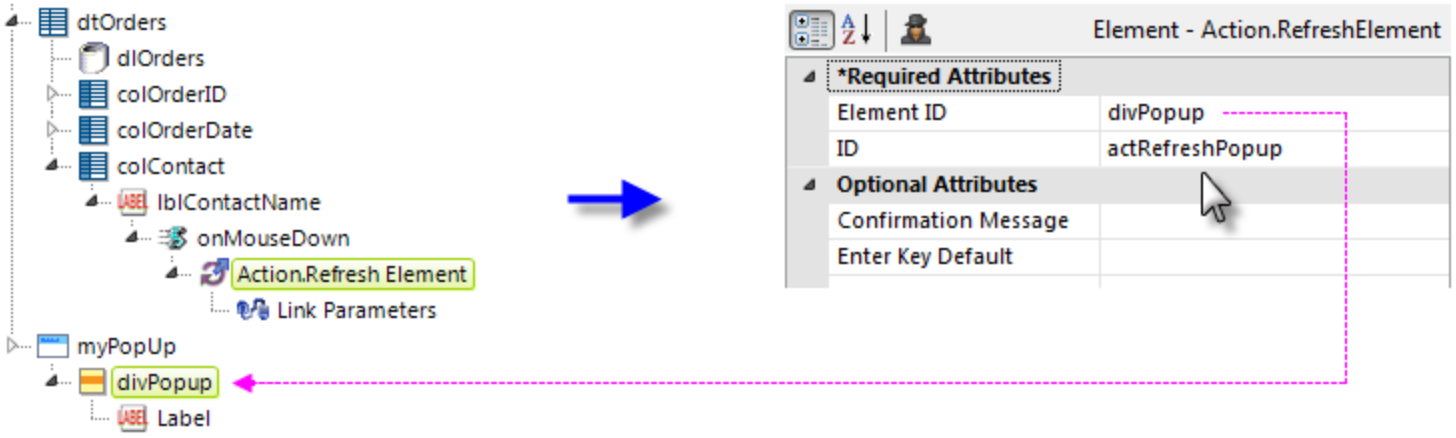
An alternate approach is to place the Popup Panel *outside* of the Data Table and pass parameters to it, so that it can do its own lookup and then present detail data. We'll use two Event Handlers again and make use of the fact that, even though the hidden pop-up panel has been generated (once when the report is first shown), we can use AJAX technology to "re-draw" the hidden pop-up when a link in the Data Table is clicked. The fact the panel isn't visible at the time doesn't matter and, right after it's re-drawn, we'll show it.

An important detail is that the DHTML Events scheme recognizes the pressing of a mouse button (*MouseDown*) as distinct from when the button bottoms out (*Click*). It also recognizes the button coming back up after a click (*MouseUp*) but we don't need to use that event right now.



Consider the example above. A Popup Panel has been added to a report, but this time it's *not* under a Data Table column. The first child element of the panel is a **Division**, and below it a **Label**. In this example, the label is just a way to verify that the parameter

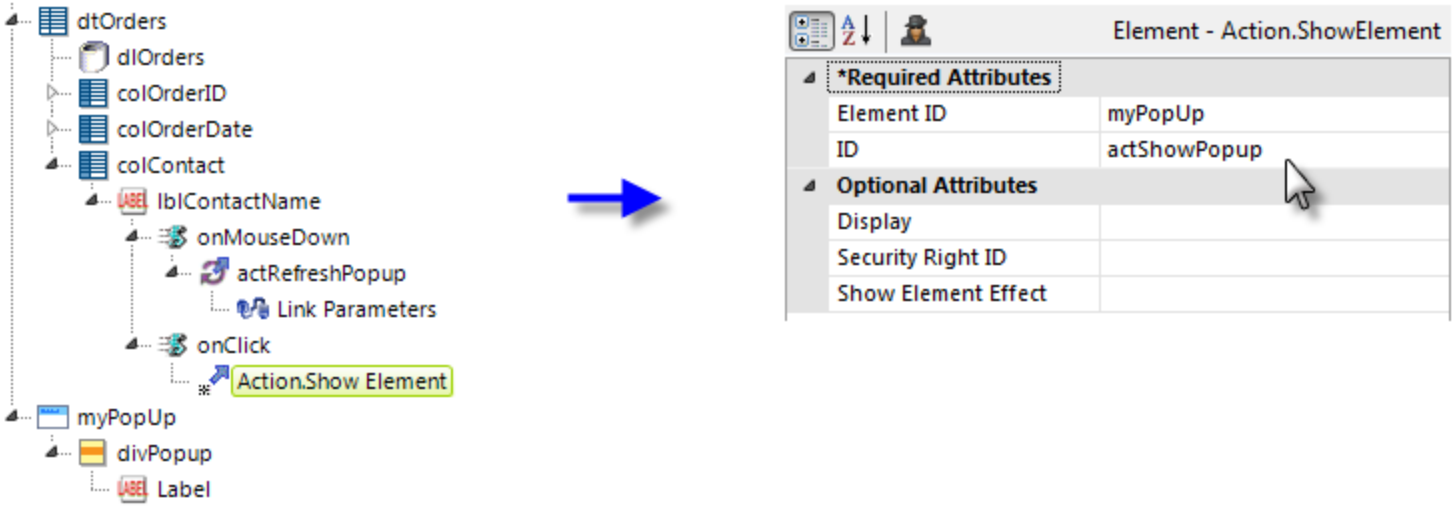
we'll pass has been received; the parameter will be available as a normal @Request token. The Division has no attributes set other than its unique element **ID** (which *is* important).



Now, as shown above, an **Event Handler** element has been added beneath the **Label** ("lblContactName") used to display data in a Data Table column. Its **DHTML Event** attribute value is set to *onMouseDown*.

Beneath it, an **Action.Refresh Element** element has been added, and its attributes have been set as shown. When the mouse button is pressed down, the division in the hidden panel will be refreshed, using AJAX.

The Action.Refresh Element element can have a **Link Parameters** child element; its values are "received" in the pop-up as @Request tokens when the pop-up panel is refreshed. If we use an @Data token as the parameter value, we can pass data specific to the row containing the link into the pop-up. In a scenario where you want to show detail data in the pop-up, the Link Param can send identifying data (like an OrderID) into it. Then elements in the pop-up might use that OrderID to look up and display detail data in the pop-up.



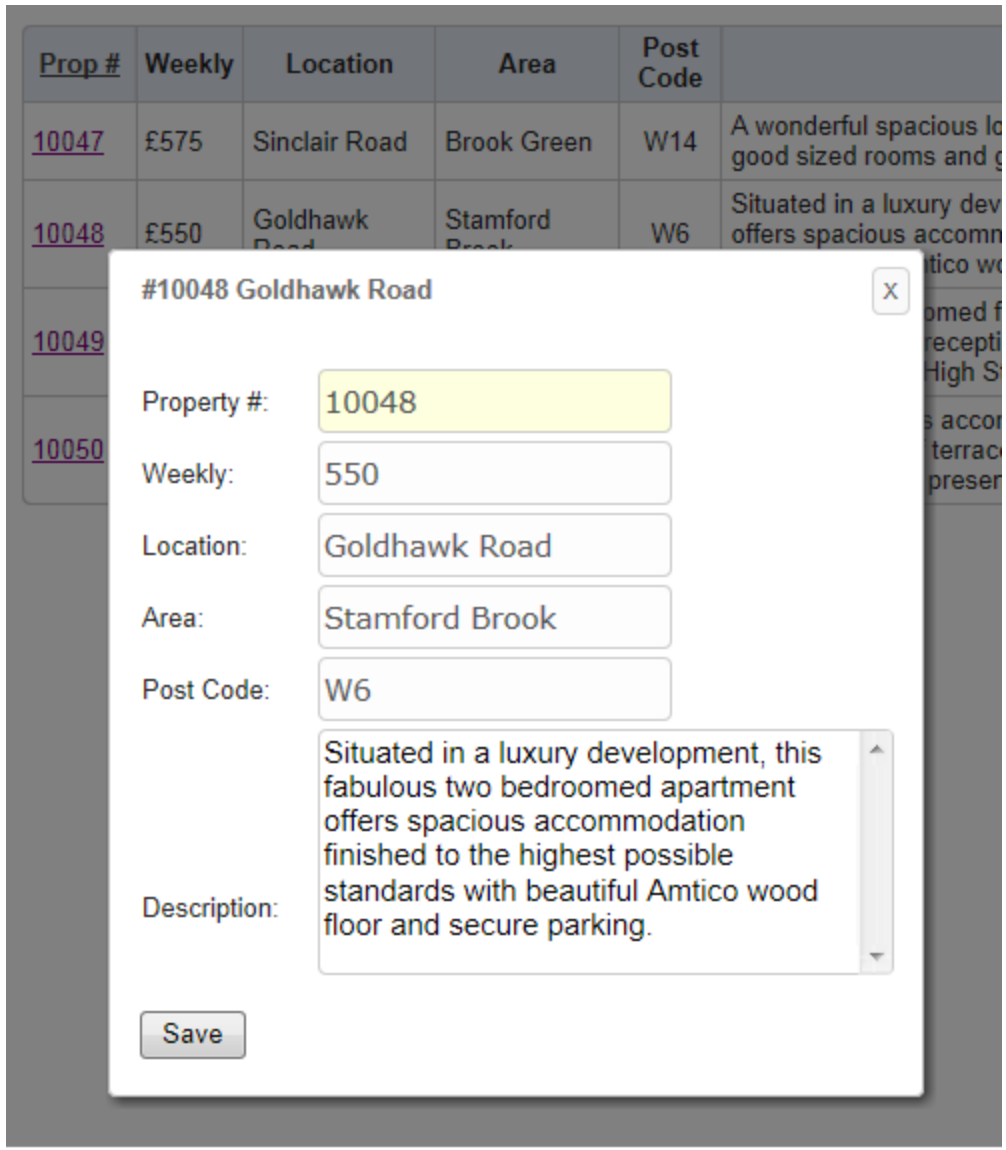
Finally, we've added a second Event Handler element, as shown above, beneath the Label in the Data Table column. Its DHTML Event attribute value is set to *onClick*.

Beneath it, an **Action.Show Element** element has been added, and its attributes have been set as shown. This element will cause the popup panel to be shown.

So, to review: the Label element that users will click has *two* Event Handlers beneath it. The first one will pass a request variable to the pop-up panel and refresh it; the second one will cause the refreshed pop-up panel to be shown.

Data Entry in Popup Panels

One common use of pop-up panels is to provide a data entry form for Data Tables, like this:


Prop #
Weekly
Location
Area
Post Code
10047
£575
Sinclair Road
Brook Green
W14
10048
£550
Goldhawk
Stamford
W6
10049
10050
#10048 Goldhawk Road
X
Property #:
10048
Weekly:
550
Location:
Goldhawk Road
Area:
Stamford Brook
Post Code:
W6
Description:
Situated in a luxury development, this fabulous two bedroomed apartment offers spacious accommodation finished to the highest possible standards with beautiful Amtico wood floor and secure parking.
Save

Typically, you'd use a Process task to update the data when the Save button is clicked. You'd expect that the user input values would be passed to the task as Request variables, as usual.

However, it's important to realize that the @Request token IDs will automatically have a row designator ("_Row1") appended to them, because the Input controls are within a Data Table row. This presents a complication: how will the task "know" which row these values refer to and therefore which tokens to use? For example,

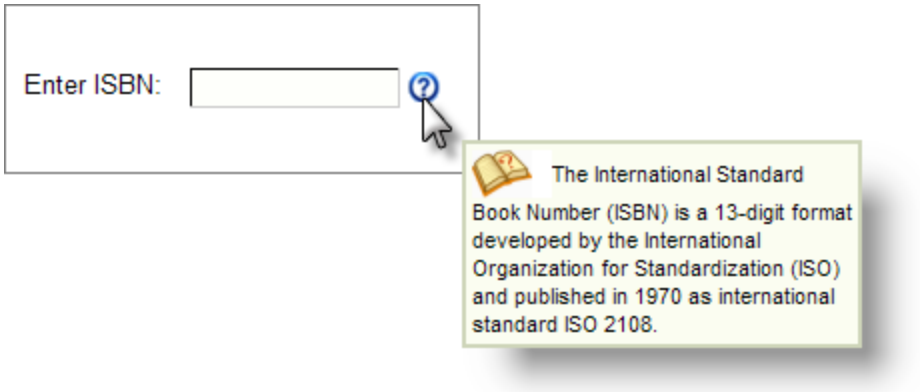
`@Request.inpPropertyNo_Row1~`, or `@Request.inpPropertyNo_Row2~`, or ?

The recommended technique for handling this is to use **Input Hidden** elements, one for each user input element in the pop-up panel, placed *outside* the Data Table. When the Save button is clicked, an *onMouseDown* Event Handler is used with Action.JavaScript to call code that assigns the values from the pop-up panel's Input elements to the Input Hidden elements. An additional Action.Process element under the Save button then calls the Process task, which uses the @Request tokens for the Input Hidden elements to update the data.

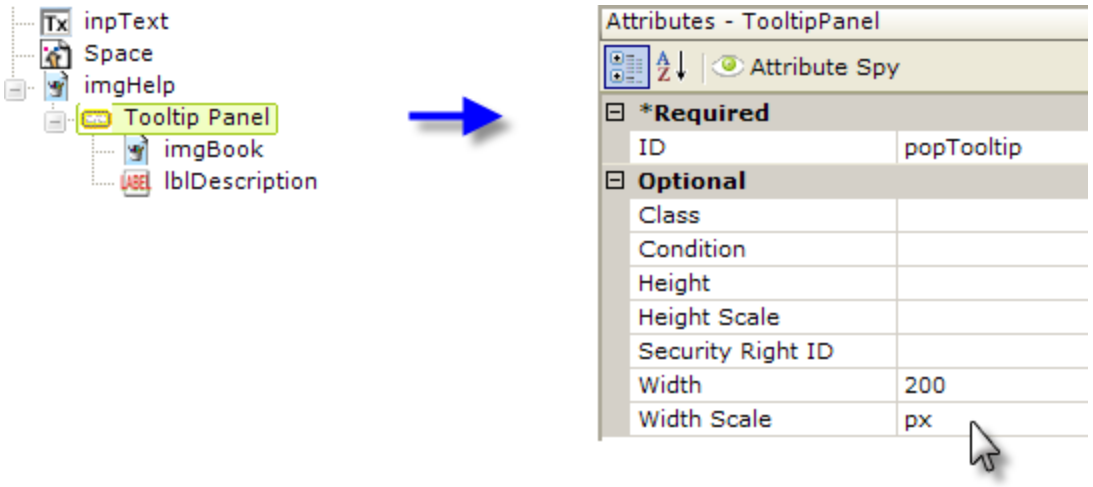
You can see this in action and view sample definitions in the Samples pageon DevNet.

The Tooltip Panel

Another variety of popup panel is available: the **Tooltip Panel**. Like its namesake, the traditional "tooltip", the Tooltip Panel element is used to supply additional information to the user, and it appears when a user hovers their mouse over its parent element.



However, as shown above, this element is much more versatile than a regular tooltip, allowing you to control its size and include a range of other elements within it, including Labels, Images, and even Charts. The panel will be displayed just below and to the right of the mouse position.



The definition snippet above shows how the Tooltip Panel element is used; it can be the child of a variety of other elements. No Event Handler element is required; the panel will appear and disappear automatically, based on mouse movements.

You can use an Event Handler *in addition* to a Tooltip Panel, but if you assign the *onMouseOver* event to the Event Handler, the Tooltip Panel will *not* work. Assigning other events, such as *onClick*, to the handler will not interfere with the Tooltip Panel.

The Tooltip Panel element has a **Condition** attribute, allowing you to control if, and when, it will appear.

ReportCenter Menu

Logi Info developers often have a requirement to provide a menu of the reports in their Logi application. The **ReportCenter** family of elements provides an easy and flexible way to implement such a menu.

The following topics discuss the ReportCenter Menu elements:

- Including Reports in the Menu
- Creating the Menu
- Scheduling Report Distribution

About the ReportCenter Elements

The **ReportCenter** elements provide a simple way to create a menu of report links that can be included in any report definition. They also provide a way to display and manage bookmarks (see *Bookmarks*), without needing to build these elements separately. And, finally, they provide a quick and easy way to create and manage email-based report distribution schedules without the need to build separate elements or use process tasks.



An example of a typical menu is shown above. Menu items can be presented as:

1. Individual **Report** links
2. Report links grouped beneath a **menufolder**
3. **Saved Bookmark** links - developers can enable option link icons to allow users to manage bookmarks and/or schedule them for delivery via email, at runtime.

Reports can be run in the current, or in a different, browser window and individual menu items can be dynamically shown or hidden in the menu, based on tokens and security rights.

As an element, the **ReportCenter Menu** sits within a report definition that you create, giving you the flexibility to position it and style it to match the surrounding page appearance, as desired.

The ReportCenter Menu *cannot* provide a menu of reports and bookmarks available in *multiple* Logi applications; its scope is limited to the Logi application in which the ReportCenter Menu element resides.

The menu background and border colors, border style, fonts, and other appearance-related qualities can be specified using container elements, themes, and/or style classes.

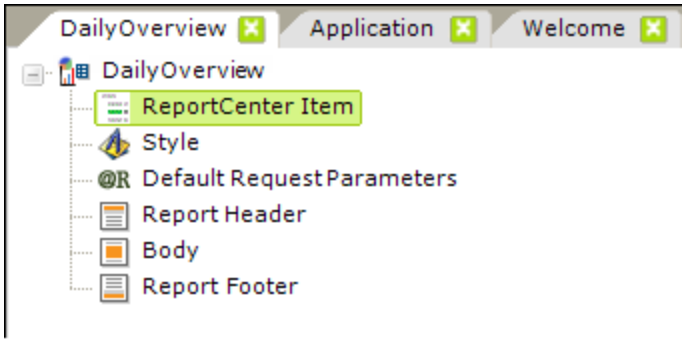
A ReportCenter Menu Sample Application is available on DevNet and can be reviewed for more implementation examples. The following sections illustrate how the ReportCenter elements are used.

Other Menu Variants

Logi Info also includes other menu elements you may wish to investigate; they may be more useful depending on the nature of your application. Pop-up menus display a panel with menu options and the Menu Tree menu, introduced in v12.1, can be used to create horizontal and vertical hierarchical menus. For more information see "Popup Menus" on page 768 and "Menu Tree Menus" on page 776.

Including Reports in the Menu

Including a report definition in a menu is very easy: it starts by "tagging" it to be in the menu:



This is done by adding a **ReportCenter Item** element, as shown above, to a report definition.

The actual *text* of the menu item link for a report will be either a) the root element's **Caption** attribute value or, if that's blank, b) the report definition filename (without its file extension). Using the Caption attribute lets you introduce *spaces* in the link text, which can't be included in a definition filename.

You don't have to do anything else and the report will be automatically included in the menu. However, the ReportCenter Item element has the following optional attributes, which can be used to provide additional functionality:

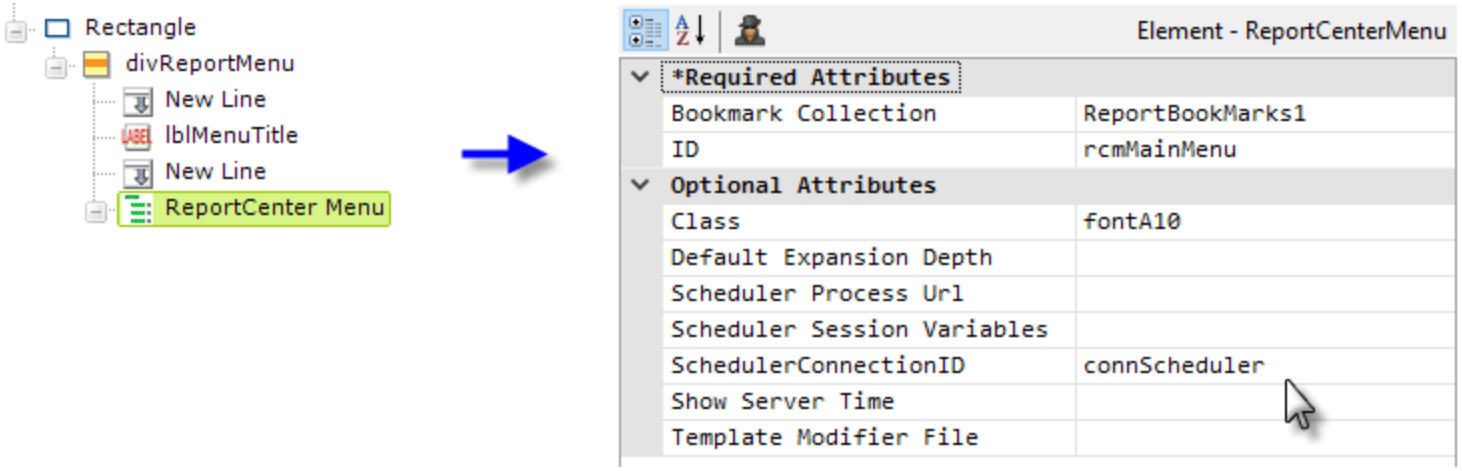
Attribute	Description
ReportCenter Allow Schedule	When set to *True*, a clock icon will appear next to menu links for report bookmarks. This icon allows the user to schedule the report to be run and distributed via email. Used when Include Bookmarks attribute (see below) is *True*, and the Logi Scheduler has been installed. Scheduling is only available for report bookmarks, not for reports. Default value: *False*

Attribute	Description
ReportCenter Allowed Export Formats	Specifies the formatting options, *HTML*, *PDF*, and/or *NativeExcel*, that will be available to the user when scheduling report bookmarks for email distribution at runtime. Multiple formats can be specified in a comma-separated list. These options appear in a selection list in the scheduling pop-up panel. Default value: *blank* - provides all three formats.
ReportCenter Exclude	Allows the developer to programmatically control the visibility of this menu item, by using tokens in this attribute value which resolve to *True* (Hidden) or *False* (Visible).
ReportCenter Export Table ID	Used when bookmarks for reports that contain multiple Data Tables are made available for scheduled distribution. Specifies the element ID of the Data Table to be exported. Information outside of the Data Table, such as headers and footers, will *not* be included in the export. Default value: *blank* - assumes there is only one Data Table. To export the data in an Analysis Grid element, set the value to *dtAnalysisGrid*
ReportCenter Folder	Specifies the text for the **menu folderitem** that the current report link will appear under. Creates the folder item if it doesn't exist. Folders can have multiple levels of depth. To create multiple levels, separate each folder name with a the "pipe" (|) symbol. For example, to create a "Sales" folder under an "Accounting" folder, enter *Accounting|Sales* Default value: *blank* - no folder
ReportCenter Frame ID	Sets the target frame for display of the report when the menu item is clicked. Options include: *_blank* - displays report in a new window *_parent* - displays report in the parent window, useful when the ReportCenter menu is inside an Include

Attribute	Description
	Frame. *_top* - displays the report in the top-most window. You may also specify an existing Frame ID to re-use the same window for each request. Default value: *blank* for the current browser window
ReportCenter Include Book-marks	Controls the appearance of savedbookmarks for the current report in the ReportCenter Menu. Default value: *False*

Creating a ReportCenter Menu

Once we've tagged all of the report definitions for the menu, we can build the actual menu:



This is done by adding a **ReportCenter Menu** element to the report definition that presents the menu.

The elements used to create the menu example shown at the top of this document are shown above. It was created by using **Rectangle** and **Division** elements to contain the menu, with style applied to the Rectangle to provide a border, background color, and padding. A separate **Label** element was used to provide the menu title. This is, of course, just one of many possible ways of presenting the menu.

The ReportCenter Menu element's attributes are shown above, right and have been configured as follows:

Attribute	Description
Bookmark Collection	(Required) Saved bookmark collection name, e.g. *myBookmarks1*. If bookmarks have been created on a per-user basis, using a GUID or a user name, identify the collection with a token here. *Note*: The path to the folder where bookmarks have been saved must be configured in the **General** element's **Bookmark Location** attribute in the _Settings definition. Bookmarks can also be stored in a database, see *Storing Bookmark, Gallery, and Save Files in a Database*.
ID	(Required) A unique identifier for the element
Class	Specifies the style class for the menu report links. Menu *folder* item font is specified by overriding the `.rdFolder` class, in your stylesheet.
Default Expansion Depth	Specifies which level of the menu folders items will be expanded by default. *Blank* or *0* = no expansion, or enter a number such as *1* or *2*.
Scheduler Process Url	Specifies the full URL of the Logi Info application that will run the scheduled task. The URL should specify the folder that contains the rdProcess.aspx file. For example: `http://myWebServer/myApplication` The default value is the URL used to access the current web application.
Scheduler Connection	If users will be allowed to schedule reports for email distribution, this attribute specifies the ID of the **Connection.Scheduler** element configured in the _Settings definition (a pull-down list of connections is

Attribute	Description
ID	provided).
Scheduler Session Variables	Specifies a comma-separated list of Session variables to be passed by the Logi Scheduler to a scheduled task during execution.
Show Server Time	Specifies whether the Scheduler's user interface should display time based on the server's current time. This can be useful when the application user's Time Zone differs from the server's zone. The user can infer and adjust the time for running scheduled reports. Default value is *False*.
Template Modifier File	Specifies the name, with file extension, of a *Template Modifier Files* to be applied to the menu at runtime. This is an XML file that can be used to change the underlying template used to create the menu. Template Modifier files can be in any folder accessible to the application; if a folder isn't specified, the `_SupportFiles` folder is assumed. You can also use this attribute to set a default Time Zone for Logi Scheduler.

Nothing more is needed for the basic menu; running the definition now will cause all of the other report definitions tagged with ReportCenter Item elements to appear in the menu.

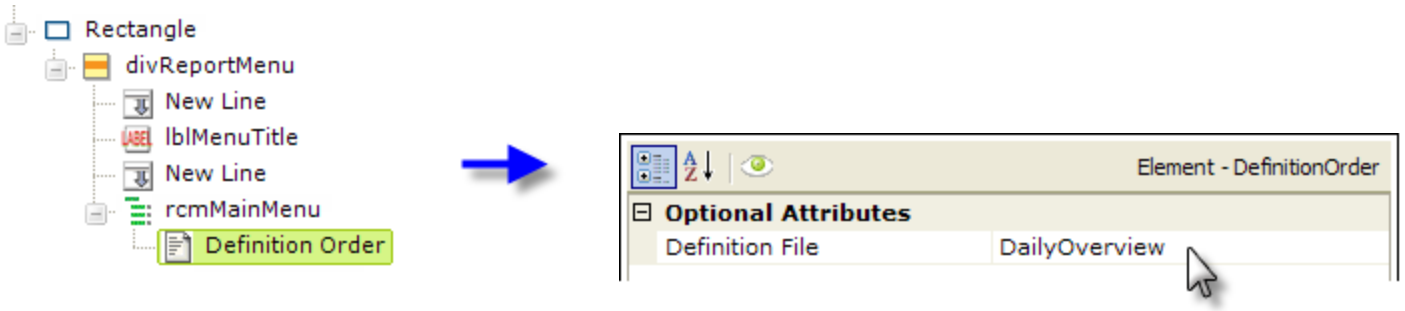
Controlling the Report Links Dynamically

The ReportCenter Menu element creates its own internal *DataLayer.Definition List* element and filters it based on the tagged definitions. However, if you want more control over the included report definitions, you can override that internal datalayer by adding your own DataLayer.Definition List element beneath the ReportCenter Menu element and filtering it as desired.

This datalayer type can retrieve root element "custom attributes" along with the other definition information; these are attributes you enter *manually* by editing the source of a definition's root **Report** element. You might use a custom attribute, for example, to indicate whether definitions are to be included in the menu or not, then apply filtering to the datalayer based on the custom attribute's column value.

Ordering Menu Items

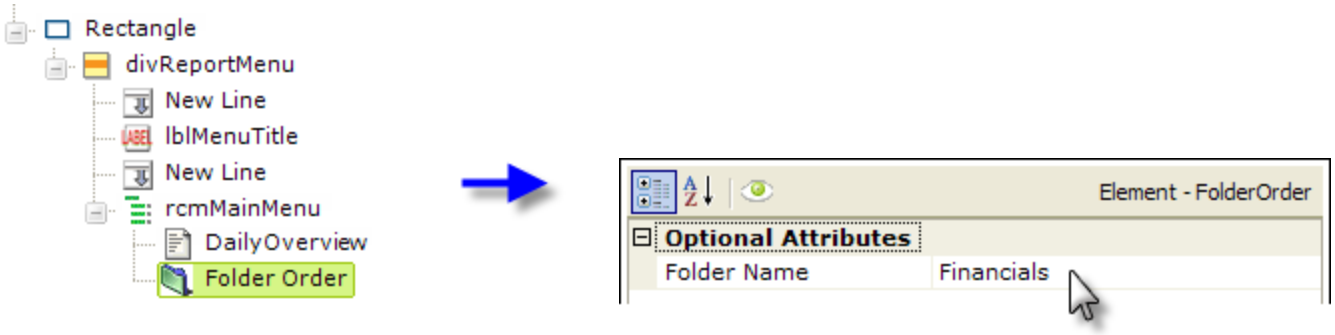
The menu items in a basic menu will normally be ordered *alphabetically* by their menu link text. However, if desired, you can rearrange the order of the menu items.



This is done by adding one or more **Definition Order** elements beneath the ReportCenter Menu element, as shown above, and assigning a report definition to them. The menu items will then appear in the top-to-bottom order of the Definition Order elements. Any remaining menu items not represented by a Definition Order element will be shown in alphabetical order in the menu, *below* those items that are represented by one.

Ordering Menu Folder Items

As described earlier, the ReportCenter Item element allows you to create folder items in the menu. In a manner similar to controlling the arrangement of menu items, you can also control the order of these folders within the menu items.

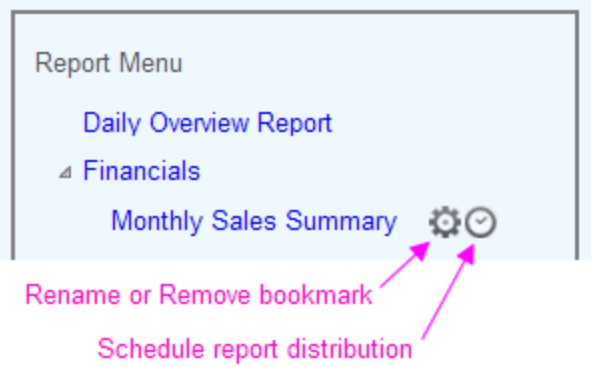


This is done by adding one or more **Folder Order** elements beneath the ReportCenter Menu element, as shown above, and assigning a report definition to them. The Folder Name attribute value must *exactlymatch* the ReportCenter Item element's ReportCenter Folder value. The menu folder items will then appear in the top-to-bottom order of the Definition Order and Folder Order elements. Any remaining menu items not represented by a Definition Order or Folder Order element will be shown in alphabetical order in the menu, below those items that are represented.

Additional child Definition Order and Folder Order elements can be placed beneath a Folder Order element in order to control the arrangement of child menu items and folders displayed when a menu folder is expanded.

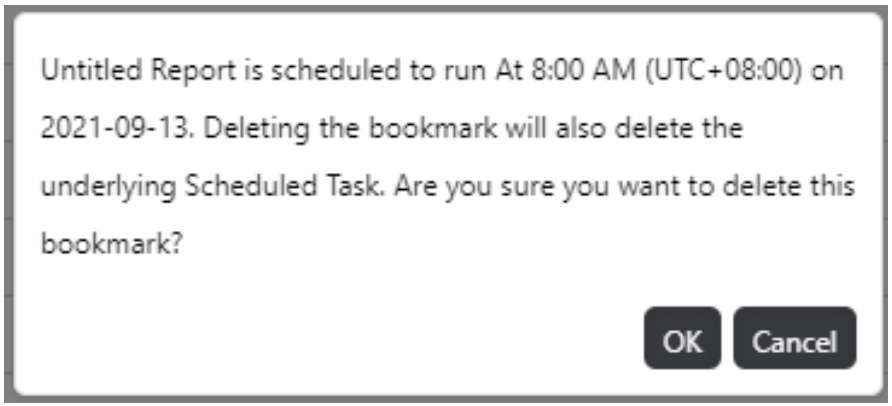
Scheduling Report Distribution

When report bookmarks are being used and the appropriate ReportCenter Item attributes have been set to allow scheduling, two icons will appear at the end of each bookmark menu link:



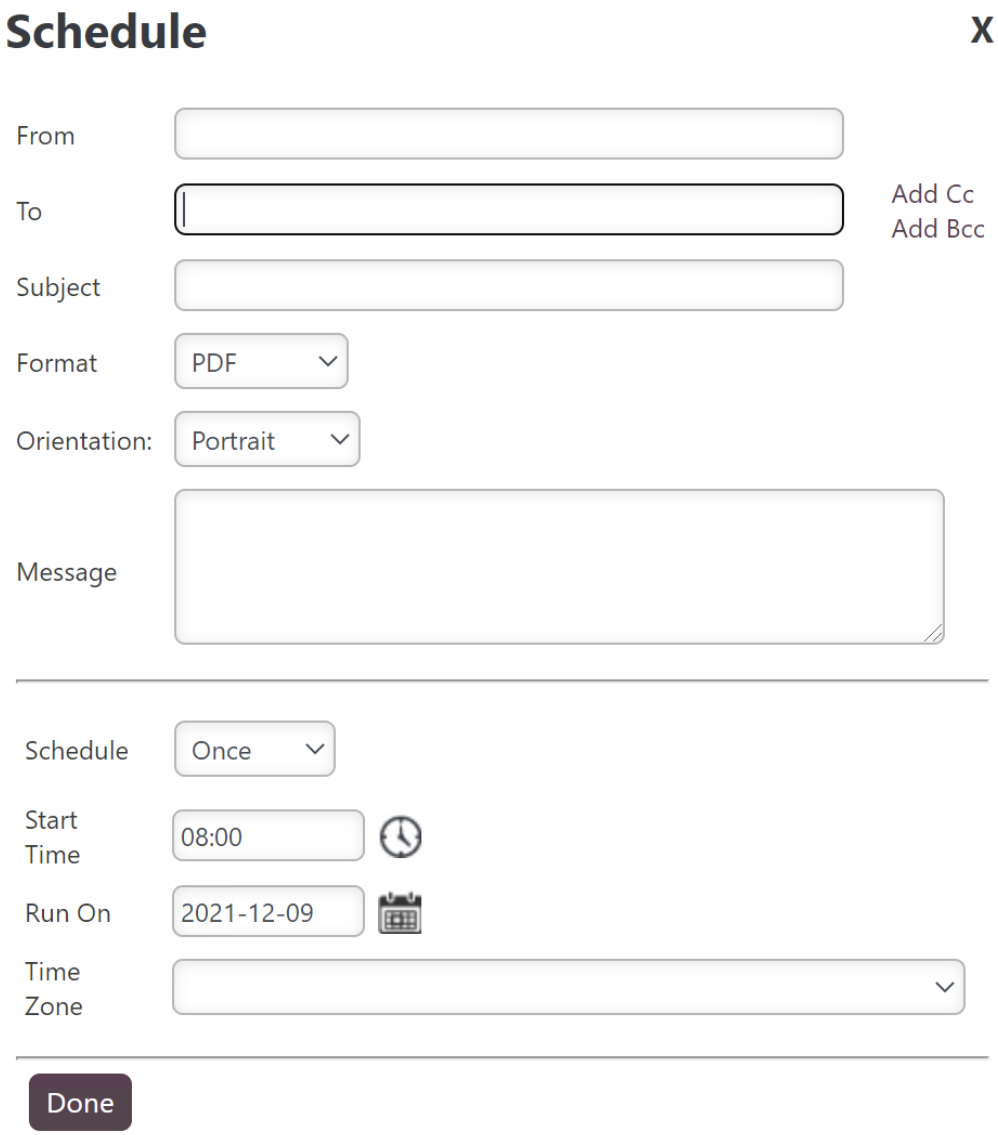
These icons allow users to **rename** or **remove** the bookmark, or **schedule** report distribution, as shown above.

Upon removing a bookmark, Info generates a confirmation dialog box that lists the time and date the bookmark is set to run and asks if you still want to delete the bookmark:



To proceed, select **OK**. Otherwise, select **Cancel**.

When the **Schedule** icon is selected, the modal dialog box shown below, is displayed and the user can fill-in appropriate information:



Multiple addresses can be entered in the "To:" field by separating them with commas or semi-colons.

Report developers can set a default time zone for reports using TMFs.

The child element **Email List** can be used with ReportCenter Menu to make it easier for a user to enter email addresses by showing an auto-complete list for the To: text box in the form shown above. The email addresses come from a column ina datalayer added beneath Email List. Specify the name of that column in the Email List element's **Email Column Name** attribute.

At the designated time, the Logi Scheduler will run the report, export it in the specified format and attach it to an email (for PDF and Excel formats) or embed it in an email (Web Mail format), and send the email using the first (top-most) **Connection.SMTP** element found in the _Settings definition. This enables an administrator to switch SMTP servers without having to change report schedules. The developer, of course, controls which export formats are available using a ReportCenter Item attribute.

If it's supported by your browser, an **HTML5** feature called "local storage" will be used to automatically store the email addresses you enter, preserve them between sessions, and provide them back to you in a list for selection in subsequent sessions. If your browser doesn't support this feature, or if it's turned off in your browser options, obviously the element won't be able to provide the list of addresses for re-use.

Popup Menus

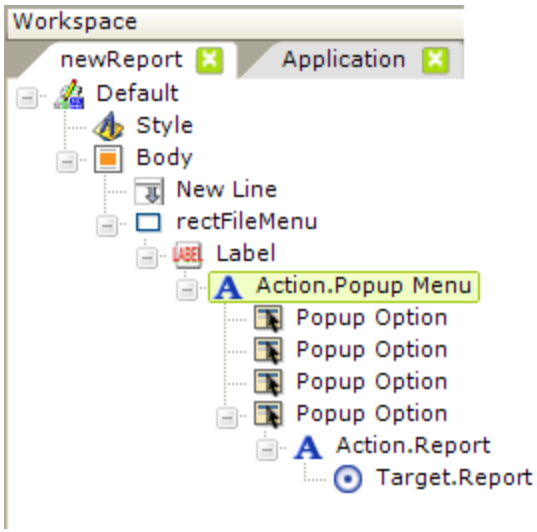
Logi Info includes elements that allow developers to display **pop-up menus**, offering action choices to the user. Each choice can be custom-defined, with its appearance controlled using style classes, and can have a specific action assigned to it.

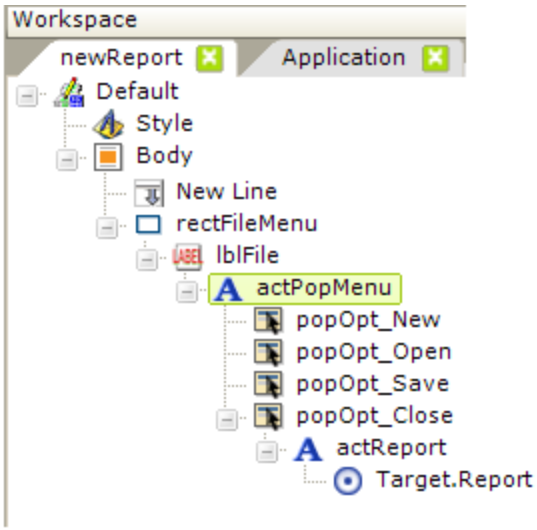
The following topics discuss the use of the pop-up menu element:

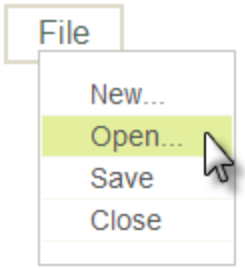
- Adding an Image to the Pop-up Options
- Using the onMouseOver Event
- Controlling Pop-up Location
- Controlling Pop-up Appearance

About Pop-up Menus

Two elements, **Action.Popup Menu** and **Popup Option**, produce the desired pop-up menu when a related link is clicked.







The elements used to create pop-up menus are shown, above left. The **Action.Popup Menu** element can be used beneath a number of elements, including Labels and Images. In this example, the Label element is a child of a Rectangle shape element, to give it some borders.

One or more **Popup Option** elements are used beneath the Action.Popup Menu element. These are the individual items that will appear in the pop-up menu. The Popup Option element's attributes include an **ID**, which must be set, and a **Caption** which defines what text will appear in the pop-up menu item. The **Condition** attribute allows the display of individual items to be dynamic.

Another **Action** element is added beneath each Popup Option, defining what action will occur when the pop-up menu item is clicked. Link Parameters can be added, if desired, to pass values to the target.

You can even add an Action.Popup Menu element *beneath* a Popup Option element, so that selecting a menu item will display a sub-menu.

In the center example above, the elements have been given IDs that make their purpose a little clearer and, finally, the resulting menu is shown, above right. This is the basic arrangement for producing pop-up menus and redirecting the user to another report page or a process based on their pop-up menu selection. The pop-up menu only appears when the File link is clicked.

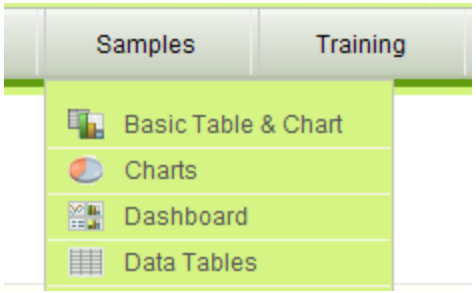
Other Menu Variants

Logi Info also includes other menu elements you may wish to investigate; they may be more useful depending on the nature of your application. "ReportCenter Menu" on page 753 is useful for displaying a menu based on the available reports and the Menu Tree menu (see "Menu Tree Menus" on page 776), introduced in v12.1, can be used to create horizontal and vertical hierarchical menus.

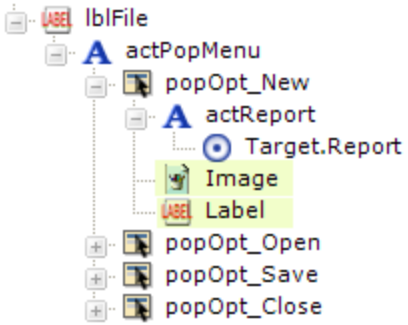
Adding an Image to the Pop-up Options

The menu item text that appears in the pop-up menu can be accompanied by an image.





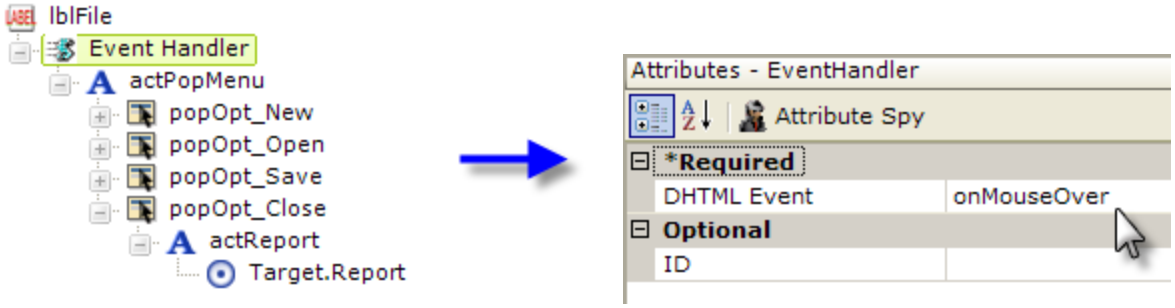
The examples above show a menu with and without the images. To include an image, you need to configure the Popup Option element differently:



Instead of using the Popup Options element's **Caption** attribute to set the text that will be displayed, leave it blank and then add an **Image** and a **Label** element, as shown above, as child elements. The Action element remains a child of the Popup Option element, rather than of the Image or Label.

Using the onMouseOver Event

You can make the pop-up menu appear when the mouse is hovered over the main menu link or image, which saves a mouse click:



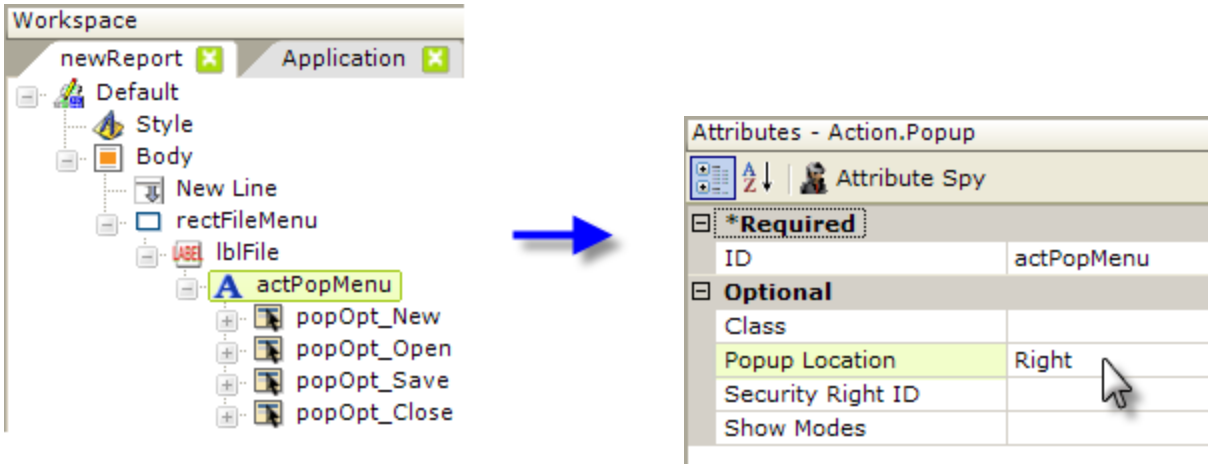
By introducing an **Event Handler** element *between* the main menu Label and the Action.Popup elements, as shown above, and setting its DHTML Event attribute to *onMouseOver*, you can cause the pop-up to appear when the mouse passes over the label, so clicking the link is not necessary.

Controlling Pop-up Location

The Action.Popup Menu element's **Popup Location** attribute controls where the pop-up menu will appear:



This allows the pop-up menu to be used with either a horizontal and vertical main menu. The **Left** pop-up menu example, shown above left, places it *directly below* and *left-aligned* with its parent link. The **Right** example places the pop-up menu *beside it* to the right and *top-aligned* with its parent link.



The Popup Location attribute is shown above and its default value is *Left.*

Controlling Pop-up Appearance

Pop-up appearance is controlled using style sheet classes. Main menu items (typically Label elements) can be assigned classes of your own design using the style sheet that's part of your Logi application. A separate existing style sheet controls the appearance of the pop-up menu but you can *override* the classes in that style sheet by adding the same classes to your application's style sheet.

Here's an example:

```
/* menu item text color and decoration - normal */
.rdPopupMenu A:link {
text-decoration: none;
color: #555555;
}

/* menu item text color and decorations - after 1st click */
.rdPopupMenu A:visited {
text-decoration: none;
color: #555555;
}

/* background color or pop-up menu */
.rdPopupMenu {
background-color:white;
}

/* background color, etc. for selected item */
```

```
.yuimenubaritemlabel-selected, .yuimenuitemlabel-selected, .yuimenubaritem-selected, .yuimenuitem-
selected {
background-color:#E0E0E0;
}
/* pop-up menu item text */
.yuimenuitemlabel {
font-size:100%;
}
```

Menu Tree Menus

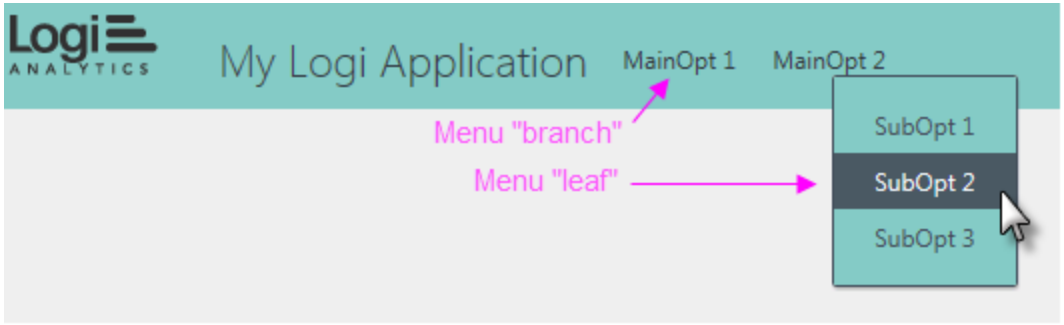
The **Menu Tree** element, and its child elements, are used to define a menu, in a horizontal or vertical orientation, with optional sub-menu branches and menu items.

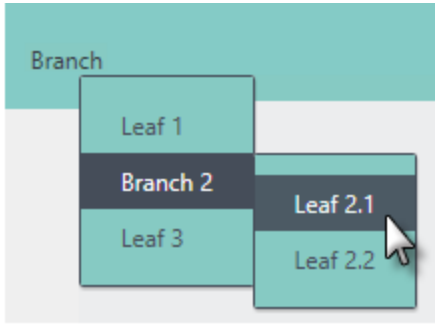
The following topics discuss the Menu Tree element:

- Menu Tree Attributes
- Menu Branch and Leaf Elements
- Creating a Menu
- Customizing Menu Appearance

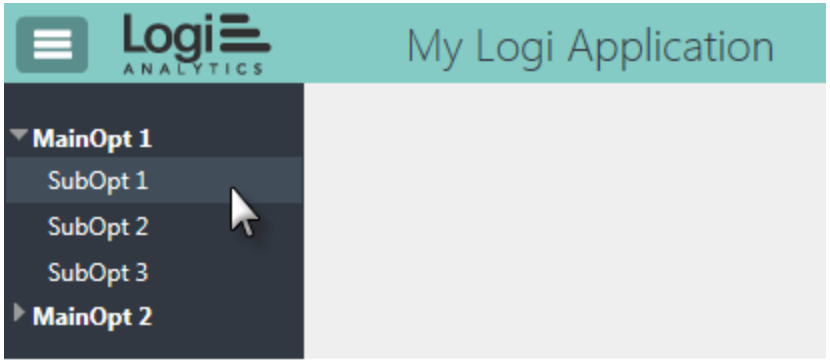
About the Menu Tree

Menus are typically used to launch actions such as running another report or linking to another web page. The **Menu Tree** element and its child elements, introduced in v12.1, allow you to easily to create a multi-level menu.

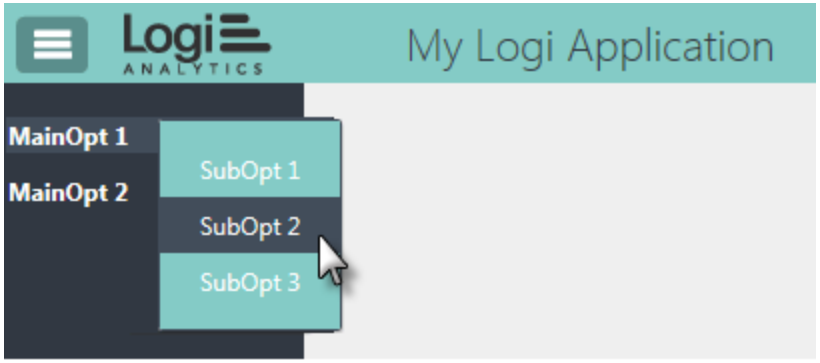




The Menu Tree can be configured for three different modes; the default mode is *HorizontalPopup*, shown above. The example uses two menu *branches* (MainOpt 1, MainOpt 2), each of which has three menu *leaves* (SubOpt 1, SubOpt 2, SubOpt 3). You can also nest branches and leaves to get additional suboptions, as shown at the far right.



The *Vertical* mode, shown above, lists the branches and leaves in a vertical hierarchy that can be expanded and collapsed.



And the *VerticalPopup* mode uses a vertical list of branches and pop-up panels to display the leaves. The arrangement of menus on the page is dependent on their containers (Divisions, Rows, etc.). The styling of the examples shown is controlled by a theme. You can supply additional styling to change their appearance.

Other Menu Variants

Logi Info also includes other menu elements you may wish to investigate; they may be more useful depending on the nature of your application. Pop-up menus (see "Popup Menus" on page 768) display a panel with menu options and the "ReportCenter Menu" on page 753 is useful for displaying a menu based on the available reports.

Menu Tree Attributes

The Menu Tree element is the container for the entire menu. It has the following attributes:

Attribute	Description
ID	(Required) Specifies a unique element ID.
Branch Collapsed Image	When a vertical menu mode is used, specifies an image that appears beside a menu branch that's collapsed. If located in _SupportFiles, then only the file name is required. Otherwise provide a complete file path. If left blank, a default image is used.
Branch Expanded Image	When a vertical menu mode is used, specifies an image that appears beside a menu branch that's expanded. If located in _SupportFiles, then only the file name is required. Otherwise provide a complete file path. If left blank, a default image is used.
Class	Specifies one or more CSS classes to be used by the element. When set, this class will also be used by all child elements that don't have their own class. May be used with or without a theme.
Disable Remember Expansion	Specifies whether the expanded/collapsed state of the menu branches is stored in the browser's Local Storage and re-applied at the next viewing. The default value is *False*, so specify *True* to disable this feature.
Horizontal Menu Spacing	Specifies the number of spaces displayed between menu branches when the Menu Mode is *HorizontalPopup*. The default is *5* spaces.

Attribute	Description
Level Indent	Specifies the amount of indentation, in pixels, used to offset successive menu leaf levels in a hierarchy when using a vertical menu mode. The default value is *10* pixels.
Menu Mode	Specifies the type of menu to be displayed: *HorizontalPopup*, *Vertical*, or *VerticalPopup*. See the About the Menu Tree section of "Menu Tree Menus" on page 776 for a visual example of each mode.
Security Right ID	When using Logi Security, specifies one or more Security Right IDs, separated by commas. Users with these Security Right IDs will be able to see the menu, users without them will not.

Menu Branch and Leaf Elements

The menu "options" are created using **Menu Branch** and **Menu Leaf** elements. A branch is an option that will have sub-options (leaves), while a leaf is an option that has no sub-options. A menu can have any number of branches and leaves, and many levels of nesting.

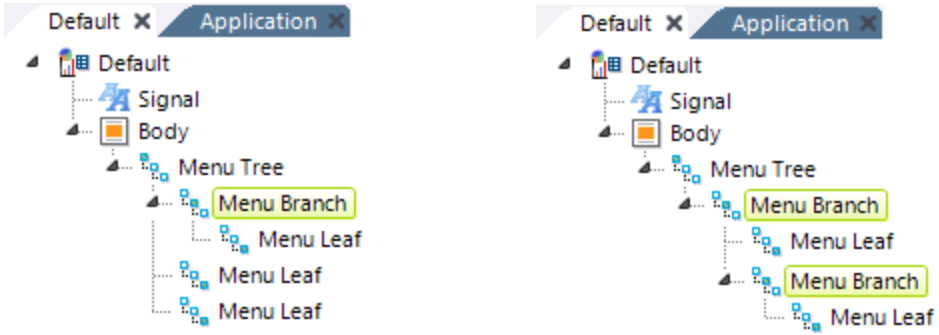
These two elements have the same attributes:

Attribute	Description
Caption	(Required) Specifies the visible text of the menu option.
Class	Specifies one or more CSS classes to be used by the element. When set, this class will also be used by all child elements that don't have their own class. May be used with or without a theme.
Condition	Specifies an expression that evaluates to a value of True or False. If True, then the option is displayed, otherwise it's removed. Expressions should be in JavaScript or intrinsic functionsyntax. Typically, expressions compare values using a token, such as @Data.value~ < 0. Enclose both sides of the expression in quotes when working with strings: "@Data.myColumn~" == "SomeValue".
Expanded	(Menu Branch only) Specifies whether a branch is expanded when the menu is initially displayed. The default value is *False*.
ID	Specifies a unique element ID.
Security Right ID	When using Logi Security, specifies one or more Security Right IDs, separated by commas. Users with these Security Right IDs will be able to see the option, users without them will not.

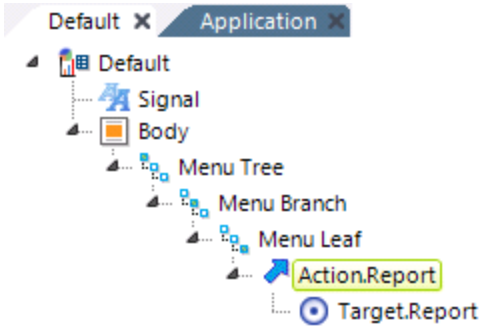
Attribute	Description
Tooltip	Specifies the text that will appear when the user hovers the cursor over the menu option.

Creating a Menu Tree Menu

Once you've decided which menu mode your page design will use, it's easy to create the menu:



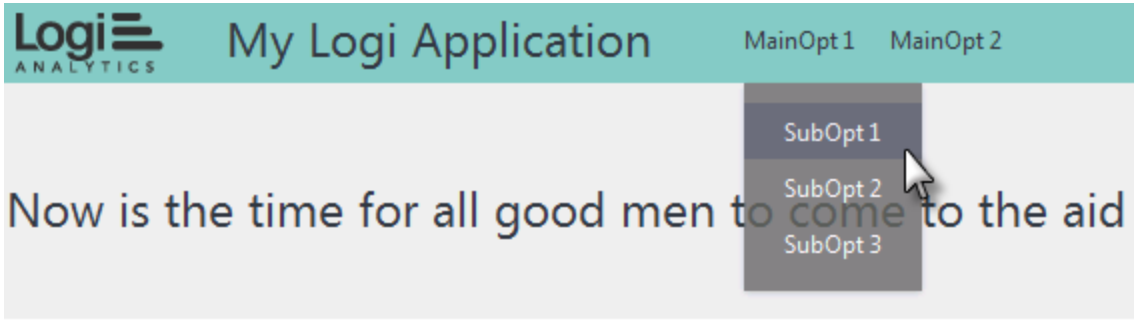
As shown in the image above, simply add a Menu Tree element and whatever combination of Menu Branch and Menu Leaf elements are necessary. The example on the left is simple, with only one option (branch) with a sub-option. The example on the right nests a branch beneath another branch.



As shown above, **Action**, **Event Handler**, and other elements typically found beneath links are used to control what happens when a menu option is clicked. The Menu Leaf element is the parent of these link-related elements.

Customizing Menu Appearance

An easy way to customize the appearance of Menu Tree menus is by customizing the theme used with the report or application, *Using the Theme Editor*. Menu border, background, and item colors can be set in the Colors category, Menus section of a custom theme. You can also apply styling using CSS classes directly to the Menu Tree, Menu Branch, and Menu Leaf elements. For example, to make the menu item fonts larger, select a class like *ThemeTextLarger* to an element's **Class** attribute. If you're using your own style sheet, assign a class from it that specifies a larger font size to the attribute. You can also use Conditional Class elements beneath any of the Menu Tree Menu elements in order to apply classes dynamically. For more information about Conditional Class elements, see *Using Style Sheets*. Here's an example of how to use CSS to change the appearance of the Menu Tree pop-up panel:



The pop-up panels shown above have these qualities: they have no border, they appear just below the header, and they're a little transparent. To achieve that, you'll need to override an existing theme class by adding it to your own style sheet:

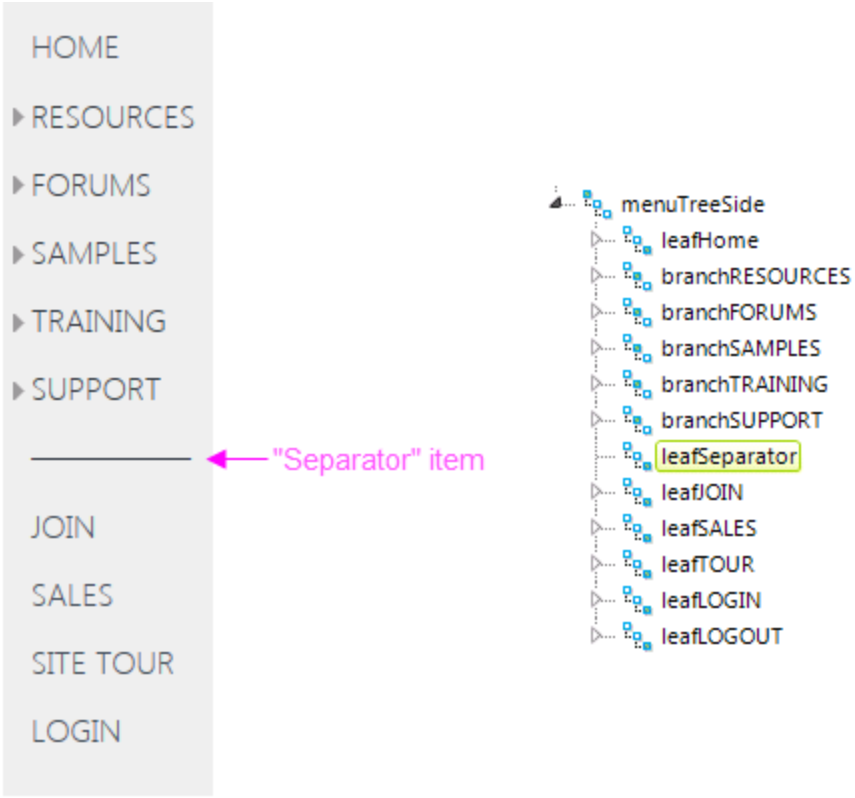
```
.rdPopupMenu { border: 0px; opacity: 0.8; top: 39px !important; }
```

You'll need to adjust the "top" attribute to align it correctly with the bottom of your header.

This change may affect *all* pop-up panels in your application.

Adding a "Separator" Menu Item

When using vertical menus, sometimes it's desirable to provide some kind of visual separator between menu items:



In the example shown above, a "separator" item provides visual separation between one category of menu items and another. This is easily created by adding a **Menu Leaf** element that has no action associated with it (so it's not a link) and has a Caption that provides the desired visual. In the example above, several of the Unicode characters "Horizontal Bar" (U+2015) were used.

Glossary

A

API

API, short for Application Program Interface, is a set of routines, protocols, and tools for building software applications. In business intelligence, APIs may be used to enable end-users to directly update source systems.

Authentication

Authentication is the verification of a user's identity.

Authorization

After a user's identity has been authenticated, authorization grants or denies access to reports, columns, and records to selected users or user-groups.

B

Big Data

Refers to both the ever-growing volumes of data in use today and also to services that are specifically engineered to provide and manipulate very large data volumes.

Business Analytics

Business analytics, or business intelligence (BI), gives customers the ability to rapidly create scalable, interactive data analysis applications, and self-service capabilities users can access from anywhere and on any device.

C

Columnar Data Store

Columnar data store is a type of big data repository containing structured data in columns and rows. The main benefits are that the data can be highly compressed and is easily searchable.

CRM

A Customer Relationship Management (CRM) system is a database-based system that records a company’s daily customer-related transactions. CRMs can help customer representatives to provide better service, close more deals, and increase revenue.

CSS

Cascading Style Sheets (CSS) is a technology that allows the presentation aspects of web pages to be separated from the page content. It can be used to add "styling" (e.g. apply fonts, colors, alignment, spacing, and more) to web pages.

D

Data Discovery

Data discovery is the capability to analyze data on-the-fly and uncover insights from it.

Data Enrichment

Data enrichment is a method of preparing data to make it ready for analysis and exploitation, and can include formatting, adding calculations, joining with other data, and more.

DevNet

The Logi Developer Network website.

Drill Down

Drill Down is a capability that allows the user to get a view of the underlying or supporting data used in an analysis.

Drill Through

Drill Through is similar to Drill Down but takes it one step further by applying analysis to the underlying or supporting data.

E

Elemental Development

A development approach used in Logi Info that lets developers build feature-rich applications by using reusable, pre-built elements, rather than by writing low-level code.

F

Forecasting

A technique involving data mining and analysis leading to predictions about what will happen in the future.

G

Geo Mapping

The combination of geographic and other data to produce map visualizations, such as Google or Leaflet maps.

H

Heatmap

A Heatmap chart, sometimes called a "tree map", which uses a unique arrangement of rectangles to represent data and relationships, using color and size.

I

Interpolation

The process of evaluating a literal value match containing one or more placeholders, yielding a result in which the placeholders are replaced with their corresponding values.

J

JavaScript

JavaScript is a programming language supported by the majority of modern web browsers and used by many websites.

JDBC

Java Database Connectivity (JDBC) is an API used to access relational databases. Open Database Connectivity (ODBC) is a similar API designed for use with Java.

JSON

JavaScript Object Notation (JSON) is a lightweight data-interchange format that's easy for humans to read and write, and easy for computers to parse and generate.

K

KPI

Key Performance Indicators (KPIs) are visual indicators, in the form of color-coded shapes, which are tied to a pre-defined, critical threshold.

L

LDAP

The Lightweight Directory Access Protocol (LDAP) is an Internet protocol applications use to look up information from a server and is frequently used for containing user login information.

M

My Term

My definition

N

NoSQL

"Not only SQL" (NoSQL) is an alternative to traditional relational databases, and doesn't rely on tables and a pre-determined schema. NoSQL databases are especially useful for working with large sets of distributed data.

O

ODBC

Open Database Connectivity (ODBC) is an API used to access relational databases. Java Database Connectivity (JDBC) is a similar API designed for use with Java.

OLAP

Online Analytical Processing (OLAP) is the process of analyzing data stored in multi-dimensional "cubes".

R

REST

Representational State Transfer (REST) is a type of API used to provide interoperability between computer systems on the Internet.

S

SSM

The Self-Service Module (SSM) is a package that includes Logi Info + SSRM + Discovery or Logi Platform Services.

SSRM

The Self-Service Reporting Module (SSRM) is a Logi Info add-on module that adds special elements to Info and includes the InfoGo application.

W

Write-Back

The ability to update data sources, typically by adding, editing, or deleting data.