TOC

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material may assist with understanding historical functionality, please be aware that the software described is no longer maintained at this version level and may contain outdated or inaccurate information.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.

Datalayers

A *datalayer* is a temporary container for data retrieved from a datasource and, after retrieval, its data is cached in memory and/or in XML data files on the web server.

The following topics introduce developers to the use of datalayers in Logi Info:

Selecting the Correct Datalayer

In Logi reporting products, developers use datalayers to **retrieve data** for tables, charts, and input select lists from a **data-source**. A datasource can be one of the following:

- OLEDB-, ODBC-, and JDBC-compliant databases
- XML, CSV, Excel, JSON, and other data files
- LDAP directories
- Web feeds, web services, or web pages
- REST and SOAP APIs
- Hard-coded, static data
- Your Logi application's metadata

A number of different datalayer elements are available and your selection can depend on both the datasource and the retrieval technique. The following table lists the typical uses for datalayers within Logi reporting products. The links direct you to individual topics that discuss the use of each datalayer.

Datalayer Type	Description
"DataLayer.ActiveSQL" on page 236	A special type of datalayer designed for use with the Analysis Grid super-element, it only retrieves a limited number of rows based on an initial SQL query and, in response to runtime manipulations of the Analysis Grid interface by users, it dynamically modifies and resends its query.
"DataLayer.Bookmarks" on page 252	Retrieves data directly from bookmark collection files.

Datalayer Type	Description
"DataLayer.Cached" on page 255	Modifies the normal data retrieval activities of datalayers; when it's used, data is retrieved, cached, and made available for use in Logi reports for a specific time period, after which the data is refreshed (deleted and recreated).
"DataLayer.CSV" on page 69	Retrieves data directly from a .CSV text file.
~~DataLayer. Data Services~~ *Deprecated*	Uses Logi Services for data retrieval and is only available in Logi Info if the Discovery Module v3.x has been installed.
DataLayer.Dataview	Retrieves the data for the Thinkspace element.
"DataLayer.Definition List" on page 277	Retrieves a list of definition files used in your application, and their properties, such as author name, timestamp, and engine version.
"DataLayer.Directory" on page 78	Retrieves a list of files and/or folders in a specified directory, including their size, timestamps, and other properties.
"DataLayer.Excel" on page 301	Retrieves data directly from a Microsoft Excel spreadsheet file.
"DataLayer.Fixed File Format" on page 82	Retrieves data directly from text files that use column widths to specify data format.
"DataLayer.Google App"	Enables connection to the datastore of a published application hosted by the Google App Engine.

Datalayer Type	Description
on page 306	
"DataLayer.Google Spreadsheet" on page 313	Retrieves data in a Google online spreadsheet.
DataLayer.GPX File	Retrieves data directly from a GIS data file in the GPX format.
"DataLayer.JSON" on page 98	Retrieves data directly from a JSON data file. *Prior to v12.1, this datalayer was named DataLayer.JSON File.*
DataLayer.KML File	Retrieves data directly from a GIS data file in the KML format.
"DataLayer.LDAP" on page 87	Retrieves data from a Lightweight Directory Access Protocol (LDAP) directory.
"DataLayer.LDAP - DataLayer.LDAP Authentication" on page 95	Authenticates user against a Lightweight Directory Access Protocol (LDAP) directory.
DataLayer.Linked	Reuses data retrieved in another datalayer. For more information, see "Link Datalayers" on page 61.
"DataLayer.MDX" on page 284	Retrieves cube data and populates Logi OLAP Grid or OLAP Table elements.
"DataLayer.Mongo Find"	Retrieves documents from a MongoDB collection using the MongoDB Find API.

Datalayer Type	Description
on page 323	
"DataLayer.Mongo Map Reduce" on page 328	Runs a MongoDB map-reduce operation to return one or more documents.
"DataLayer.Mongo Run Command" on page 333	Runs a MongoDB command, suitable for use with the Aggregation Pipeline, a simpler alternative to using map-reduce operations, to return one or more documents.
"DataLayer.Plugin" on page 294	Retrieves data from a custom-written code module, the "plug-in".
"DataLayer.REST" on page 111	Retrieves data from a REST-style web service.
DataLayer.Scheduler	Retrieves data associated with the Logi Scheduler service. For more information, see Using Logi Scheduler.
~~DataLayer.SimpleDB~~ *Deprecated*	Retrieves data from Amazon's SimpleDB web service.
"DataLayer.SP" on page 126	Retrieves data from a SQL database, using a Stored Procedure.
"DataLayer.SQL" on page 146	Retrieves data from a SQL database, using a SQL query.

Datalayer Type	Description
"DataLayer.Static" on page 164	Uses hard-coded data values.
"DataLayer.Twitter" on page 337	Retrieves tweets and messages from the Twitter web site.
"DataLayer.WebFeed" on page 176	Retrieves data from an RSS or Atom web feed.
"DataLayer.Web Scraper" on page 184	Retrieves data from within web pages or HTML files.
"DataLayer.Web Service" on page 204	Retrieves data from a SOAP-style web service.
"DataLayer.XML" on page 219	Retrieves data directly from an XML data file. *Prior to v12.1, this datalayer was named DataLayer.XML File.*
"DataLayer.XOLAP Query" on page 297	Pre-defines a data view based on a Logi XOLAP cube.

The type of datalayer used often matches the type of **Connection** element used in your _Settings definition. Some datalayers, such as DataLayer.CSV and DataLayer.Excel, directly access the web server file system and may not require a connection. The

Connection.HTTP element allows the following datalayers to optionally use a connection to an HTTP or HTTPS datasource that requires authentication:

- DataLayer. CSV
- DataLayer.Excel
- DataLayer.Fixed Format File
- DataLayer.GPX File
- DataLayer.JSON
- DataLayer.KML File
- DataLayer.Web Feed
- DataLayer.Web Scraper
- DataLayer.XML

The following datalayers include the optional **Http Method** attribute, which allows you to select the one of these verbs to be sent with a REST request: DELETE, GET, POST, or PUT, or to enter a custom verb:

- DataLayer.JSON
- DataLayer.REST
- DataLayer.XML

The default is POST.

Many datalayers are run by the Logi Engine in *separate* threads, which means that data retrieval time is limited to the longest running query, rather than the number of queries, resulting in excellent performance. This includes all datalayers except the following: Local Data datalayers, DataLayer.Linked, DataLayer.XML, DataLayer.Cached, DataLayer.Bookmark, DataLayer.Static, and any datalayer whose data is displayed using the Auto Columns element.

Data Engine performance, when processing large numbers of DateTime-type columns in a result set, can be significantly improved by skipping Time Zone processing, if it's not needed in the application. This is done using a special constant in the _Settings definition:

```
rdSQLIncludeGMTOffset = False
```

However, this feature does *not* offer improved performance with small numbers of DateTime columns.

Debugging Datalayers

Datalayer results and performance can be viewed during development using the Debugger Trace Report:

Get DataLayer.SQL	ID = dlOrderSQL		.050
	Connection ID	connNW	.146
	Connection Type	SqlServer	.147
	Source	SELECT * FROM Orders WHERE CustID = 'VINET'	.178
	Process SQL	Open connection	.181
	Process SQL	Send SQL command to data source	.249
	Process SQL	Received response	.297
	Process SQL	Schema built	.310
	Processing SQL	Requesting first row	.335
	Processing SQL	Retrieved first row	.336
	Process SQL	Connection closed	.397
	Process SQL	Database server took 0.115 seconds to process the query	.398
	Processed Row Count	20	.398
	Processed Data	View Memory Stream Data (7,215 bytes)	.398

The debug information for a typical DataLayer.SQL run is shown above.

This is *very* useful in determining the exact query (especially if tokens are being used) and the execution time required. The actual data retrieved can also be viewed using the Memory Stream Data link. If multiple datalayers run concurrently, they'll be shown here with individual links to their details.

Element IDs for datalayers are optional. However, we suggest you *always* provide a unique datalayer element ID, for ease in understanding the definition and, significantly, so that the datalayers are easily identified in the Debugger Trace page.

More information about use of the debugger can be found in *Debug Reports*.

Using the Datalayer Data

The following example shows how the data in a Data Table corresponds with the XML data retrieved using DataLayer.SQL to query the Northwind database:

EmployeeID	LastName	FirstName	Title
1	Davolio	Nancy	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative
5	Buchanan	Steven	Sales Manager
6	Suyama	Michael	Sales Representative
7	King	Robert	Sales Representative
8	Callahan	Laura	Inside Sales Coordinator
9	Dodsworth	Anne	Sales Representative

```
<dt EmployeeID="1" LastName="Davolio" FirstName="Nancy" Title="Sales Representative" />
<dt EmployeeID="2" LastName="Fuller" FirstName="Andrew" Title="Vice President, Sales" />
<dt EmployeeID="3" LastName="Leverling" FirstName="Janet" Title="Sales Representative"/>
<dt EmployeeID="4" LastName="Peacock" FirstName="Margaret" Title="Sales Representative"/>
<dt EmployeeID="5" LastName="Buchanan" FirstName="Steven" Title="Sales Manager" />
<dt EmployeeID="6" LastName="Suyama" FirstName="Michael" Title="Sales Representative" />
<dt EmployeeID="7" LastName="King" FirstName="Robert" Title="Sales Representative" />
<dt EmployeeID="8" LastName="Callahan" FirstName="Laura" Title="Inside Sales Coordinator" />
<dt EmployeeID="9" LastName="Dodsworth" FirstName="Anne" Title="Sales Representative" />
```

The SQL query used was:

```
SELECT EmployeeID, LastName, FirstName, Title FROM Employees
```

The Logi Engine converts data retrieved from any datasource into XML. Each row from the datasource is equivalent to one XML element and each column is equivalent to one XML attribute. For example, the Data Table shown above contains four columns and nine rows, which is equivalent to nine XML elements, each with four attributes.

Data retrieved by a datalayer is cached in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

Data retrieved by the DataLayer.XML and DataLayer.Web Service elements may need to be reformatted for use, and the **XslTransform** element is available for this purpose.

Use of "bracketed" column names, such as "[Order Date]", is supported. The brackets allow column names to include spaces, special characters, or SQL reserved words.

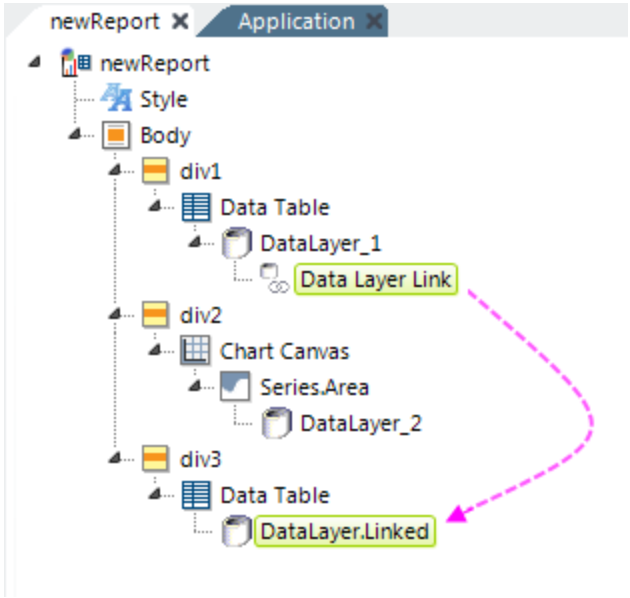
The data retrieved with a datalayer is available using **@Data** tokens, in the format @Data.*ColumnName*~. The spelling of the column name is **case-sensitive**.

For example, `@Data.LastName~` retrieves all values, in all rows, for the *LastName* column. This might sound as if you get all the data in one place in your report but you don't. When the reporting engine generates the report, it **iterates** through each row in the datalayer; the @Data token at any point in time represents the value in a column for the *current* row in the iteration.

Data Scope

The scope, or availability, of data is generally limited to the element (Data Table, Chart, etc.) which is the parent element of the datalayer element used to retrieve the data. The following example illustrates the use of multiple datalayers and the availability of

their data.

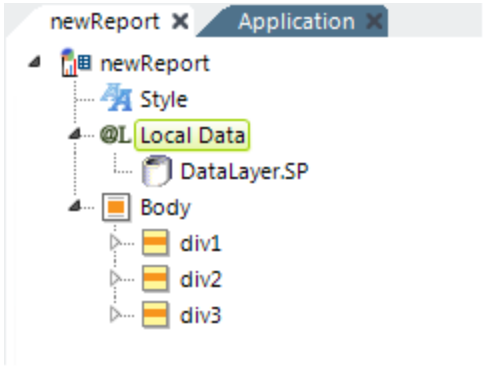


1. The Data Table element within the *div1* division can only use data from *DataLayer_1* datalayer.
2. The chart's Series.Area element within the *div2* division can only use data from the *DataLayer_2* datalayer.
3. The Data Table element within the *div3* division can also use the data from *DataLayer_1* by virtue of the two elements, Data Layer Link and DataLayer.Linked, which link it to the datalayer's cached data.

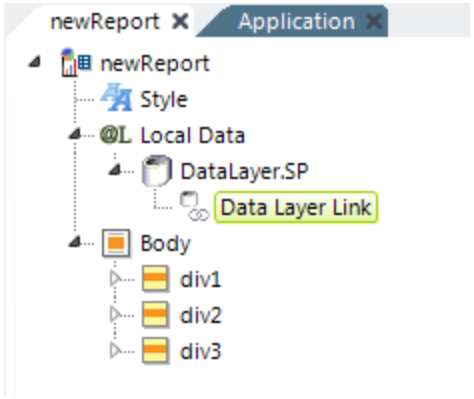
Linking datalayers is a very useful practice, allowing the data retrieved into one datalayer to be available for re-use in another. This eliminates extra datasource queries and improves performance. For more information, see "Link Datalayers" on page 61.

Using Local Data

A special data container element, **Local Data**, can be used to make data available *anywhere* within the report:



First row of data *only* is available anywhere in the definition, using @Local tokens



All rows of data are available using linked datalayer and @Data tokens

As shown above, Local Data is a child of the report root element, not of the Body or other lower elements. The data values retrieved by Local Data's child datalayer is accessed using special **@Local** tokens, in the format @Local.*ColumnName*~. These tokens can be used *anywhere* in the report, and this is an excellent way to use data, for example, in headers and footers.

If there are multiple Local Data elements in use, it may be difficult to determine which data came from which Local Data element. To make usage easier to understand, if a Local Data element is given an element ID, then that ID can be incorporated into its tokens. For example, if the Local Data element ID is "localOrders" and its datalayer queries the Northwind Orders table, this token

can be used to access Freight column data: `@Local.localOrders.Freight~`. This is an optional notation variantyou may wish to use to improve definition readability.

By themselves, @Local tokens can only be used to access the values of the *first row* of the data retrieved under Local Data. However, you can use the linking elements shown earlier to link the Local Data datalayer to other datalayers in the report definition, which then provides access to *all rows* of data in the Local Data datalayer. The linked data will then be subject to the same rules as those for any other datalayer (scope limited to its parent element and accessed using @Data tokens).

Developers often use this technique (linked Local Data datalayers) to reduce data retrieval to one query, which is then re-used via linking in multiple places in the report, for best performance.

Using Multiple Local Data Elements

It's also common to use multiple Local Data elements in the same report definition. They will run *sequentially*, not simultaneously, so that data from one can be used in the query of another. In the data retrieved, the effect is a "union" of all of the Local Data data.

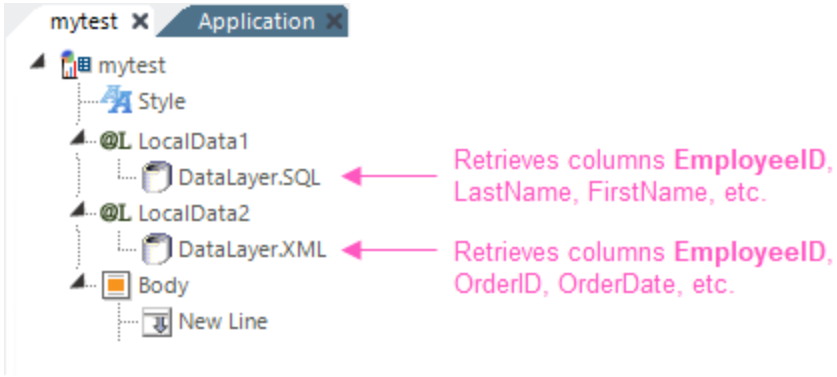
Local Data datalayers are run before other datalayers, so data from them can be used to condition the retrieval of data by other datalayers. Their datalayers are *not* re-run with all AJAX refresh requests. They are only re-run when the element being refreshed contains either a DataLayer.Linked element linked to the Local Data datalayer, or when it contains an @Local 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).

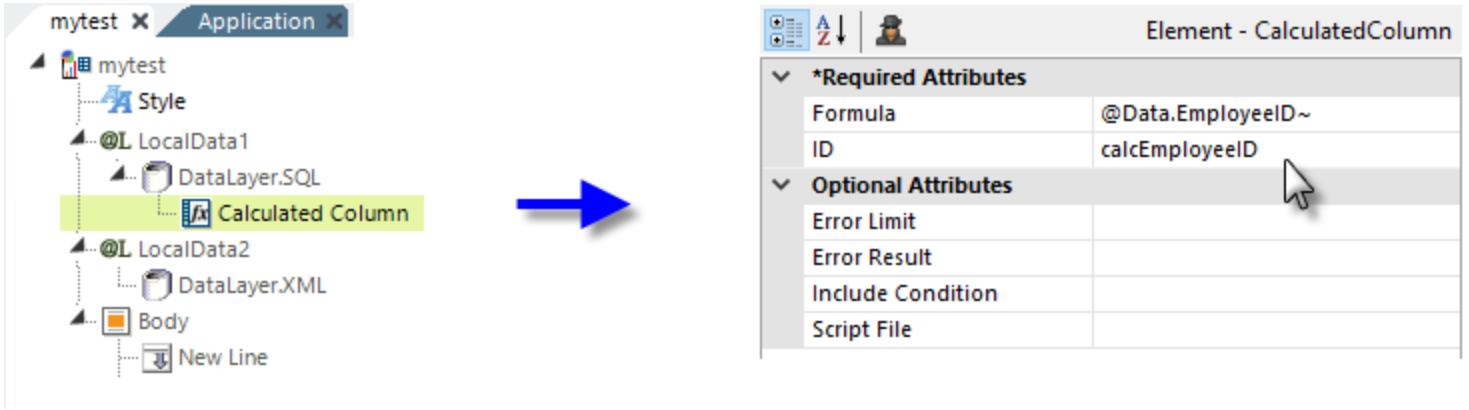
This can be very useful, for example, if you just want to run the Local Data datalayer *once* when the report is first loaded.

Handling Columns with the Same Name

If you're using multiple Local Data elements, it's possible that their datalayers could retrieve data from data sources with similar schema and identical column names.



In the example shown above, suppose that LocalData1 → DataLayer.SQL retrieves a column named "EmployeeID" and LocalData2 → DataLayer.XML does too. What will happen? When the data from both datalayers is combined, the values for EmployeeID from the first datalayer will be *overwritten* by the values from the second datalayer. Using the @Local.EmployeeID~ token later in the application will only produce values from the second datalayer.



To make the values from the first datalayer available, use a **Calculated Column** element, as shown above, beneath it. This will add a new column with the same values but a different name. Then in the application, use the @Local.calcEmployeeID~ token to access the values.

If the EmployeeID column contains string values, in the Formula attribute you'll need to enclose the @Data.EmployeeID~ token in double-quotes.

Manipulating the Data

Once data has been retrieved into a datalayer, a number of elements can be used to manipulate its data. This is generally done by either re-arranging rows, removing rows, or by adding columns to the datalayer. These elements are summarized below; the links direct you to individual topics that discuss the use of each element:

Element	Description
Aggregate Column	Adds a new column to the datalayer that aggregates data from another column. Functions include *Average*, *Count*, *DistinctCount*, *Max*, *Median*, *Min*, *Mode*, *Sum*, and *StdDev.* For more information, see *The Aggregate Column*
Calculated Column	Adds a new column to the datalayer whose data is the result of a formula that uses data from other columns in the same row or other token values. For example, multiplying the values from two columns together. JavaScript functions are supported in formulae.
Color Spectrum Column	Adds a new column to the datalayer, containing color values that fall between the LowValueColor and HighValueColor attribute values. The color for each row is generated based on where the current Data Column value falls between the Min Range and Max Range.
Compare Filter	Analogous to the *WHERE* clause in a SQL query, this element removes rows from the datalayer that do not meet specified criteria. Uses native data engine code for fastest performance. For more information, see *The Compare Filter*.
Condition Filter	Similar to the Compare Filter, this element uses scripting to remove rows from the datalayer that do not meet specified criteria. However, this filter uses scripting and therefore performs *much more slowly* than the Compare Filter. We recommend that you use the Compare Filter instead whenever possible. For more inform-

Element	Description
	ation, see The Condition Filter.
Contain Filter	Analogous to using a *WHEREx CONTAINS y* clause in a SQL query, this element removes rows from the datalayer that do not return results in a text search of specified columns. For more information, see *The Contain Filter*.
Crosstab Filter	"Pivots" the data to convert the datalayer into a "cross-tab" format.
DeDuplicate Filter	Analogous to using a *DISTINCT* clause in a SQL query, this element removes rows from the datalayer that have duplicated values in specified columns. For more information, see *The DeDuplicate Filter*.
Difference Column	Adds a new column to the datalayer that contains the difference, as a number or as a percentage, between a numeric value in a column in the current and the previous rows.
File Column	Used with BLOBs and CLOBS, this element copies the column data and saves it to a file, then optionally adds columns to the datalayer containing the saved file's name and path. For more information, see *The File Column (BLOBs)*.
Flattener	Processes hierarchical data into a tabular set of rows and columns, so that it can be used with a range of Logi elements. For more information, see *The Flattener*.
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. For more information, see *The Forecasting Elements*.

Element	Description
Formatted Column	Adds a new column to the datalayer that represents the source value after formatting. For more information, see *The Formatted Column*.
Group Filter	Groups rows in the datalayer based on one or more column values and allows group aggregate values to be created. For more information, see *The Group Filter*.
Join	Analogous to a *JOIN* clause in a SQL query, this element adds columns to the datalayer so that multiple tables can be joined to create a single dataset. For more information, see "Join Datalayers" on page 53.
Lookup Element	Performs a "look up" of related data. It runs its own datalayer once for each row in its parent datalayer and automatically joins the results. For more information, see *The Lookup Element*.
Moving Average Column	Adds a new column to the datalayer based on the average value of another column spread over some number of previous rows. Used to smooth data series and make it easier to spot trends.
Percent of Total Column	Adds a new column to the datalayer containing the percentage of some value in the row compared to the total of that value in all rows.
Rank Column	Adds a new column to the datalayer containing either a numeric or percentile rank based on all the values of some other column.
RegEx Filter	Removes rows from the datalayer by applying pattern matching using regular expressions. For more information, see *The RegEx Filter*.
Relevance Fil-	Removes rows (*irrelevant* data) from the datalayer that do not meet a threshold; optionally, irrelevant rows

Element	Description
ter	can be retained, grouped together, and handled as an individual entity. For more information, see *The Relevance Filter*.
Remove Columns	Removes one of more columns from the datalayer to reduce the size of the data during processing. If a column is no longer required, you can use this element to delete it.
Rename Columns	Renames one or more columns in the datalayer. To use it, specify a comma-separated list of existing column names in the Column Names attribute, and a matching number of comma-separated new names in the New Column Name attributes. Columns named in the first list but which do not exist will be ignored.
Running Total Column	Adds a new column to the datalayer containing values based on the sum of all previous values of another column.
Security Filter	Removes rows from the datalayer, like a Condition Filter, but is applied based on a user's security rights. For more information, see Securing Data.
Sequence Column	Numbers the rows in a datalayer by adding a new column containing a sequence integer for each row, starting at 1. Also very useful for *counting* the number of datalayer rows.
Sort Filter	Sorts the datalayer rows. For more information, see *The Sort Filter*.
Switch Column	Makes data replacements. It's analogous to using both a CASE structure and a REPLACE function in a SQL query. For more information, see The Switch Column.
Time Period	Adds a new column to the datalayer containing a time period value (Year, Month, Day, Hour, Minute, etc.)

Element	Description
Column	functions. For more information, see Time Period Columns.
UnCrosstab Filter	Used 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. For more information, see The UnCrosstab Filter.
Unit Conversion Column	Converts column values from one standard measurement to another. For more information, see The Unit Conversion Column.

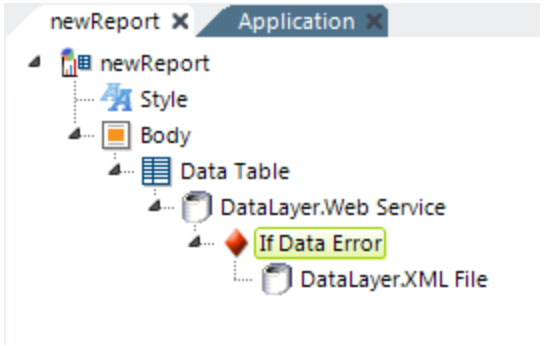
Columns that are added to a datalayer using these elements are then available for use and subsequent manipulation themselves, like any other datalayer column, and their values are available using @Data tokens.

The Include Condition Attribute

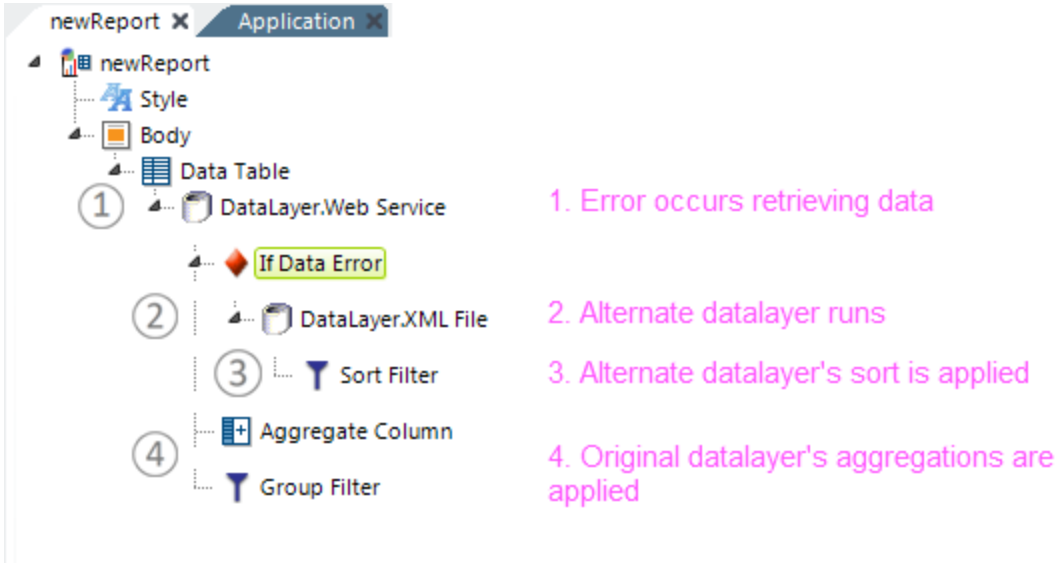
All of the datalayer manipulation elements listed in the table above have an **Include Condition** attribute. If the value of this attribute is left blank or contains a formula that evaluates to *True*, the element is applied to the datalayer. If the value evaluates to *False*, the element is ignored and does not affect the datalayer. This powerful feature allows developers to dynamically determine if the datalayer will be manipulated or not.

Datalayer Error Handling

The **If Data Error** element allows developers to handle errors, such as a failure to find or connect to data sources, or missing or corrupt data files, that may occur while retrieving data into a datalayer. In this way, errors are trapped and processing can be switched to alternate data sources, instead of displaying the system error page.



Consider the example shown above. A Data Table normally uses a Web Service to retrieve the latest currency conversion information. However, if there is some problem with the web service and an error occurs, the If Data Element allows the datasource to switch to an XML data file. In this way, data is still presented and an error page is not displayed.

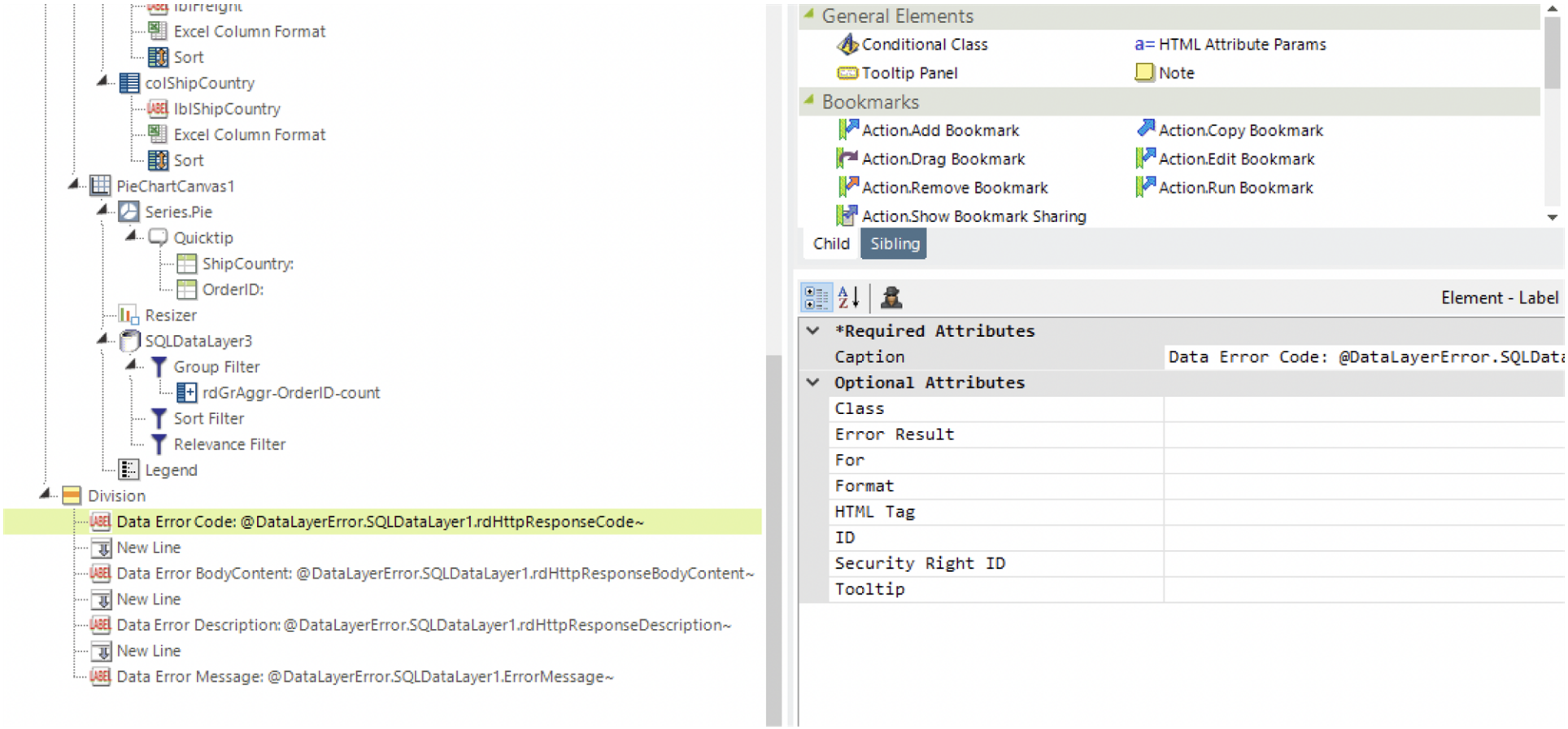


What about any data manipulation elements that may exist? As shown above, when an error occurs, the alternate datalayer replaces the original datalayer as the data retrieval mechanism and applies its own data manipulation elements. Then the application will process any data manipulation elements for the original datalayer.

You may use multiple, nested 'If Data Error' elements. The Debugger Trace page will include an entry containing the actual error message and any details generated each time an If Data Error element is processed. The @Function.ErrorDataLayerID~ token can be used to determine the element ID of the datalayer that encountered the error.

You can also retrieve Data Table and Data Chart error messages using the 'If Data Error' element and capture all of the error information in the new token element, `@DataLayerError`.

To utilize this feature, add the 'If Data Error' element to your datalayer and configure the `@DataLayerError` token using your datalayer ID:



Depending on the information you want to display on the page, you can choose from the following tokens:

- `ErrorMessage~`
- Restful data source only: `rdHttpResponseBodyContent~`

- Restful data source only: `rdHttpResponseBodyCode~`
- Restful data source only: `rdHttpResponseDescription~`

As a result, your error page will resemble the following:

```
Data Error Code: 400
Data Error BodyContent: {"errorCode":"E0000019","errorSummary":"Bad request. Accept and/or Content-Type headers likely do not match supported values.","errorLink":"E0000019","errorId":"oaewnI33TLGT_SNrI-yVoIxIg","errorCauses":[]}
Data Error Description: Bad Request
Data Error Message: Error getting a Web request. Error: {"errorCode":"E0000019","errorSummary":"Bad request. Accept and/or Content-Type headers likely do not match supported values.","errorLink":"E0000019","errorId":"oaewnI33TLGT_SNrI-yVoIxIg","errorCauses":[]}
```

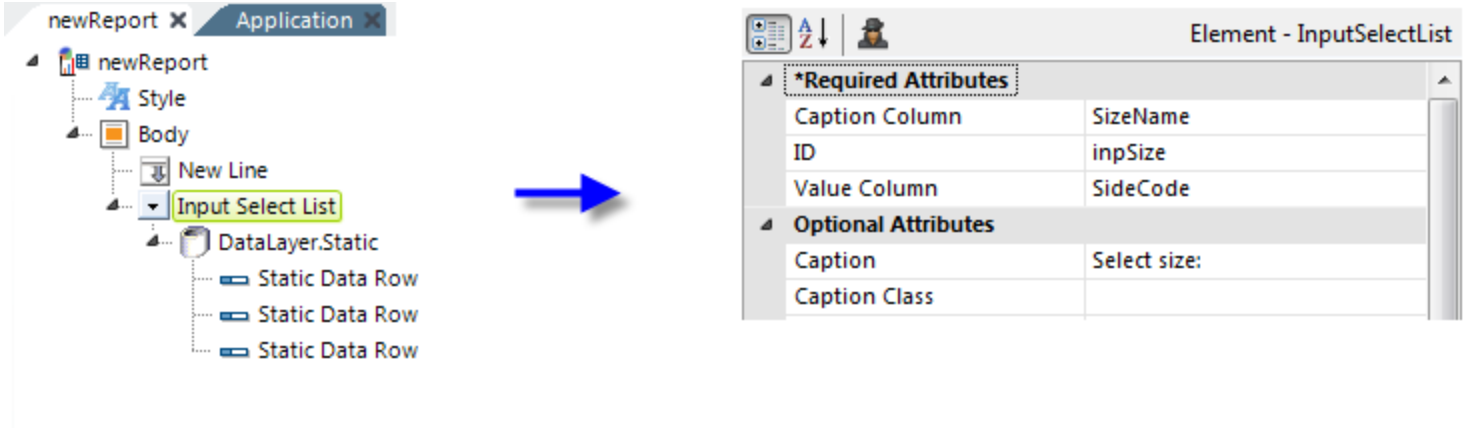
 Use divisions to customize the error message to include images, buttons, and links.

Usage Examples

Datalayer elements are available for various table, chart and user-input elements. In Studio, the Element Toolbox Panel displays all the datalayer elements when an appropriate parent element is selected. The following three examples illustrate common uses of datalayers.

Using Hard-Coded Data in an Input Select List

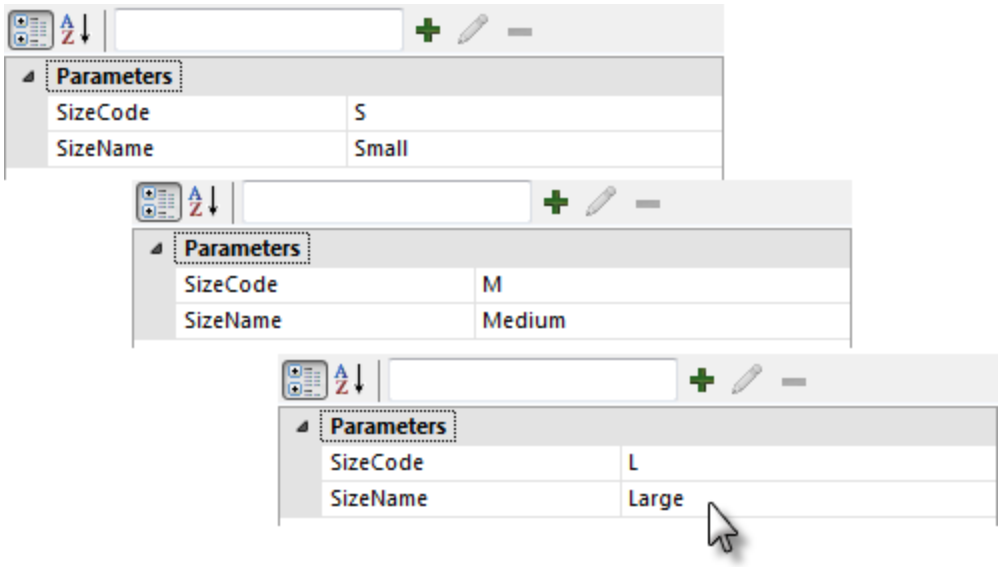
Hard-coded values can be used with the **DataLayer.Static** element:



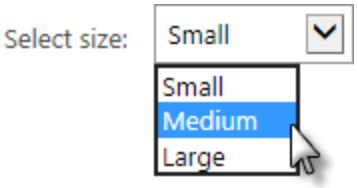
In the example above, the user should be able to select garment sizes from a drop-down list. Hard-coded data makes sense here because the standard sizes (Small, Medium, Large) are few in number and static. To make this functional:

1. Add the **Input Select List** element to the definition, as shown above.
2. Beneath it, add a **DataLayer.Static** element, which has no attributes, and three **Static Data Row** child elements, one for each choice (garment size) in the drop-down list.

3. Set the Input Select List element's attributes as shown above. The values for the **Caption Column** and **Value Column** attributes are an arbitrary names ("SizeName" and "SizeCode") that will be referenced in the next step.



4. Set the attributes for each **Static Data Row** elements as shown above, one element for each size.



When the report is run, the datalayer will get its displayed options (the "Caption") and related values from the Static Data Row elements and the Input Select List will display them.

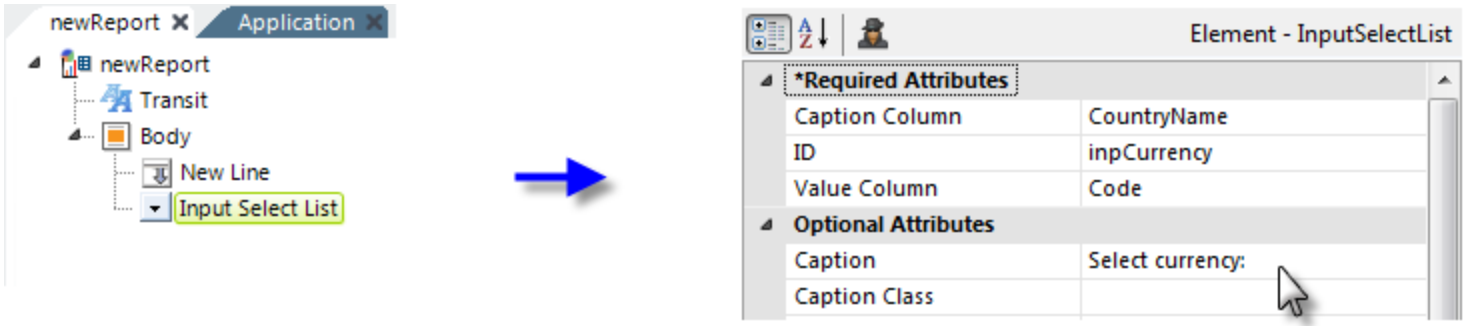
Using XML File Data in an Input Select List

The **DataLayer.XML** element is used to retrieved from an XML file. This approach is useful when there is a substantial amount of data and, though generally static, it may change someday and/or being able to edit the data outside of the report definition is desirable. In this example, the user will be able to select a currency from a drop-down list.

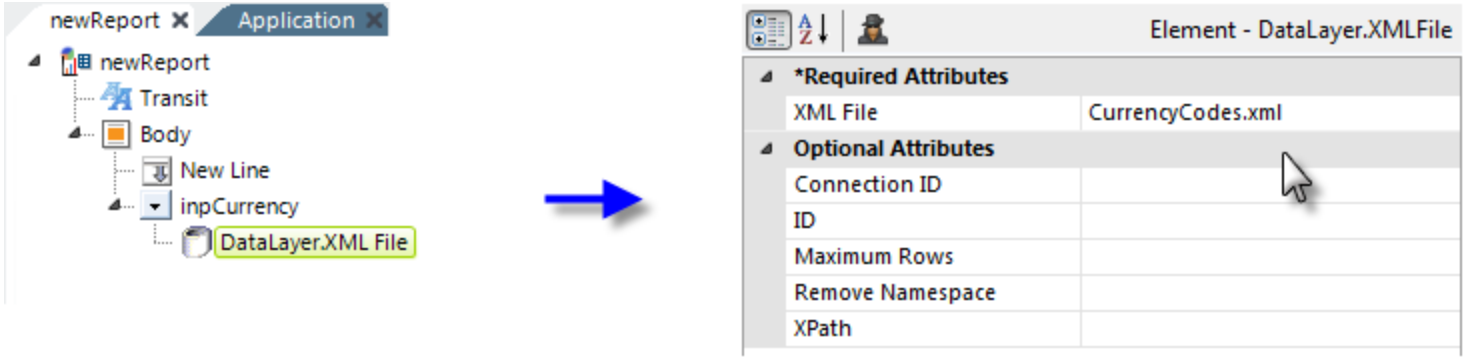
CurrencyCodes.xml | newReport | Application

```
<?xml version="1.0" encoding="utf-8"?>
<rdData>
  <Currency CountryName="AFGHANISTAN" CurrencyName="Afghani" Code="AFN" />
  <Currency CountryName="ÅLAND ISLANDS" CurrencyName="Euro" Code="EUR" />
  <Currency CountryName="ALBANIA" CurrencyName="Lek" Code="ALL" />
  <Currency CountryName="ALGERIA" CurrencyName="Algerian Dinar" Code="DZD" />
  <Currency CountryName="AMERICAN SAMOA" CurrencyName="US Dollar" Code="USD" />
  <Currency CountryName="ANDORRA" CurrencyName="Euro" Code="EUR" />
  <Currency CountryName="ANGOLA" CurrencyName="Kwanza" Code="AOA" />
  <Currency CountryName="ANGUILLA" CurrencyName="East Caribbean Dollar" Code="XCD" />
```

The currencies are contained in CurrencyCodes.xml, part of which is shown above, and which should be added to the application's **Support Files**.



1. Add the **Input Select List** element to the definition, as shown above.
2. Beneath it, add a **DataLayer.XML** element.
3. Set the Input Select List element's attributes as shown above. The values for the **Caption Column** attribute and **Value Column** attribute correspond to the attribute names in the XML file.



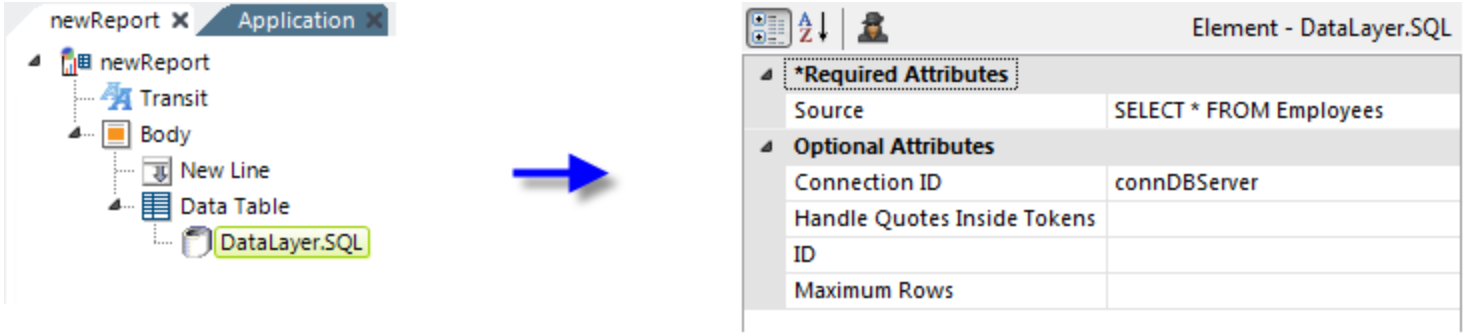
4. Set the **DataLayer.XML** element's attributes as shown above (assumes the file is in the _SupportFiles folder.)

When the report is run, the datalayer will read all of the values from the XML file and the Input Select List will display them. If this report calls another report or process, based on the attributes set above, in that report the @Request.inpCurrency~ token will contain the value from the XML file's **Code** attribute. So, selecting the Australian Dollar in the select list will result in the value "AUD" being passed to the next report.

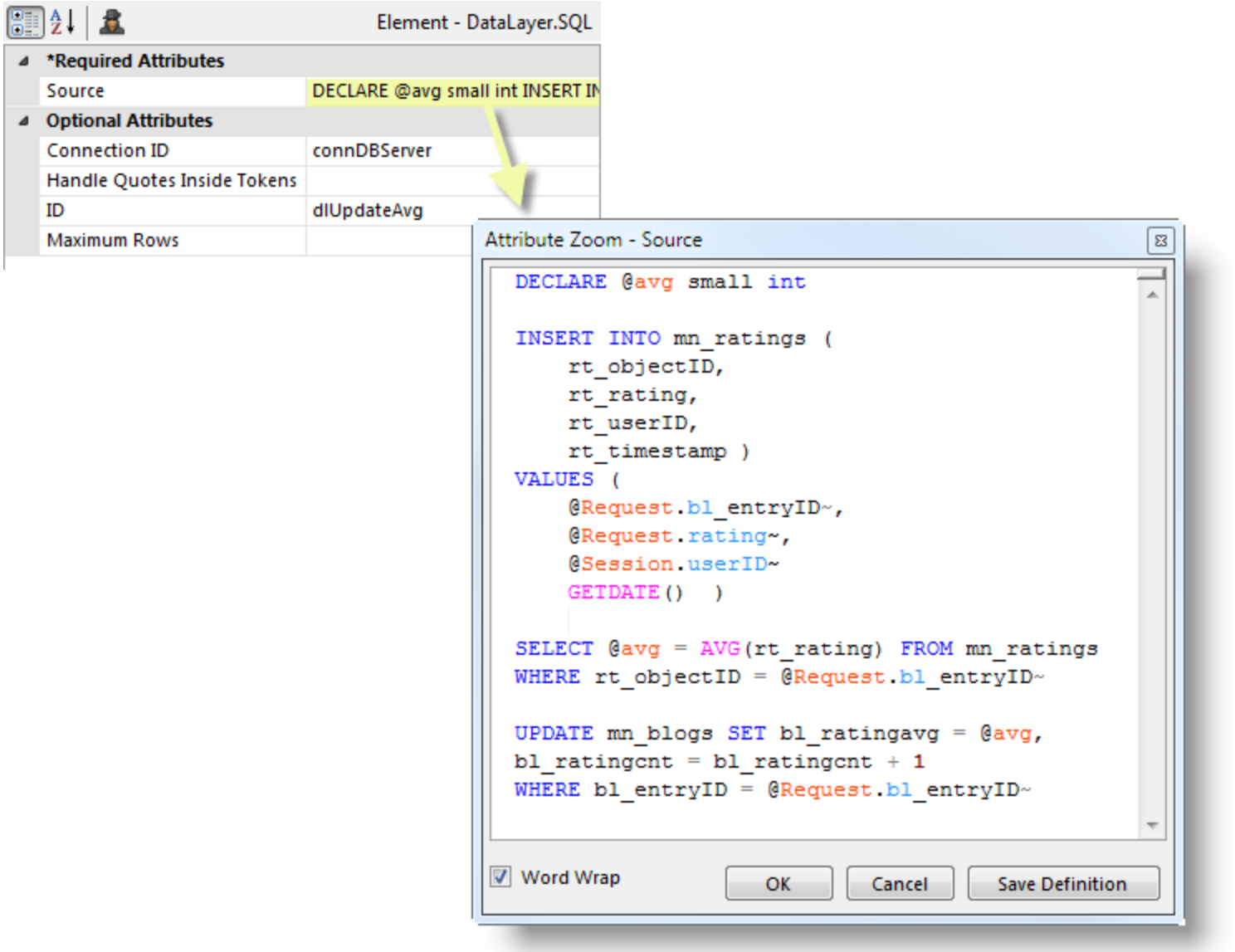
Using Datalayers to Retrieve SQL Data

One of the most common usages of the datalayer is to retireve data from a database. The next two examples show how to use a datalayer to retrieve data from a SQL database and they assume an appropriate connection element for the database server has already been configured. See *Datasource Connections* for more information.

This example uses **DataLayer.SQL** to run a query. The datalayers's **Source** attribute value contains the actual SQL statement to be executed and any valid SQL code can be used. This example uses a connection to a Microsoft SQL Server database.

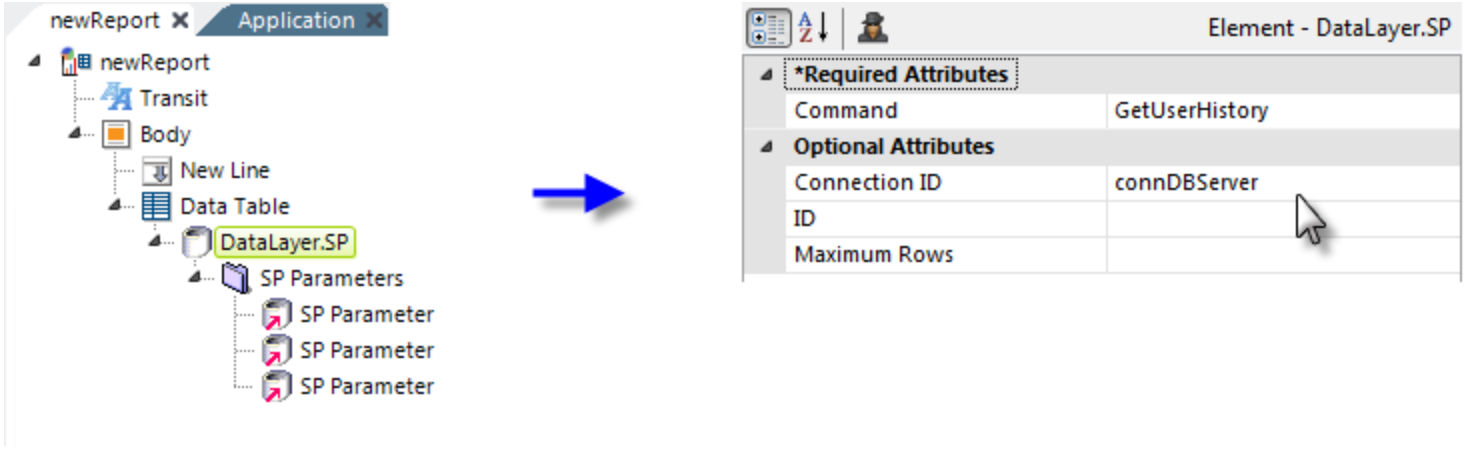


The example above shows a simple SQL query that returns data to a simple Data Table.

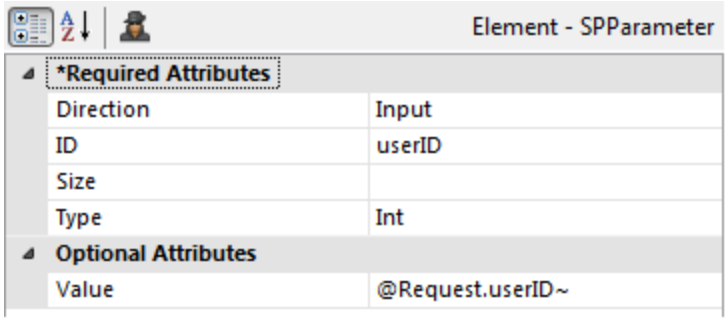

Element - DataLayer.SQL
*Required Attributes
Source
DECLARE @avg small int INSERT IN
Optional Attributes
Connection ID
connDBServer
Handle Quotes Inside Tokens
ID
dlUpdateAvg
Maximum Rows
Attribute Zoom - Source
DECLARE @avg small int
INSERT INTO mn_ratings (
rt_objectID,
rt_rating,
rt_userID,
rt_timestamp)
VALUES (
@Request.bl_entryID~,
@Request.rating~,
@Session.userID~
GETDATE())
SELECT @avg = AVG(rt_rating) FROM mn_ratings
WHERE rt_objectID = @Request.bl_entryID~
UPDATE mn_blogs SET bl_ratingavg = @avg,
bl_ratingcnt = bl_ratingcnt + 1
WHERE bl_entryID = @Request.bl_entryID~
Word Wrap
OK
Cancel
Save Definition

However, as shown in the example above, the **Source** attribute value (opened in the Attribute Zoom Window) can consist of *multiple* SQL statements. They will all be executed, as long as the syntax is correct, and any results will be returned to the datalayer. The Attribute Zoom window is opened for any attribute by double-clicking the attribute name. More information is available in "DataLayer.SQL" on page 146.

While the multi-statement capability shown above makes it easy to use complex SQL queries, the *best performance* and the *best security* for them is achieved by using the **DataLayer.SP** element to run a **stored procedure**.



In the example shown above, **DataLayer.SP** is used to call a stored procedure. Using stored procedures provides the best protection against a SQL Injection attack.



An **SP Parameters** element, and one **SP Parameter** element for each parameter, are used to pass arguments to the stored procedure and to receive output values. The result set from the stored procedure is retrieved into the datalayer.

The *order* of the SP Parameter elements, not their IDs, is used to determine their correlation to the declared variables in the stored procedure code. More information is available in "DataLayer.SP" on page 126.

Filter Datalayers

Datalayer elements retrieve data from datasources and developers frequently need to apply conditioning to them to exclude data that is unnecessary or unwanted. This filtering is 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. This topic discusses use of datalayer filtering elements.

The following topics provide additional information about filtering datalayers:

- Applying a Static Filter
- Applying a Dynamic Filter

About Filtering Elements

Filtering elements are added to Logi definitions as children of datalayer elements. Their purpose is to *remove rows* from the datalayer based on some criteria. These elements include:

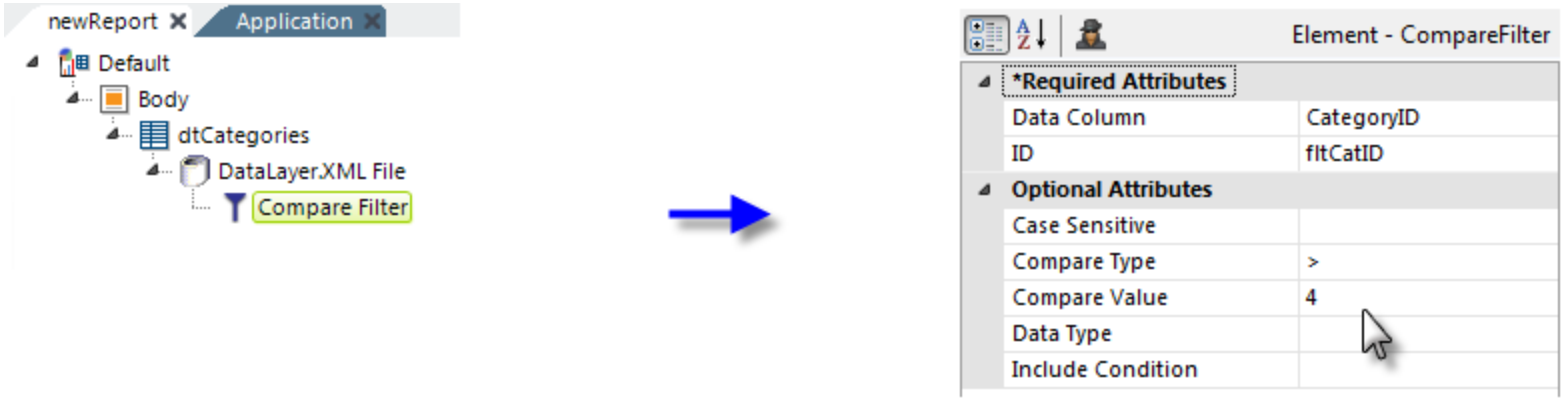
Element	Description
Analysis Filter	This super-element adds a complete user interface, configurable from simple to complex, for creating and managing real-time datalayer filtering. It's consistent with the filtering interface now found in other super-elements like the Analysis Grid. For more information, see *The Analysis Filter*.
Compare Filter	Analogous to the *WHERE*clause in a SQL query, this element removes rows from the datalayer that do not meet specified criteria. Uses native data engine code for fastest performance.
Condition Filter	Similar to the Compare Filter, this element uses scripting to remove rows from the datalayer that do not meet specified criteria.

Element	Description
Contain Filter	Analogous to using a *WHEREx CONTAINS y*clause in a SQL query, this element removes rows from the datalayer that do not return results in a text search of specified columns.
DeDuplicate Filter	Analogous to using a *DISTINCT* clause in a SQL query, this element removes rows from the datalayer that have duplicated values in specified columns.
RegEx Filter	Removes rows from the datalayer by applying pattern matching using regular expressions.
Relevance Filter	Removes rows (*irrelevant*data) from the datalayer that do not meet a threshold; optionally, irrelevant rows can be retained, grouped together, and handled as an individual entity.
Security Filter	Removes rows from the datalayer, like a Condition Filter, but is applied based on a user's security rights.

It's possible to use multiple filters, of the same type or in mixed combinations, to achieve the desired filtering effect.

Applying a Static Filter

The following example illustrates use of the **Compare Filter**, with static filtering criteria:



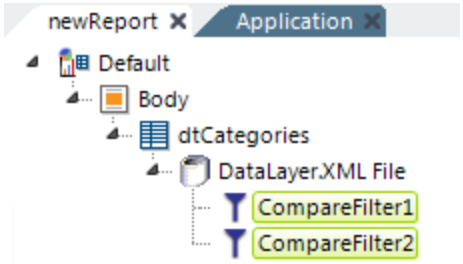
1. As shown above, a **Compare Filter** element is added as a child to a datalayer element.
2. Its **Data Column** attribute value is set to the column in that datalayer whose value will be compared.
3. The **Compare Type** attribute is set to define the comparison operator. A number of standard operators are provided in a pull-down list and you may also type-in your own operators, such as *!=* (not equal).
4. Finally, the **Compare Value** attribute is set to the *static*, or literal, value the data column values will be compared against. Tokens may be used here. Ensure that date values used here are specified in the *yyyy-mm-dd* format.

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.

Applying a Dynamic Filter

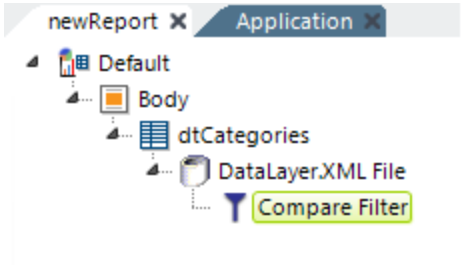
This topic provides an example that illustrates use of the Compare Filter, with dynamic filtering criteria.

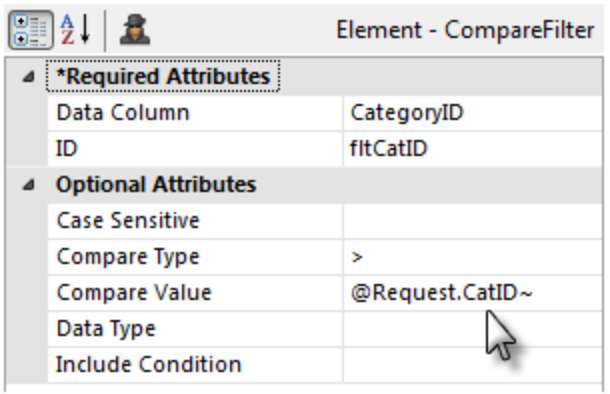
1. Developers can use two or more **consecutive** Compare Filter elements:



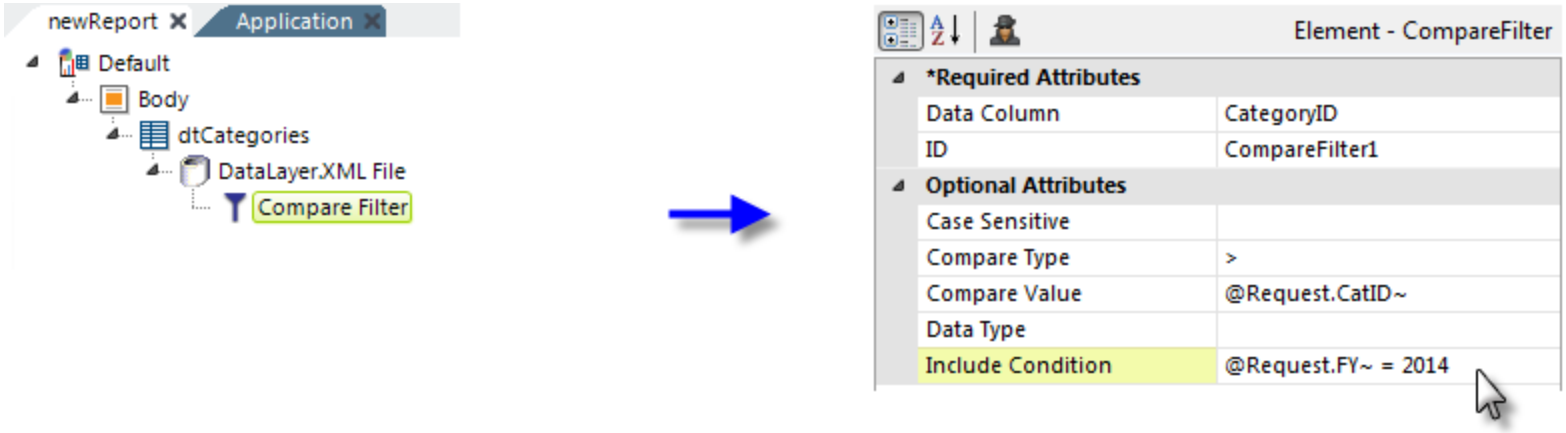
In this case, the comparisons will be evaluated **sequentially**, based on their arrangement from top to bottom in the element tree. So, after the first Compare Filter is applied, only records that meet its criteria will remain in the datalayer to be evaluated for the second Compare Filter.

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





As shown above, 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. 3. 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.

Join Datalayers

The **Join** element provides the functionality of a SQL-style join, allowing developers to combine and relate data from different datasources.

About Joins

The traditional SQL SELECT statement can include an optional **JOIN** clause, which **combinesrecords** from two or more tables in a database. The combining action may be predicated on common values in the columns of the tables. The SQL query that contains a JOIN returns a result set that includes the combined data.

Logi products offer a **Join** element that operates in the same way on the data retrieved into datalayers in Logi reports. The effect of using this element is to create a datalayer that contains the combined data. One of the interesting features of the Join element is that it's not restricted to SQL datasources, so it can perform the joining action with **non-SQL datasources** as well, or between databases that don't support the same JOIN syntax.

Here's an example of a JOIN that combines two tables, based on their common Order ID values:

OrderID	CustomerID	OrderDate	ShipName
10248	VINET	7/4/1996	Vins et alcools Chevalier
10249	TOMSP	7/4/1996	Toms Spezialitäten
10250	HANAR	7/8/1996	Hanari Carnes
10251	VICTE	7/8/1996	Victuailles en stock
10252	SUPRD	7/9/1996	Suprêmes délices
10253	HANAR	7/10/1996	Hanari Carnes
10254	CHOPS	6/3/1996	Chop-suey Chinese
10255	RICSU	7/12/1996	Richter Supermarkt
10256	WELLI	7/15/1996	Wellington Importadora
10257	HILAA	7/16/1996	HILARION-Abastos

OrderID	ProductID	UnitPrice	Quantity	Discount
10248	11	14.0000	12	0
10248	42	9.8000	10	0
10248	72	34.8000	5	0
10249	14	18.6000	9	0
10249	51	42.4000	40	0
10250	41	7.7000	10	0
10250	51	42.4000	35	0.15
10250	65	16.8000	15	0.15
10251	22	16.8000	6	0.05
10251	57	15.6000	15	0.05

JOIN on OrderID

OrderID	CustomerID	OrderDate	ShipName	ProductID	UnitPrice	Quantity	Discount
10248	VINET	7/4/1996	Vins et alcools Chevalier	11	14.0000	12	0
10248	VINET	7/4/1996	Vins et alcools Chevalier	42	9.8000	10	0
10248	VINET	7/4/1996	Vins et alcools Chevalier	72	34.8000	5	0
10249	TOMSP	7/4/1996	Toms Spezialitäten	14	18.6000	9	0
10249	TOMSP	7/4/1996	Toms Spezialitäten	51	42.4000	40	0
10250	HANAR	7/8/1996	Hanari Carnes	41	7.7000	10	0
10250	HANAR	7/8/1996	Hanari Carnes	51	42.4000	35	0.15
10250	HANAR	7/8/1996	Hanari Carnes	65	16.8000	15	0.15
10251	VICTE	7/8/1996	Victuailles en stock	22	16.8000	6	0.05
10251	VICTE	7/8/1996	Victuailles en stock	57	15.6000	15	0.05

As you can see, the records from both tables are combined into a single table using the **Order ID** column as the key to matching rows from one table to rows from the other table. In a Logi application that uses a Join element, this occurs in the datalayer, and the resulting data is available using the usual @Data tokens.

The example shown above uses an **Inner Join**, one of four types of joins available using the Join element. The Inner Join is the most common join operation used in applications; it creates a new result table by combining column values of two tables based upon a "match condition". The match condition in the example was that the Order ID value in one table equals the Order ID value in the other table.

The **Left Outer Join**produces a result that includes *all* of the records of the "left" table, plus matched values from the "right" table. NULLs are filled-in when values from the right table don't match the match condition.

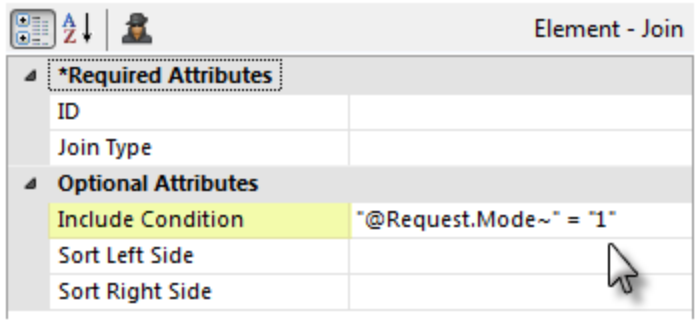
The **Full Outer Join** produces a result that contains all records from both tables, and fills in NULLs for missing matches on either side.

Finally, the **Union Join** simply builds a result that includes all rows and columns from both source tables, without trying to do any matching. NULLs are filled-in for columns in any row that don't have a value.

More information is available here about SQL joins.

Dynamic Joins

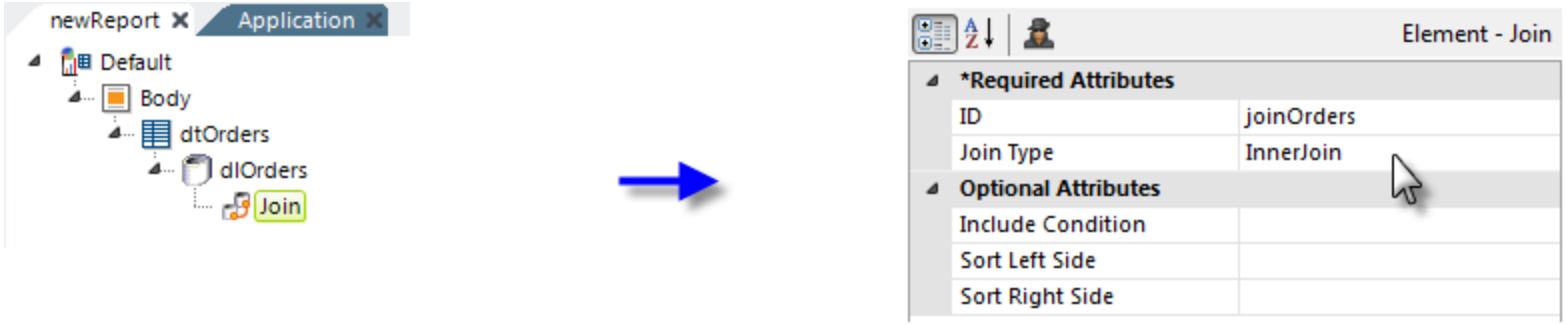
The Join element has an **Include Condition** attribute:



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 datalayers will be joined or not.

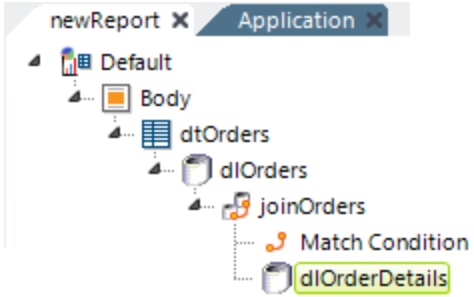
Joining Datalayers

Here's how the join in the example above was created:

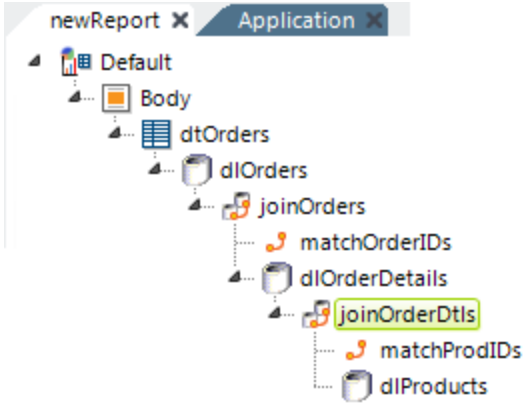


In the example, we wanted to display our data in a table, so a **Data Table** element, with a supporting datalayer element beneath it, was used. This datalayer retrieves data from the Orders table. Then a **Join** element was added beneath the datalayer, as shown above. Its attributes were set to give it a unique ID, and the *InnerJoin* type was selected.

Next, a **Match Condition** element was added beneath the Join element. This element is used to identify the columns in the joined tables that will determine which records are selected, or "matched". The **Left Data Column** attribute identifies the column in the left table, the **Right Data Column** attribute, the one in the right table. In the example, the column names were the same. You can use tokens to specify the column names.



Finally, a **second datalayer** is added, which retrieves records from the Order Details table. And that's all there is to it. When the report runs, all of the joined data is available to the Data Table using the usual @Data tokens.



You can have many joins in a report definition. So, you could even extend the example by adding another join, as shown above, on the Products table. One of the default steps of the join process is sorting the data before the join, and this can take time. If the data being retrieved into the datalayers is already sorted, skipping the join process sort can improve performance. You can skip the sort by setting the Join element's **Sort Left Side** or **Sort Right Side** attributes to *False*.

Debugging Joins

Joins can get tricky very quickly, especially if you have NULL data in your datasources. It's very useful to be able to see what's actually going in each step of the join process and this can be done using the **Debugging Trace Report**. As you may know, debugging can be turned on using the **Debug** icon in Studio's toolbar and in the _Settings definition by setting the **General** element's **Debugger Style** attribute to *DebuggerLinks*.

Logi Debugger Trace Report

Event	Event Detail	Value	Time *
Get XML DataLayers			.052
Get DataLayer - SQL	ID	dlOrders	.052
	Connection Type	SqlServer	.053
	Source	SELECT * FROM Orders	.053
	Processing SQL	Sending SQL command to data source.	.059
	SQL Executed	Start streaming records into engine.	.062
	Starting Read	Retrieved first row.	.062
	Streaming Completed	10 rows of data read into the engine.	.092
	DataLayer for Element dtOrders	View File Stream Data (3,843 bytes)	.094
	Run Join. ID=joinOrders		.105
	Join XPath Match Condition	rdData/joinOrders[@OrderID='@Data.OrderID~']	.105
	Run Join DataLayer.		.105
Get DataLayer - SQL	ID	dlOrderDetails	.105
	Connection Type	SqlServer	.106
	Source	SELECT * FROM [Order Details]	.107
	Processing SQL	Sending SQL command to data source.	.110
	SQL Executed	Start streaming records into engine.	.116
	Starting Read	Retrieved first row.	.116
	Streaming Completed	10 rows of data read into the engine.	.126
	DataLayer for Element joinOrders	View File Stream Data (1,029 bytes)	.128
	Run Join. ID=joinProducts		.140
	Join XPath Match Condition	rdData/joinProducts[@ProductID='@Data.ProductID~']	.140
	Run Join DataLayer.		.140

Get DataLayer - SQL	ID	dlProducts	.141
	Connection Type	SqlServer	.141
	Source	SELECT * FROM Products	.141
	Processing SQL	Sending SQL command to data source.	.144
	SQL Executed	Start streaming records into engine.	.146
	Starting Read	Retrieved first row.	.146
	Streaming Completed	77 rows of data read into the engine.	.148
	DataLayer for Element joinProducts	View File Stream Data (17,332 bytes)	.153
	Moving Processed Data	Starting move.	.162
	Moving Processed Data	77 rows of data moved.	.164
	Data Engine	Data Processing completed.	.167
	Joining rows.		.168
	Join	View Memory Stream Data (2,769 bytes)	.193
	Moving Processed Data	Starting move.	.200
	Moving Processed Data	10 rows of data moved.	.200
	Data Engine	Data Processing completed.	.201
	Joining rows.		.202
	Join	View Memory Stream Data (6,176 bytes)	.220
	Moving Processed Data	Starting move.	.229
	Moving Processed Data	10 rows of data moved.	.230
	Data Engine	Data Processing completed.	.230
Generated new DataLaye	Memory stream	View Data	.231
	DataLayer Info XML	View Data	.238
	Save data in cache with key:8664502	View Data	.251

A truncated version of the Debugger Trace Report is shown above. The highlighted areas indicate where the datalayers are run, and where the joining occurs (based on the last element tree example, with two joins). You can look at the data at each step of the way, using the links provided, to see how the retrieved data comes in and what happens to it when the joins are applied. This can be very useful in understanding what's happening to the data "behind the scenes". The time scale is also useful in determining whether or not a join is creating a performance bottleneck.

Link Datalayers

When datalayers run, they *retrieve* and *cache* data. Performance may be improved by minimizing the re-running of datalayers, especially if long-running queries are involved, by allowing a datalayer to *share* data that's already been cached by another datalayer.

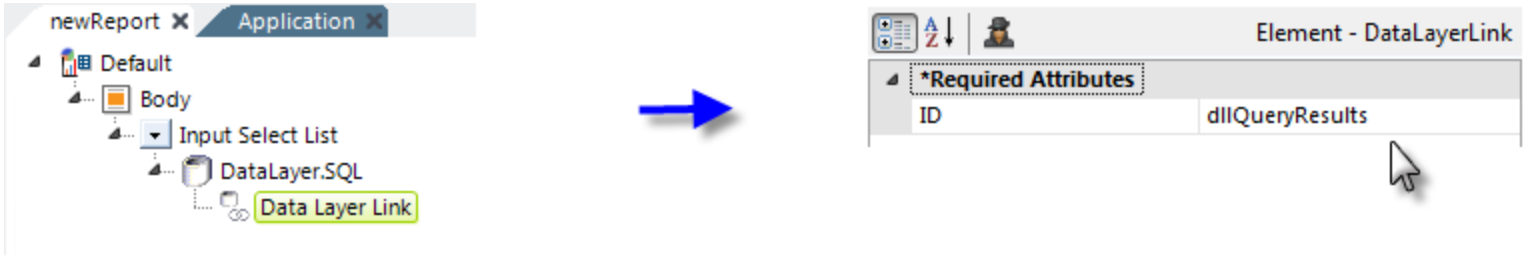
Developers are always looking for ways to improve performance, particularly where database queries are concerned. You can improve the performance of your Logi applications by eliminating repeatedqueries of the same data, through use of a feature that allows data retrieved into one datalayer to be saved and re-used elsewhere within the application. Two elements are used in this process: the first one, **Data Layer Link**, identifies the data to be saved and made available; and the second one, **DataLayer.Linked**, provides a way to access and re-use that data.

The following topics discuss the technique of linking datalayers:

- Making a DataLayer Link Available
- Linking Within the Same Definition
- Linking Across Different Definitions

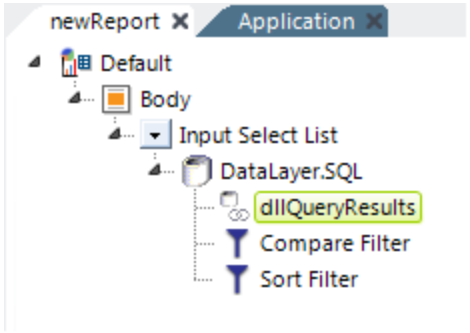
Making a DataLayer Link Available

The following example illustrates how to make the data in a datalayer re-usable:

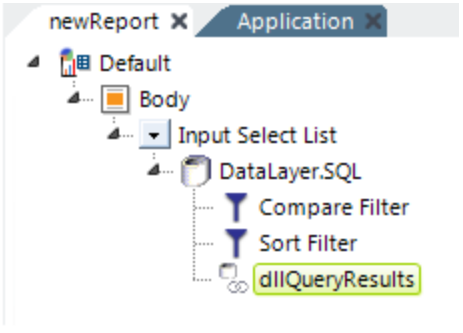


1. As shown above, the example definition contains an **Input Select List** and **DataLayer.SQL** elements. The datalayer uses a regular SQL query to retrieve data from the database to present as options in aselection list.
2. Beneath the datalayer, a **Data Layer Link** element has been added. The effect of the element is to "save" a copy of the data for future use.

You *must* give the element a *unique* ID.

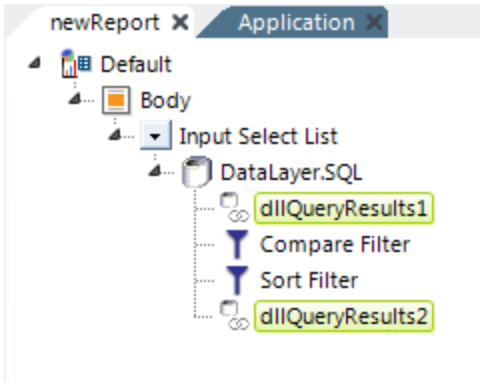


Raw data is available for re-use



Filtered, sorted data is available for re-use

If you use elements to filter, sort, and otherwise manipulate the data in the datalayer, then the *position*of the Data Layer Link element in the element tree is significant. For example, above left, the *raw* data, as it was retrieved from the database, will be available for re-use. But, above right, because the link comes after (below) the other elements, the *filtered* and *sorted* data will be available.

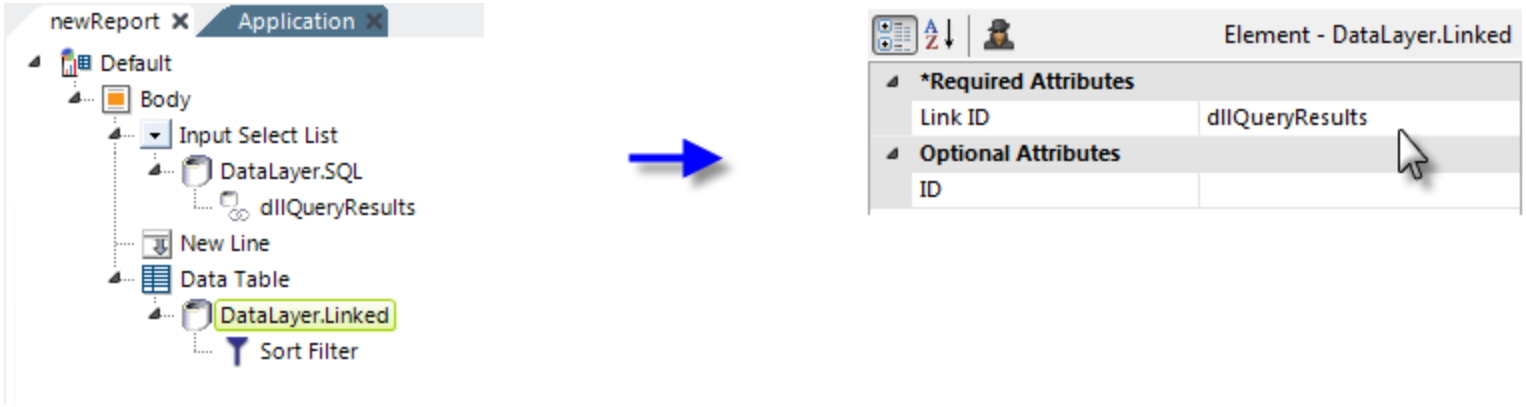


You can even use *multiple* Data Layer Link elements beneath a datalayer. In the example above, *both* the raw and the filtered, sorted data would be available for re-use, for different consumers.

The data in the datalayer is now available for re-use, saving us the performance load of running one or more queries later in our application.

Linking Within the Same Definition

When we have a need to re-use the data from the datalayer, we can do so using the **DataLayer.Linked** element:



1. As shown above, in the same definition, a Data Table has been added and we'd like to use it to present some of the same data retrieved earlier for the selection list.
2. A **DataLayer.Linked** element has been added beneath the Data Table.
3. Its **Link ID** attribute has been set to the ID of the Data Layer Link element we used earlier. This makes the data retrieved earlier available for use in the table.
4. Linked data can subsequently be manipulated using standard elements, such as the Sort Filter.

That's all there is to it. Multiple DataLayer.Linked elements can be used in a definition; because each one represents the original data, sorts and filters applied to one will not affect the data in another.

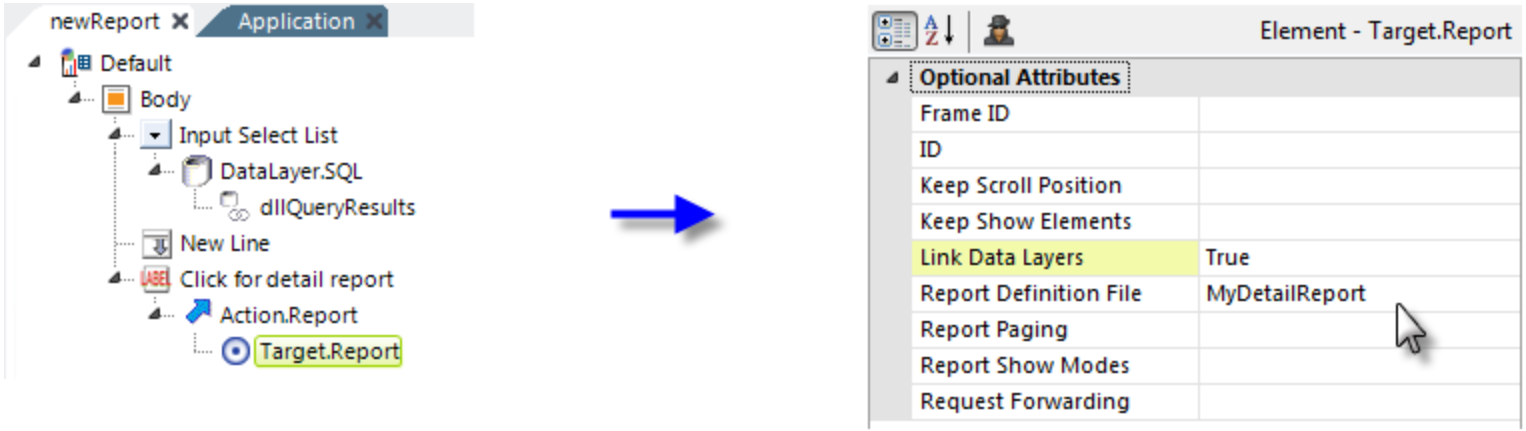
This technique is especially useful when presenting the same data in *different ways* in a single report; for example, by using a table and a chart of some kind. Logi charts can use DataLayer.Linked as their datalayer and re-use the data retrieved for the table.

Due to order-of-operations differences, datalayers that have the Handle Quotes Inside Tokens attribute set to *True* will not process quotes as expected when used underneath Chart Canvas elements. In this scenario, we recommend that you use the datalayer under Local Data and link it, using the techniques described in this topic, to a datalayer for the chart.

A datalayer used to retrieve data for a chart *cannot* be linked later in the definition for use with a Data Table. This is because the data retrieved into a datalayer for a chart is processed somewhat differently than data retrieved for a table. However, the reverse usage, a datalayer beneath a Data Table linked to a datalayer beneath a chart later in the definition is okay. Some developers, faced with this restriction, prefer to use a **Local Data** element to retrieve the data, then link its datalayer to both a chart and Data Table later in the definition.

Linking Across Different Definitions

We've seen that data can be re-used by linking datalayers within the *same* definition. They can also be re-used in the same manner *across different* definitions. This is done by making the definition that wants to re-use the data a "target" of the definition that originally retrieves the data. The target report then includes a DataLayer.Linked element.



1. As shown above, the "parent" definition has been modified to include **Label**, **Action.Report**, and **Target.Report** elements. This makes the label text become a link that displays the next ("detail") report.
2. To make the re-usable data available in the detail report, the Target.Report element's **Link Data Layers** attribute has been set to *True,* as shown above, right.
3. In the detail report definition, a **DataLayer.Linked** element is used in the same manner shown in "Linking Within the Same Definition" on page 65 in order to access the re-usable data.

You can only share data using this technique among definitions within the *same* Logi application. You cannot share it with other Logi applications.

If you're considering opening your child report in a new window, keep in mind that linked data is only available in the *same session*. For example, if you specify "New Window" in your Target element's **Frame ID** attribute and run the parent report inside Studio's Preview window, the linked datalayer won't be found. This happens because the new window opened for the child report will be outside of Studio and will start its own new session.

However, under most default browser configurations, when you run the parent report in a browser window, the new window for the child report will be opened in new *tab*, which uses the same session, and the linked datalayer will be found. But, if your browser is configured to open a new *window* for each new URL and that causes a new session to be started, then the linked datalayer will not be found by the child report.

The ability to link datalayers within and across definitions is a powerful feature in Logi applications and a good technique for developers who want to reduce the number of datalayer queries they need to run.

DataLayer.CSV

Developers often need to read data from text files in Comma-Separated Values (CSV) format and Logi Info includes the **DataLayer.CSV** element for this purpose.

The following topics discuss this datalayer:

- DataLayer.CSV Attributes
- Working with DataLayer.CSV
- Using Studio's DataLayer Wizard

DataLayer.CSV - Attributes

A **DataLayer.CSV** element is added as a child element to a Data Table or other data container element. Its attributes are set so that it accesses the desired .CSV data file, and so it handles column names as desired. The datalayer reads and caches the data from the .CSV file. You can add child elements beneath the datalayer to affect its contents, including:

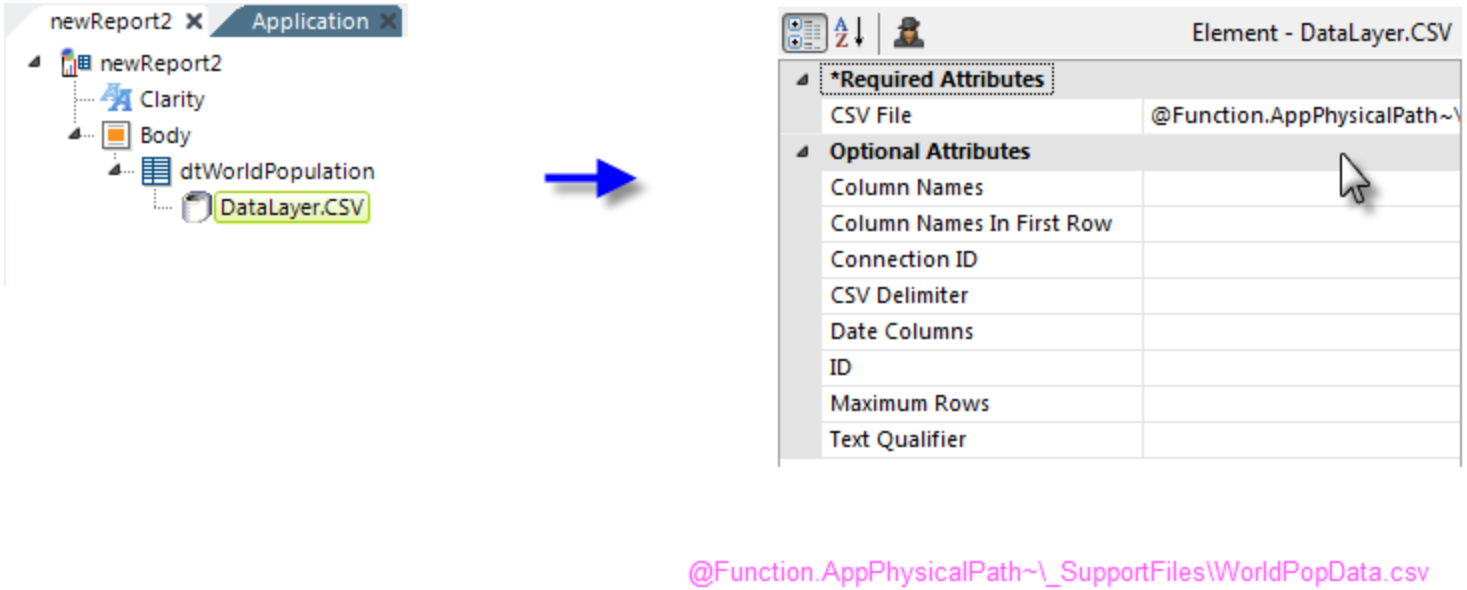
The DataLayer.CSV element has the following attributes:

Attribute	Description
CSV File	(Required) Specifies the fully-qualified file system **path** and **filename** for the data file. The token @Function.AppPhysicalPath~ can be used to get the application folder path. Example: `@Function.AppPhysicalPath~\SampleData\Contacts.csv` May also be a complete URL to a file on another web server. Example: `http://www.example.com/data/records.csv`
Column Names	If left blank and if the **Column Names In First Row** attribute is set *False* or is blank, columns retrieved into the datalayer will be assigned default names in the form: Column1, Column2, Column3, etc. If, however, an optional comma-separated list of **custom** datalayer **column names** is specified here, they will override the default names. Or, if the **Column Names In First Row** attribute is set *True*, valid text names from the **first data row** will automatically be assigned as the datalayer column names, overriding the default names.
Column Names In First Row	Specifies if the text in the **first data row** should be used as the column names in the datalayer. The default value is *False*.

Attribute	Description
Connection ID	Specifies a connection to a data source that's defined in the _Settings definition.
CSV Delimiter	Specifies the **character** used to **delimit** the columns of data in the .CSV file. You may use *t* = Tab, *v* = Vertical Tab, *r* = Carriage Return, *n* = Line Feed, or *Space* = blank space. The default character is a *comma*.
Date Columns	Specifies a comma-separated list of column names of those columns that should be formatted as **DateTime** data. The names entered here should match those specified using earlier attributes for the **datalayer** column names, e.g. these may be the default names: Column1, Column5, etc. or the custom names specified in the Column Names attribute, or the names read from the first row of data.
ID	Specifies an unique element ID. Recommended for easier identification when debugging.
Maximum Rows	Specifies the maximum number of rows to retrieve from the data source.
Text Qualifier	Specifies the **character** used to **qualify** data that is to be handled as text. The default character is the double-quote. When data is read into the datalayer, the text qualified character is removed from the beginning and end of a column (and should therefore always occur in pairs). If the qualifier character is *doubled* within the column, it will be replaced with a single instance of the character. For example, the text `'Dave''s Rail Car'` in the file becomes `Dave's Rail Car` in the datalayer.

DataLayer.CSV - Working with DataLayer.CSV

In most respects, **DataLayer.CSV** functions exactly as other datalayer elements do and its data can be filtered and conditioned using appropriate elements. One major difference, however, is that *there is no need to use a Connection element in the _settings definition with this element.* The DataLayer.CSV element directly accesses the file system to read .CSV files, if a file path is specified.



@Function.AppPhysicalPath~_SupportFiles\WorldPopData.csv

- **Filtering**: Sort, group, or restrict the data
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled data values
- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report definitions

Data read into the datalayer is cached in XML format in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

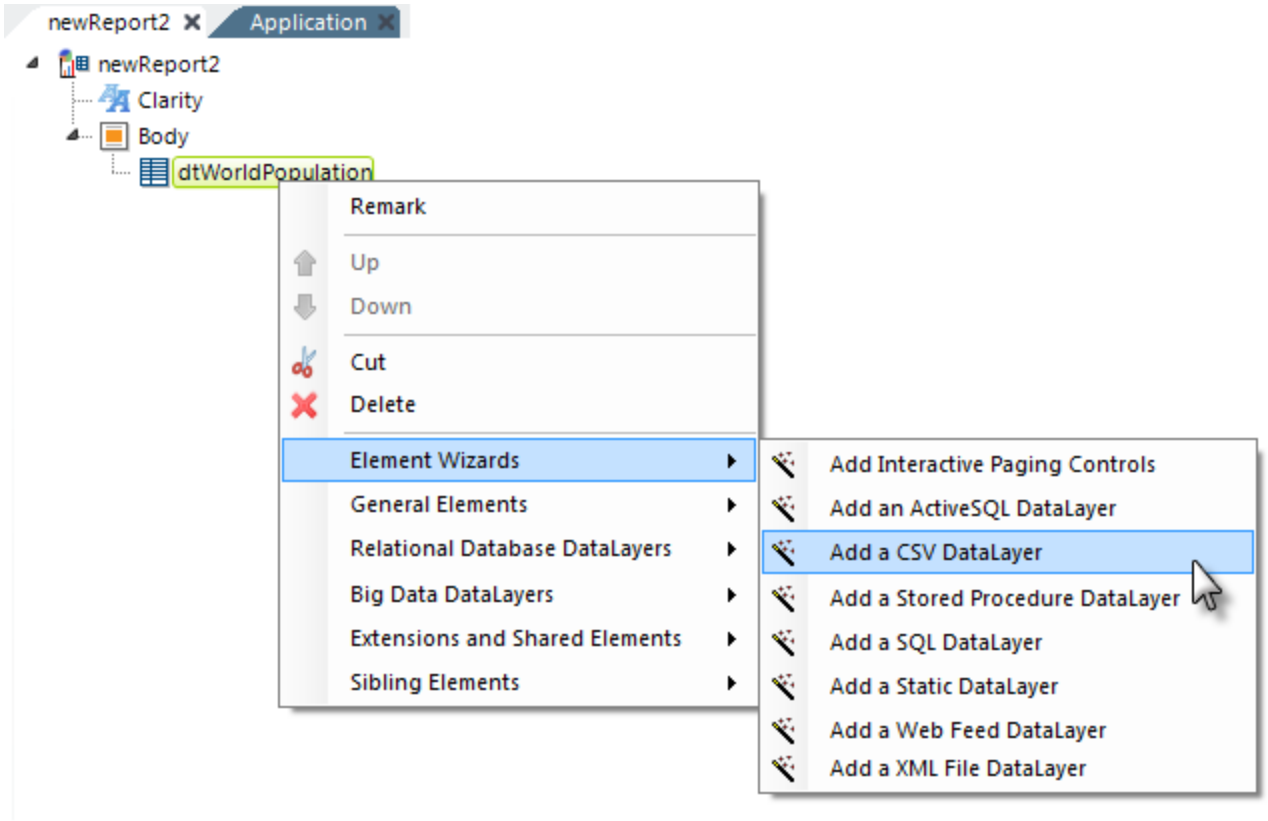
The data read with the datalayer is available using **@Datatokens**, in the format @Data.*ColumnName*~. The spelling of the column name is **case-sensitive**. The data is only available within the scope of the parent element of the datalayer, not throughout the entire report definition. The DataLayer.Linked element can be used to make the data reusable in another datalayer outside this scope.

The *Auto Columns* element can be used to quickly display in your report all the data in a datalayer.

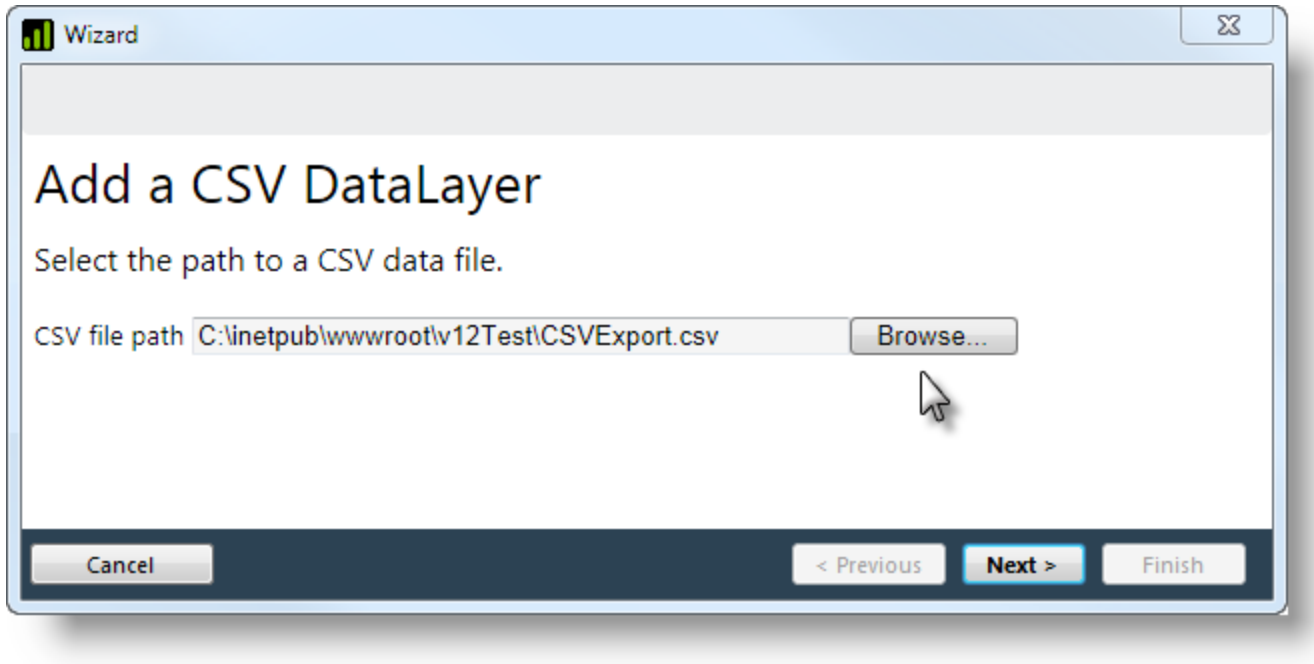
The data retrieved into the datalayer can be viewed by turning on the Debugging Link in your _Settings definition (**General** element) and using the resulting link at the bottom of your report page to view the **Debugger Trace** page. An link on the Trace page can be clicked to display the retrieved data.

DataLayer.CSV - Using Studio's DataLayer Wizard

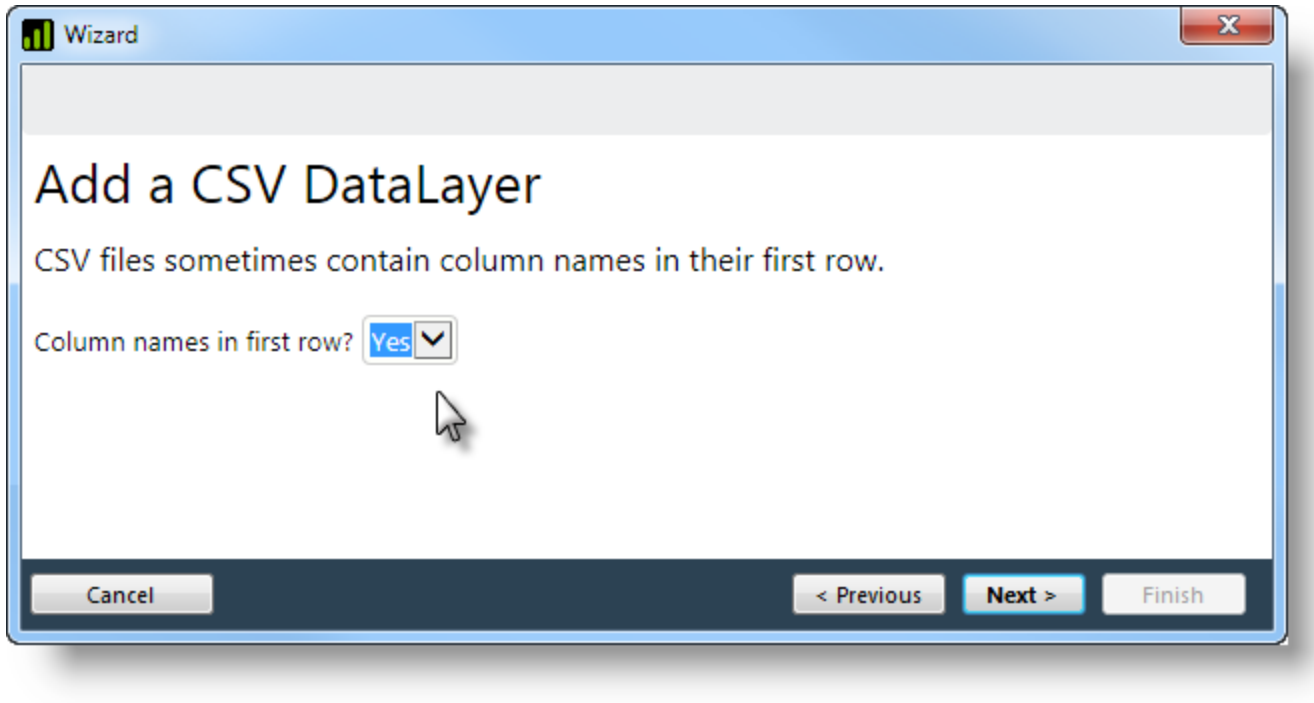
Logi Studio includes a wizard that can assist you in configuring DataLayer.CSV.



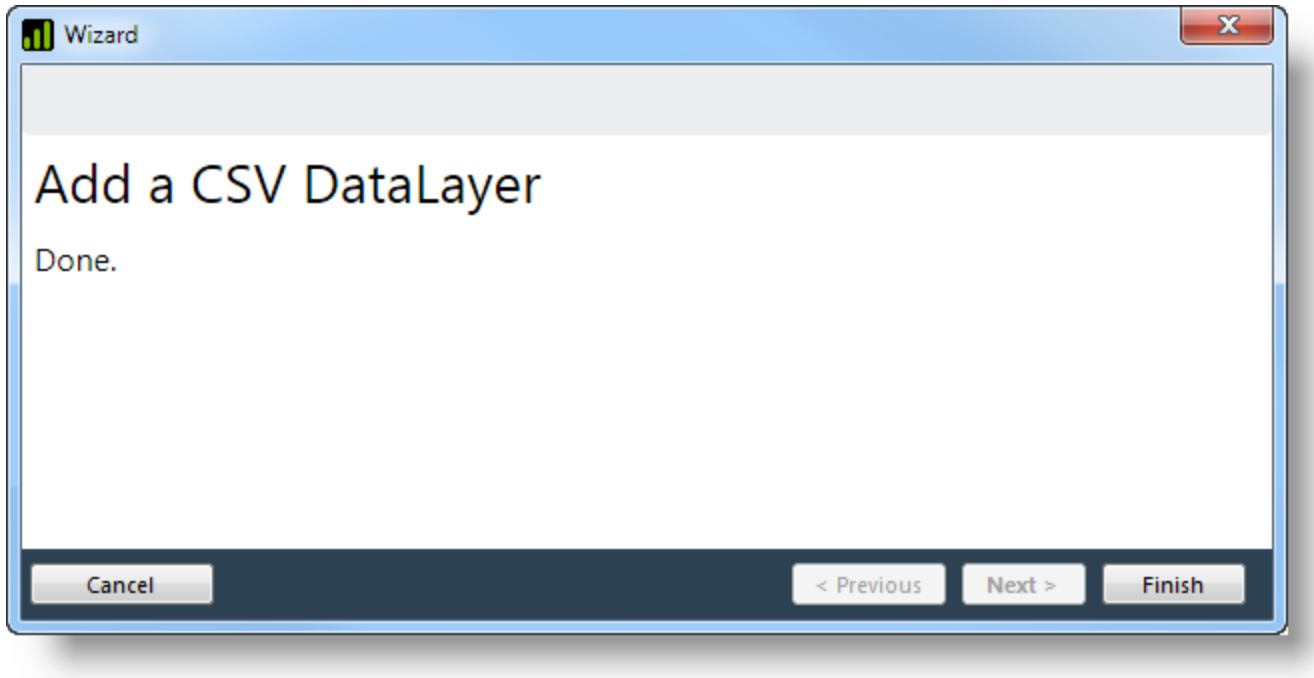
As shown above, the wizard can be started by selecting and then right-clicking the parent element under which you want to add the datalayer, and using the context menus to select "Add a CSV DataLayer". The wizard will open; use it as follows:



1. Browse to the fully-qualified path and filename of the CSV file to be used. This may be an MS Excel file. Click **Next** to continue.



2. Indicate whether the file has column names in the first row, by selecting *Yes* or *No*. Click **Next** to continue.



3. The datalayer element will be inserted. Click **Finish** to exit the wizard.

4. The wizard will insert the datalayer and configure minimal attributes for it. Other optional attributes have to be configured manually.

DataLayer.Directory

The **DataLayer.Directory** element provides developers with information about the files and folders in a specified directory.

The following sections are covered in this topic:

About DataLayer.Directory

DataLayer.Directory provides the directory information for a specific folder and/or its contents. The results, in the usual datalayer tabular format, includes information such as file name, type, size, and creation date.


This datalayer returns the directory information stored in the file system for a single folder; it does not *search* folders for files nor traverse sub-folders.

To get the information for multiple sub-folders, you can use several of these elements together under a Data Table, for example, and their results will be automatically joined together (Union) to produce a single data set. This is useful when the names of the sub-folders to be examined are known in advance.

Attributes

The DataLayer.Directory element has the following attributes:

Attribute	Description
ID	Specifies an optional element ID. Recommended for easier identification when debugging.
Directory Filter	Specifies a filters for the results, based on a complete filename or a filename that includes the standard "wild card" characters (* and ?). The default value is: *.*

Directory Folder	Specifies a fully-qualified file path for a folder to be read. Tokens may be used here. Examples:
	
	`C:\inetpub\wwwroot\yourLogiApp\_Themes`
	`@Function.AppPhysicalPath~\_Definitions\_Reports`
	The asterisk wild card character ("*") may be used here to specify folder names. The default value is your application's `rdDataCache` folder.

Directory Type	Specifies the type of directory information to return: *FilesAndFolders* = (Default) Returns information about both files and folders. *FilesOnly* = Returns information about files only. *FoldersOnly* = Returns information about folders only.

Working with DataLayer.Directory

The datalayer reads the file system directory information for the specified folder, filters it based on the **Directory Filter** attribute value (if any), and caches it as rows and columns, with one row per file or folder. Data retrieved into the datalayer is cached in XML format.

The data retrieved with a datalayer is available using @Data tokens, in the format @Data.*ColumnName*~. The spelling of column names are *case-sensitive*.

The data retrieved is only available within the scope of the parent element of the datalayer, not throughout the entire report definition. The DataLayer.Linked element can be used to make the data reusable in another datalayer outside this scope.

The datalayer will be populated with these column names, for use with @Data tokens:

Column	Description
Name	The simple name of the file or folder, with the file extension. Example: *myFile.xml*
PhysicalName	The fully-qualified path and name of the file or folder. Example: *C:\inetpub\wwwroot\myLogiApp_SupportFiles\myFile.xml*
Type	The item type, either *File* or *Folder*
Size	The item size, in bytes. Folders report a value of *0*.
DateModified	The time stamp indicating when the item was last modified. Example: *6/8/2016 2:25:12 PM*
DateCreated	The time stamp indicating when the item was created. Example: *6/8/2016 2:25:12 PM*
Extension	The file extension, if any, with leading period. Example: *.xml*

Column	Description
NameOnly	The file or folder name without file extension. Example: *myFile*
ParentFolder	The name of the folder thatcontains this file or folder.

File access permissions may need to be granted to the account that your Logi application runs under on the web server in order to read data with DataLayer.Directory, especially if the files or folders are not within your Logi application folder.

The data retrieved into the datalayer can be viewed by turning on the Debugger in the _Settings definition (General element) and using the resulting icon at the bottom of the report page to view the **Application Trace** page. A link on the Trace page will display the retrieved data.

DataLayer.Fixed File Format

The **Fixed File Format** datalayer enables retrieval of data from a text file in which the values are in a format specified by column widths.

The following sections are covered in this topic:

- Attributes
- Working with DataLayer.Fixed File Format

Attributes

The **DataLayer.Fixed File Format** element has the following attributes:

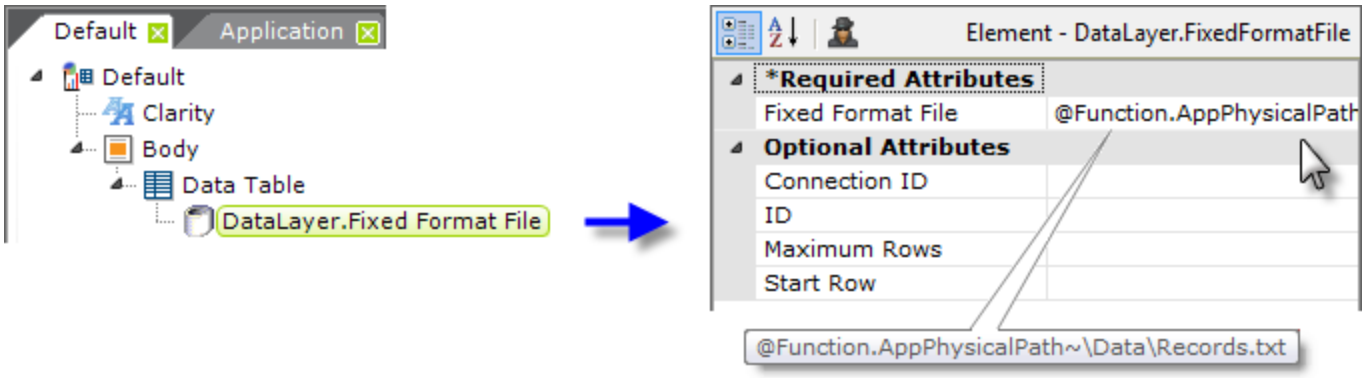
Attribute	Description
Fixed Format File	(Required) The location and name of the fixed format file. The path should not contain spaces, and can be a web server path, such as `C:\Data\records.txt` or a URL, such as `http://www.example.com/data/records.txt`
ID	An optional element ID. Recommended for easier identification when debugging.
Connection ID	Specifies a connection to a datasource that's defined in the _Settings definition.
Maximum Rows	The maximum number of rows to retrieve from the file.

Attribute	Description
Start Row	The row number at which to begin reading data. Default: *1*.

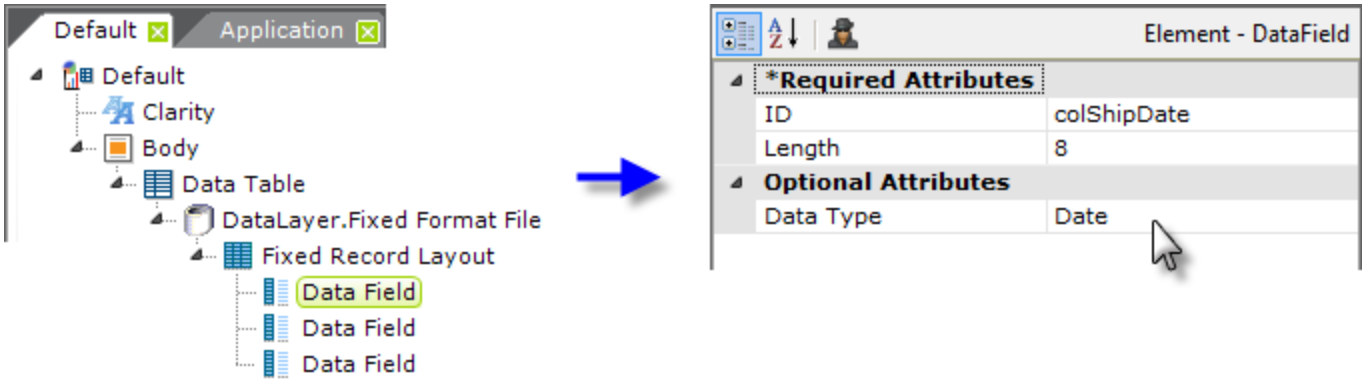
Working with DataLayer.Cached

The Fixed File Format datalayer enables retrieval of data from a text file in which the values occupy a certain number of characters and are not otherwise delimited.

The example above shows data of this type; the first "column" of data is eight characters wide, the second is two characters wide, and the last one is five characters wide. To use the datalayer, you must define the record layout using a **Fixed Record Layout** element and one or more **Data Field** elements.



In the example shown above, a **DataLayer.Fixed Format File** element has been added as a child element beneath a Data Table and its attributes set as shown. The **Fixed Format File** attribute will accept a value that's a literal web server file path, with an optional token (as shown above), or a fully-formed URL beginning with HTTP://.



Next, a Fixed Record Layout element has been added as a container for three Data Field elements. The Data Field elements are used to described each "column" of data in the text. Using the example data shown at the beginning of this section, the attributes

for the first Data Field element has been set to be 8 characters long and its data type has been set to Date. The column name for this data in the datalayer will be the ID of the Data Field element: *colDate*.

The remaining Data Field elements are then configured to describe all of the data columns. If their Data Type attribute is left blank, the data will be treated as text.

You can add other child elements beneath DataLayer.Fixed File Format and/or its child datalayers to modify the data, including:

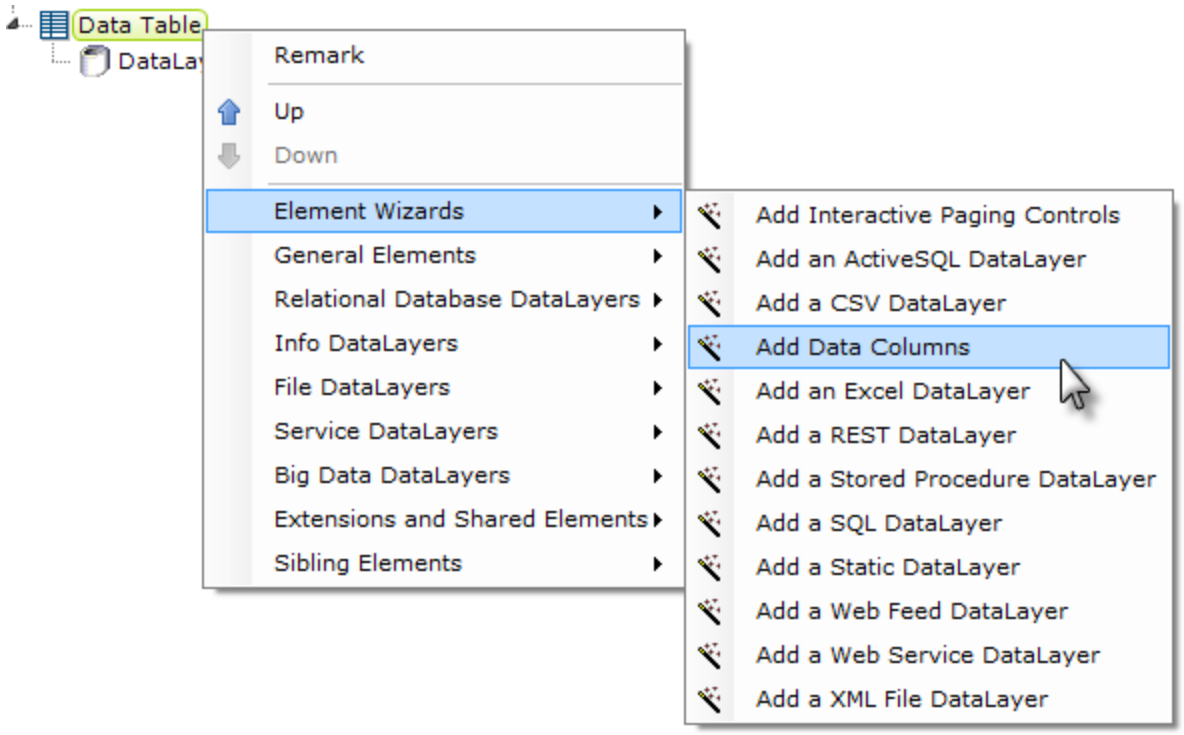
- **Filtering**: Sort, group, or restrict the data
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled data values
- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report definitions

Data retrieved into the datalayer is cached in **XML** format, in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

The data in the datalayer is available using **@Datatokens**, in the format @Data.*ColumnName*~. The spelling of the column name is **case-sensitive**. The data is only available within the **scope** of the **parent** element of the datalayer, not throughout the entire report definition. The **DataLayer.Linked** element can be used to make the data reusable in another datalayer outside this scope.

The *Auto Columns* element can be used to quickly display in your report all the data in a datalayer.

The data retrieved into the datalayer can be viewed by turning on the **Debugging Link** in your _settings definition (**General** element) and using the resulting link at the bottom of your report page to view the **Debugger Trace Report** page. There will be clear indications when the cache is being built and when data from it is being used. A link on the Trace page will display the actual cached data.



Studio includes a **wizard** that will add elements for the **columns** in the datalayer to your definition. You can select the desired columns before the operation begins. With the datalayer's parent **Data Table** or similar element selected in the Definition Editor, click the wizard link, shown above, in the Element Toolbox.

DataLayer.LDAP

The Lightweight Directory Access Protocol (LDAP) is an application protocol for reading and editing hierarchical sets of records over a network. Common implementations include user information and directory services.

The following topics discuss the DataLayer.LDAP element, which allows Logi applications to retrieve data from LDAP servers:

DataLayer.LDAP - Attributes

The DataLayer.LDAP element has the following attributes:

Attribute	Description
ID	An optional element ID. Recommended for easier identification when debugging.
Source	(Required) Specifies a query statement using a SQL-like syntax that returns a rowset of LDAP records, such as users, rights, groups, devices and other collections available from the LDAP server. See the LDAP Query Syntax sections below for more information.
Connection ID	The ID of a **Connection.LDAP** element defined in the _Settings definition. If this value is left blank, the datalayer will try to use the *first* connection element in the _Settings definition. For clarity, developers are advised to enter an ID here in all cases.

DataLayer.LDAP - Working with DataLayer.LDAP

The datalayer receives and **caches** the results returned by the LDAP query statement. You can add **child elements** beneath the datalayer to affect the results, including:

- **Filtering**: Sort, group, or restrict the result data
- **Joining**: Apply SQL-like JOINs to the data in the datalayer
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled result values
- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report definitions

Data retrieved into the datalayer is cached in **XML** format, in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

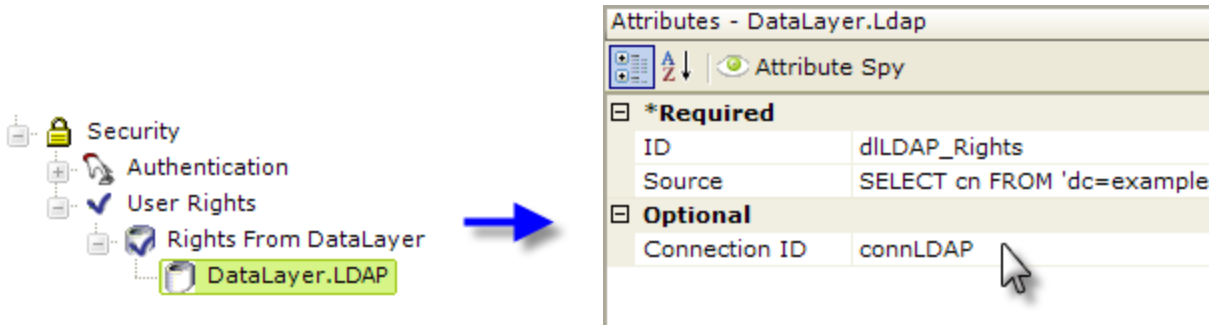
The data retrieved with a datalayer is available using @Data **tokens**, in the format @Data.*ColumnName*~. The spelling of the column name is **case-sensitive**. The data is only available within the **scope** of the **parent** element of the datalayer, not throughout the entire report definition. The **DataLayer.Linked** element can be used to make the data reusable in another datalayer outside this scope.

The *Auto Columns* element can be used to quickly display in your report all the data in a datalayer.

DataLayer.LDAP - Use with Logi Security

In order to authenticate users against an LDAP server, use the "DataLayer.LDAP - DataLayer.LDAP Authentication" on page 95 element.

When Logi Security is being used, DataLayer.LDAP can be used retrieve user rights from an LDAP server:



As shown above, DataLayer.LDAP is added as a child of the **Rights From DataLayer** element, and its attributes configured as shown. The syntax for the **Source** attribute value follows the rules described in the next sections, and includes the token containing the identifying value for the current user, for example:

```
SELECT cn FROM 'dc=example,dc=com' WHERE objectClass ='group' AND member = '@Function.UserName~'
```

DataLayer.LDAP - LDAP Query Syntax for .NET

The LDAP syntax supported in Logi products for .NET applications is similar to that of SQL. The basic keywords are:

Keyword	Description
SELECT	Specifies a comma-separated list of attributes to be retrieved for each object. If you specify *, the query retrieves only the Distinguished Name (DN) of each object, not all of its attributes. All other "column" operations of the type found in SQL queries (functions, AS, expressions, etc.) are *not* available.
FROM	Specifies the DN of the object of the search. For example, the DN of the "Users" container in an Active Directory domain might be 'cn=Users,dc=MassiveDynamic,dc=com'. The DN of the object of the search is enclosed in a pair of single quotation marks (').

WHERE	Specifies search filter expressions and multiple expressions may be strung together using AND and OR. Expressions only support these basic operators:
	`= Equal to`
	`~= Approximately equal to`
	`<= Lexicographically less than or equal to`
	`>= Lexicographically greater than or equal to`
	`& AND`
	`| OR`
	`! NOT`
	The SQL "LIKE" operator is *not* supported. The alternative is: [column] = '*SomeText*'.
	Expression comparison values are enclosed within a pair of single quotation marks (').

Query examples: List all users:

```
SELECT uid,postalAddress,mobile,mail,givenName,sn,cn FROM 'dc=example,dc=com' WHERE objectClass='Person'
```

List all groups:

```
SELECT description, cn FROM 'dc=example,dc=com' WHERE objectClass = 'groupOfUniqueNames'
```

List all groups with users:

```
SELECT uniqueMember, cn FROM 'dc=example,dc=com' WHERE objectClass = 'groupOfUniqueNames'
```

List all users for group "Developers":

```
SELECT uniqueMember FROM 'dc=example,dc=com' WHERE objectClass = 'groupOfUniqueNames' AND cn = 'Developers'
```

List all groups with user "user.0":

```
SELECT cn FROM 'dc=example,dc=com' WHERE objectClass ='groupOfUniqueNames' AND uniqueMember = '*uid=user.0*'
```

List all users in "Technical Group" Organizational Unit in Active Directory:

```
SELECT ADsPath,o,ou,objectclass,mail,name FROM 'ou=Technical Group,ou=Staff,dc=example,dc=com'
```

List all groups with user matching @Request token value:

```
SELECT cn FROM 'dc=LogiAnalytics,dc=com' WHERE objectClass = 'groupOfUniqueNames'
AND uniqueMember ='*uid=@Request.rdUsername~*'
```

As shown in the last example, you may use tokens, such as @Request and @Session, inside of the LDAP query to control the result set. Attribute values are stored with case intact in LDAP structures, but searches against them are case-insensitive by default. Certain attributes (like password) may be case-sensitive when searching.

DataLayer.LDAP - LDAP Query Syntax for Java

The LDAP syntax supported in Logi products for Java applications is similar to that of SQL. The basic keywords are:

Keyword	Description
SELECT	Specifies a comma-separated list of attributes to be retrieved for each object and can include "*" to retrieve all attributes. All other "column" operations of the type found in SQL queries (functions, AS, expressions, etc.) are *not* available.
FROM	Specifies the scope of the search. Values can include: *objectScope;* - a search of the base object only *oneLevelScope;* - a search of objects immediately subordinate to the base object, but not the base object *subTreeScope;* - a search of the base object and its entire subtree Scope values must end with a semi-colon. The base DN of the object of the search is configured in the **Connection.LDAP** element's **Base DN** attribute.

WHERE	Specifies search filter expressions and multiple expressions may be strung together using AND and OR. Expressions only support these basic operators:
	`= Equal to`
	`~= Approximately equal to`
	`<= Lexicographically less than or equal to`
	`>= Lexicographically greater than or equal to`
	`& AND`
	`| OR`
	`! NOT`
	The SQL "LIKE" operator is also supported, e.g. [column] = LIKE '%SomeText%'.
	Expression comparison values are enclosed within a pair of single quotation marks (').

Query example (also see .NET examples above, which only differ in regard to the FROM clause): `SELECT cn FROM subTreeScope; WHERE objectClass = 'groupOfUniqueNames' AND uniqueMember = '*uid=@Request.rdUsername~*'` As shown in the example, you may use tokens, such as @Request and @Session, inside of the LDAP query to control the result set. Attribute values are stored with case intact in LDAP structures, but searches against them are case-insensitive by default. Certain attributes (like password) may be case-sensitive when searching

DataLayer.LDAP - DataLayer.LDAP Authentication

The Lightweight Directory Access Protocol (LDAP) is an application protocol for reading and editing hierarchical sets of records over a network. Common implementations include user information and directory services. This topic introduces the developer to the **DataLayer.LDAP Authentication** element, which can be used to authenticate a user against an LDAP server.

The following sections are covered in this topic:

Attributes

The DataLayer.LDAP Authentication element has the following attributes:

Attribute	Description
ID	An optional element ID. Recommended for easier identification when debugging.
Connection ID	(Required) The ID of a **Connection.LDAP** element defined in the _Settings definition. If this value is left blank, the datalayer will try to use the *first* connection element in the _Settings definition. For clarity, developers are advised to enter an ID here in all cases.
Password	(Required) Specifies the password for an LDAP distinguished name (DN). Usually as a token with the value of the password passed from the standard or customized logon page: `@Request.rdPassword~`.
User DN Source	(Required) Specifies an LDAP query that returns the LDAP User Distinguished Name (DN). The returned DN

Attribute	Description

	will be used with the Password attribute to authenticate the current user. The query usually includes LDAP objects and names appropriate to your LDAP server and a token with the value of the user name passed from the standard or customized logon page, `@Request.rdUsername~`. Example LDAP queries: `(.NET and Standard LDAP Server)`
	`SELECT ADsPath FROM 'OU=Staff, DC=example, DC=com' WHERE uid='@Request.rdUserName~' AND objectClass='InetOrgPerson' (.NET and Active Directory Server)`
	`SELECT ADsPath FROM 'DC=myCompanyDomain, DC=local' WHERE SamAccountName= '@Request.rdUserName~' AND objectClass='organizationalPerson' (Java and Standard LDAP Server)`
	`SELECT DN FROM subTreeScope;`
	`WHERE uid='@Request.rdUserName~' AND objectClass= 'InetOrgPerson' (Java and Active Directory Server)`
	`SELECT CN FROM DC=myCompanyDomain, DC=local WHERE SamAccountName='@Request.rdUserName~' AND objectClass='organizationalPerson'`
	Prior to v10.0.503, this attribute was named "User DN".

Working with DataLayer.LDAP Authentication

This datalayer can only be used as a child of the Authentication element in the _Settings definition.

Unlike other datalayers, DataLayer.LDAP Authentication retrieves only a single record, if authentication succeeds, containing the **Common Name** attribute associated with the supplied user identifier. If authentication fails, nothing is returned. This datalayer does not have any child elements for filtering, joining, or extending its data.

The data retrieved into the datalayer can be viewed by turning on the **Debugging Link** in your _Settings definition (General element) and using the resulting link at the bottom of your report page to view the **Debugger Trace** page. A link on the Trace page will display the retrieved data.

DataLayer.JSON

The DataLayer.JSON element gives developers the ability to retrieve data from a JavaScript Object Notation (JSON) data file or REST-style web service.

Logi Info also includes "DataLayer.REST" on page 111 and "DataLayer.XML" on page 219 for use with REST-style web services and "DataLayer.Web Service" on page 204 for use with SOAP-style web services.

The following topics discuss the developer to the DataLayer.JSON element:

- DataLayer.JSON Attributes
- Working with DataLayer.JSON
- Using It with REST Web Services
- Multi-Step Processing
- Using Different HTTP Methods and Request Body Data

About DataLayer.JSON

This element was renamed; formerly it was known as DataLayer.JSON *File.* The element retrieves data from a static JSON file stored locally, or remotely (using a URL), or from a REST-style web service. DataLayer.JSON can also use **Connection.Logi Application Service** as its connection to data.


If you're trying to communicate with a web service that requires the TLS 1.1 or 1.2 protocol, you will need to use Logi Info v12.2-SP4 or later (earlier versions only support TLS 1.0). In addition, Info Java applications must use Oracle JDK 1.8 or OpenJDK 8 to make the protocol work.

DataLayer.JSON - Attributes

The DataLayer.JSON element has the following attributes:

Attribute	Description
Json File/URL	(Required) Specifies the identifier of the JSON data file. You may enter either a file system location or the URL for a location on the web. By default, the server will look in the _SupportFiles folder and, if the file is located there, then only the file name and extension are needed in this attribute. If it's located elsewhere in the file system, a fully-qualified path and file name is required, such as: `D:\Projects\Accounting\Data\MyData.js`If a URL is used to retrieve a file from a web server, it must be fully-formed, such as: `http://myserver/mydata.js`
Connection ID	Specifies the element ID of a Connection element defined in the _Settings definition. If a value is specified here, then Json File / URL attribute value is appended to the Connection element's Url Host attribute value (does not apply to Connection.Logi Application Service).
Http Method	Specifies the verb to be sent with a REST request. Options include: *DELETE*, *GET*, *POST*, and *PUT,* or can be a custom verb. The default value is *GET*.
ID	Specifies an optional element ID. Recommended for easier identification when debugging.
Maximum Rows	Specifies the maximum number of data rows to be retrieved.
XPath	Specifies a standard XPath string that will be used to select a set of matching nodes. All of the matching nodes are then used to generate the resulting datalayer. Tokens may be used in this attribute value.

DataLayer.JSON - Working with DataLayer.JSON

This element reads the data in a JSON file and converts it to XML in a datalayer:

```
{ "contact":
[
{
"firstName": "John",
"lastName" : "Smith",
"age" : 25,
"streetAddress": "21 2nd Street",
"city" : "New York",
"state" : "NY",
 "postalCode" : "10021",
"phonenumber": "212 555-1234"
},
{
"firstName": "Daniel",
"lastName" : "Rivas",
"age" : 34,
"streetAddress": "321 5th Avenue",
"city" : "New York",
"state" : "NY",
"postalCode" : "10024",
"phonenumber": "212 207-1346"
}
]
```

```
}
```

An example of JSON data in a data file is shown above...

```
<rdData>
 <dataTable_JsonFile>
 <contact>
 <firstName>John</firstName>
 <lastName>Smith</lastName>
 <age>25</age>
 <streetAddress>21 2nd Street</streetAddress>
 <city>New York</city>
 <state>NY</state>
 <postalCode>10021</postalCode>
 <phonenumber>212 555-1234</phonenumber>
 </contact>
 <contact>
 <firstName>Daniel</firstName>
 <lastName>Rivas</lastName>
 <age>34</age>
 <streetAddress>321 5th Avenue</streetAddress>
 <city>New York</city>
 <state>NY</state>
 <postalCode>10024</postalCode>
 <phonenumber>212 207-1346</phonenumber>
 </contact>
```

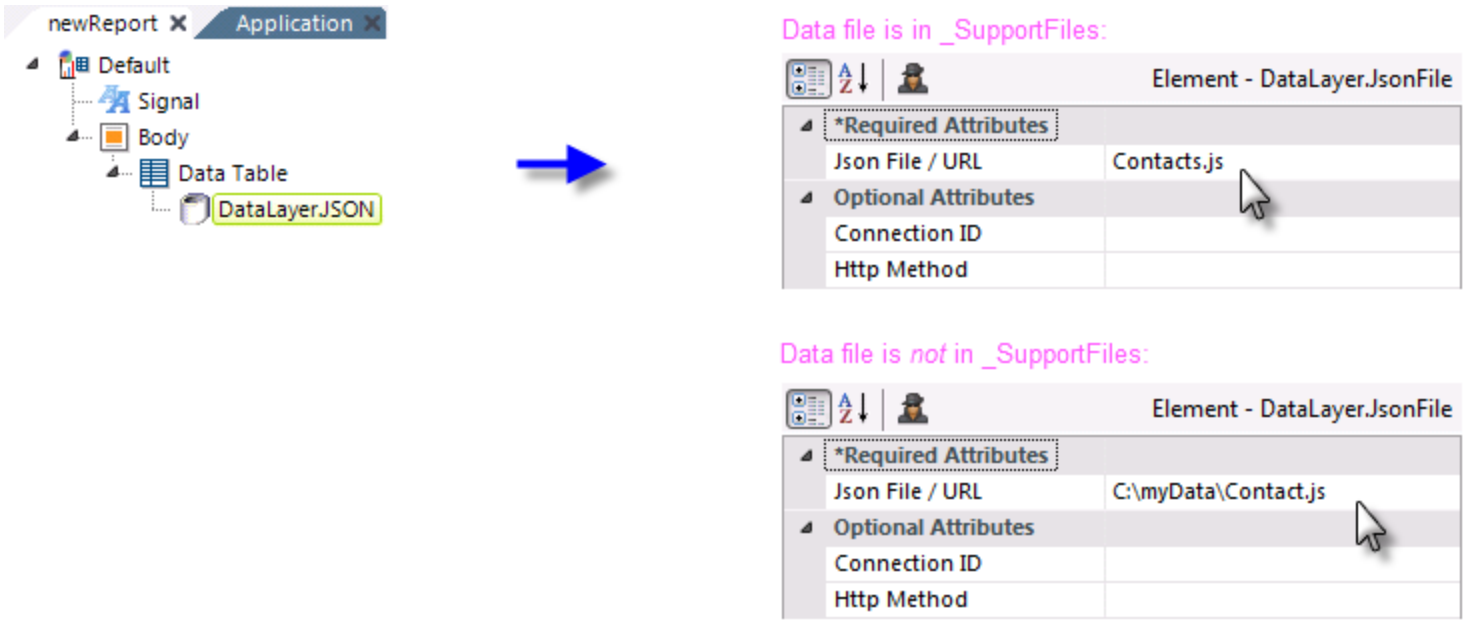
```
 </dataTable_JsonFile>
</rdData>
```

...and here's what it looks like as XML, above, after being read into the datalayer.

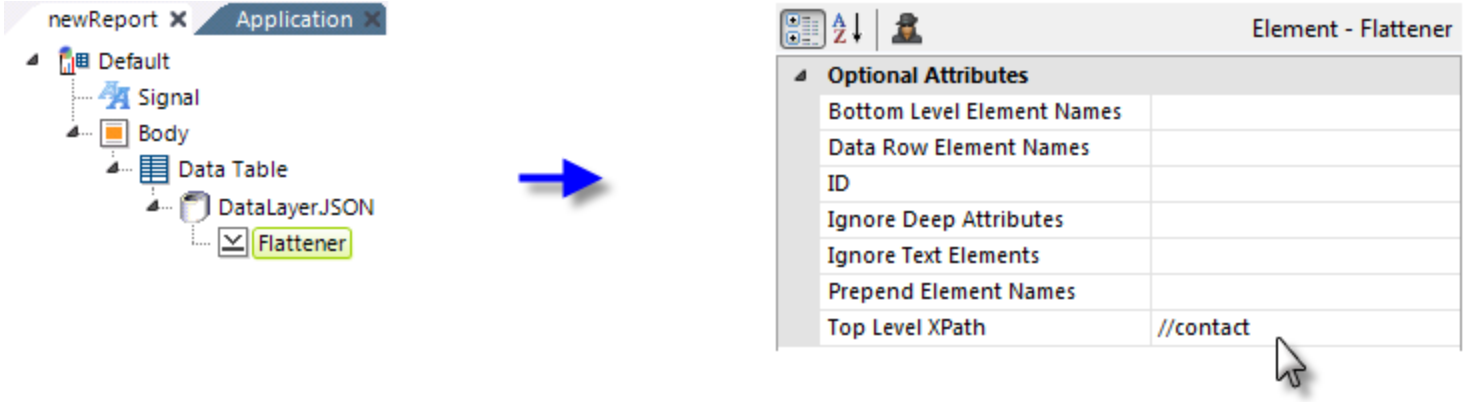
Data is usually in a hierarchical format, containing multiple levels of parent-child relationships. This structure can create difficulties for further processing and analysis. The **Flattener** child element processes hierarchical data into a tabular format, which is necessary when displaying data in most elements, like the Data Table.

```
<rdData>
 <dataTable_JsonFile firstName="John" lastName="Smith" age="25" streetAddress="21 2nd Street" city="New York" state="NY" postalCode="10021" phonenumber="212 555-1234"/>
 <dataTable_JsonFile firstName="Daniel" lastName="Rivas" age="34" streetAddress="321 5th Avenue" city="New York" state="NY" postalCode="10024" phonenumber="212 207-1346"/>
</rdData>
```

The code above shows the effects of applying the Flattener to the data shown earlier. It "flattens" the data into the row-column format that Logi elements expect. You can also apply one or more child **XslTransform** elements, which use XSL files to transform the XML data *before* it's loaded into the datalayer. This allows the adjustment of JSON data if it has a schema that cannot be understood by the Logi Server engine. For example, if the data comprises a data set with *two* distinct tables, a transform can remove a table, since the datalayer expects only one. Let's see an example of DataLayer.JSON in use:



As shown above, a **DataLayer.JSON** element has been added beneath a Data Table. Its **Json File/URL** attribute has been set to a value that can vary depending on the location of the actual file. There is no query or search feature; everything in the file will be pulled into the datalayer (where it can then be sorted, filtered, etc. using appropriate elements).

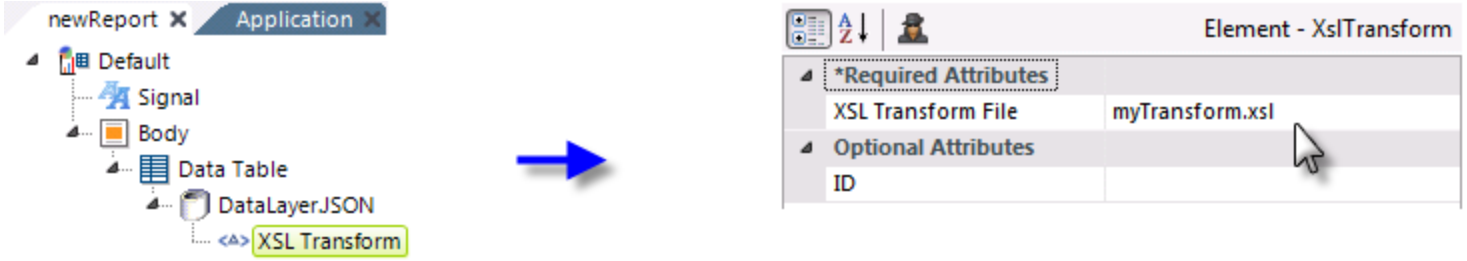


Next, a Flattener element is added and configured, as shown above, to flatten the example data we saw above into a tabular format.

rdFlattenedElement	firstName	lastName	age	streetAddress	city	state	postalCode	phonenumber
contact	John	Smith	25	21 2nd Street	New York	NY	10021	212 555-1234
contact	Daniel	Rivas	34	321 5th Avenue	New York	NY	10024	212 207-1346

The resulting Data Table is shown above.

The Flattener inserts a column in the datalayer, "rdFlattenedElement", which can assist you during development to understand the process. More information about the Flattener element is available in The Flattener.

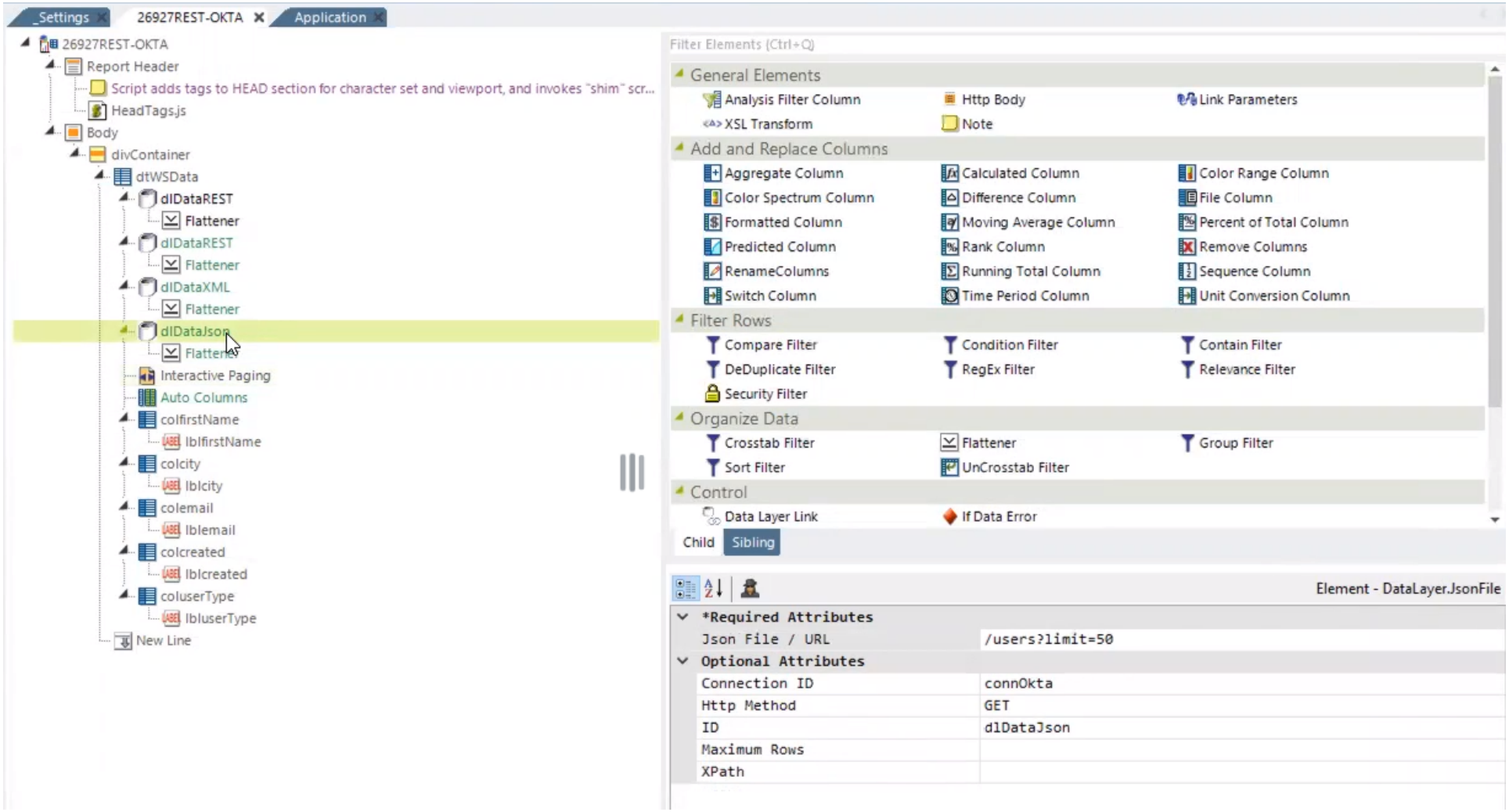


Though not related to the example data we've already seen, one or more **XslTransform** elements may be added beneath the datalayer to apply transforms to the data *before* it is retrieved into the datalayer, as shown above. The same rules apply for the XSL file location and name as for the data file (i.e. in the example above, the XSL file is in the _SupportFiles folder).

Pagination

Info now supports pagination from a Json data source, allowing you to handle large amounts of data results with ease. DataLayer.JSON automatically detects the link without any additional configuration, as long as that data source has that pagination mechanism and follows the standard RFC5988. We assemble all of the pages into a single data set and the engine consumes that data set to generate results.

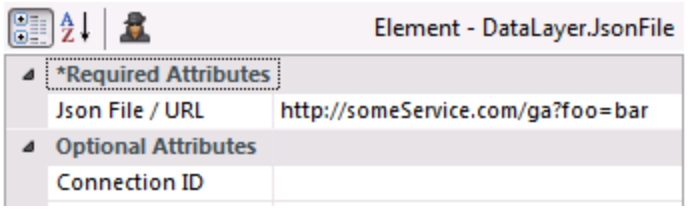
Specify the page size by setting the limit in the URL Path in the datalayer, or, defer to the default page size (200). Limit the number of data rows received by entering a number in the 'Maximum Rows' field.



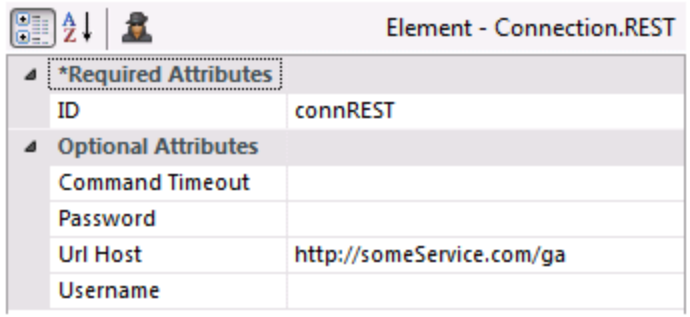
Your results only return the number of records defined by the limit and they will not include a link to additional pages.

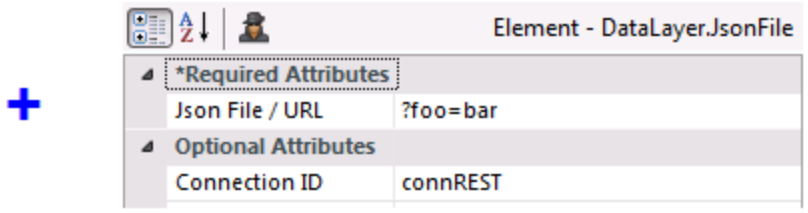
DataLayer.JSON - Using It with REST Web Services

DataLayer.JSON is able to retrieve data from REST-style web services and the simplest approach is to configure its **Json File/URL** attribute with a complete URL. If the web service requires nothing else, the data will be retrieved using the default HTTP GET method.



For more flexibility, you can instead use a **Connection.REST** element to define the first part of the URL:





Using a value in the datalayer's **Connection ID** attribute causes its Json File/URL value to be appended to the Connection element's **URL Host** attribute value, as shown above. This can be useful if your application uses multiple datalayers to connect to the same web service. If you prefer, you can leave the Json File/URL value blank and add a **Link Parameters** element beneath the datalayer. Its name/value pairs will then be appended to the Connection ID's URL Host value, preceded by a "?".

DataLayer.JSON - Multi-Step Processing

You may encounter a security scenario where a two-step process is required, such as when a web service expects a "login" API call with credentials (as opposed to supplying credentials in a Connection.JSON element) and returns an access ID, which must then be part of any other API calls. You can use this approach to handle this scenario:

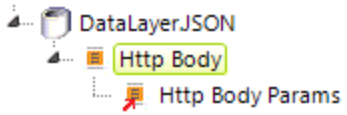
1. Use Local Data with DataLayer.JSON to authenticate and retrieve the web service's access ID. You may need to use a Flattener element to make the returned data available in tokens.
2. Use the resulting @Local token for the access ID with other DataLayer.JSON elements to retrieve the desired data.

Some web services may require you to login and get an access ID, using a login URL, and then use that access ID in the Request Header of any subsequent queries, which go to a different URL. In that case, you can use this approach:

1. Add *two* Connection.REST elements in your _Settings definition, one for the API login and one for API queries, each with their appropriate URL.
2. Add a Request Header child element beneath the "query" connection element.
3. Use Local Data with DataLayer.JSON and the "login" connection element to authenticate and retrieve the web service's access ID. You may need to use a Flattener element to make the returned data available in tokens.
4. Then use Set Session Variables to assign the resulting @Local token for the access ID to a session variable, for example "AccessID".
5. Use the @Session.AccessID~ token to configure the "query" connection element's child Request Header element.
6. All other DataLayer.JSON elements that make API calls will use the "query" connection element.

DataLayer.JSON - Using Different HTTP Methods and Request Body Data

If the web service requires you to use a different HTTP method, such as *POST* instead of *GET*, you can select it from the option list in the datalayer's **Http Method** attribute. Some web services may want you to use a custom method that's not in the option list, like *PATCH*, and in that case you can just type it right into the attribute value.



The **Http Body** element, shown above, has two attributes:

- **Content Type,** which sets the content type in the request header and defaults to *application/json* for this datalayer.
- **Http Body Content**, which is the request body data, entered as an XML or JSON document or as a URL-encoded set of name/value pairs.

If you want to encode name/value pairs in the request body data to avoid possible issues with invalid characters, set the Http Body element's Content Type attribute to *application/x-www-form-urlencoded* and use the **Http Body Params** element to define the pairs. Tokens may be used in Http Body Param values.

DataLayer.REST

The DataLayer.REST element gives developers the ability to retrieve XML or JSON documents from a web service that uses Representational State Transfer (REST) protocols.

The following topics discuss the DataLayer.REST element:

- DataLayer.REST Attributes
- Working with DataLayer.REST
- Multi-Step Processing
- Using Different HTTP Methods and Request Body Data
- Using Studio's DataLayer Wizard

About DataLayer.REST

The element retrieves data from a REST-style web service, as either XML or JSON documents and will work with SSL-secured sites.

 If you're trying to communicate with a web service that requires the **TLS 1.1** or 1.2 protocol, you will need to use Logi Info v12.2-SP4 or later (earlier versions only support TLS 1.0). In addition, Info Java applications must use Oracle JDK 1.8 or OpenJDK 8 to make the protocol work.

Logi Info also includes "DataLayer.XML" on page 219 and "DataLayer.JSON" on page 98for use with REST-style web services and "DataLayer.Web Service" on page 204 for use with web services that use the SOAP protocol.

To interact with REST-style web services from within a Process task, see *Process Tasks*.

DataLayer.REST - Attributes

The DataLayer.REST element has the following attributes:

Attribute	Description
Connection ID	(Required) Specifies the ID of a **Connection.REST** element defined in the _Settings definition.
Url Path	(Required) Specifies the portion of the URL that will be appended to the end of the URL specified in the Connection.REST element's Url Host attribute and may need to begin with a "/". For example, Connection.REST → Url Host = `http://local.yahooapis.com` DataLayer.REST → Url Path = `/MapsService/V1/geocode?appid=YD... etc.` producing this complete URL to request the web service method: `http://local.yahooapis.com/MapsService/V1/geocode?appid=YD... etc.`
Accept Type	Specifies the type of data expected to be received from a REST request. Options include *application/json* and *application/xml* (the default).
Http Method	Specifies the verb to be sent with a REST request. Options include: *DELETE*, *GET*, *POST*, and *PUT,* orcan bea custom verb.The default value is *GET*.
ID	Specifies an optional element ID. Recommended for easier identification when debugging.
Maximum Rows	Specifies the maximum number of data rows to be retrieved.
Remove	Specifies whether the namespace and schema information that some data sources will add to the retrieved

Attribute	Description
Namespace	data is removed. The default value is *False*.
XPath	Specifies a standard XPath string that will be used to select a set of matching nodes. All of the matching nodes are then used to generate the resulting datalayer. Tokens may be used in this attribute value.

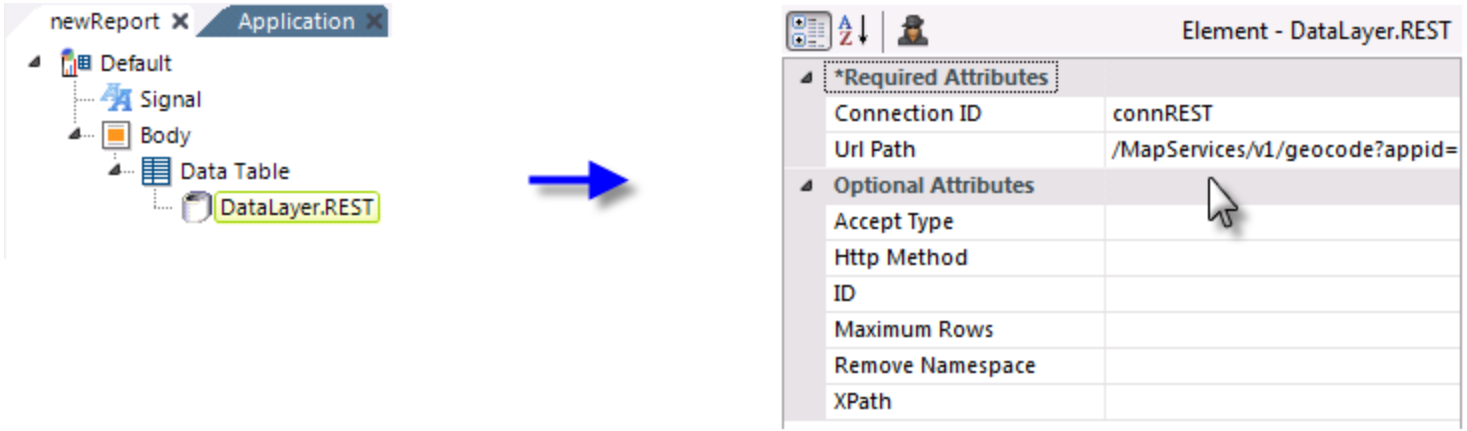
DataLayer.REST - Working with DataLayer.REST

A web service can expose one or more of its methods so that they can be called from an application, resulting in data being returned as a JSON or XML document in a serialized string.

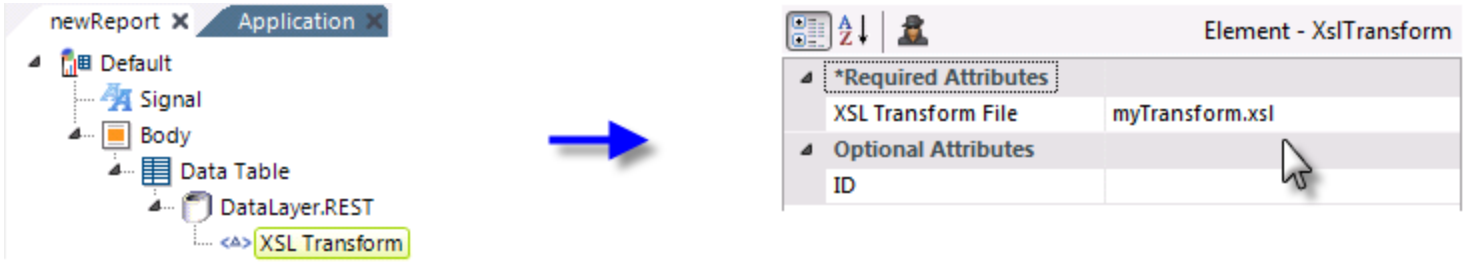
The web service's Web Application Description Language (WADL) document identifies the available methods. This is the REST equivalent of the SOAP-based web service's WSDL document it provides information about the service's methods, requirements, and data.

DataLayer.REST can be used with one or more child **XslTransform** elements, which use XSL files to transform the data *before* it's loaded into the datalayer. This allows the adjustment of data if it has a schema that cannot be understood by the Logi Server engine.

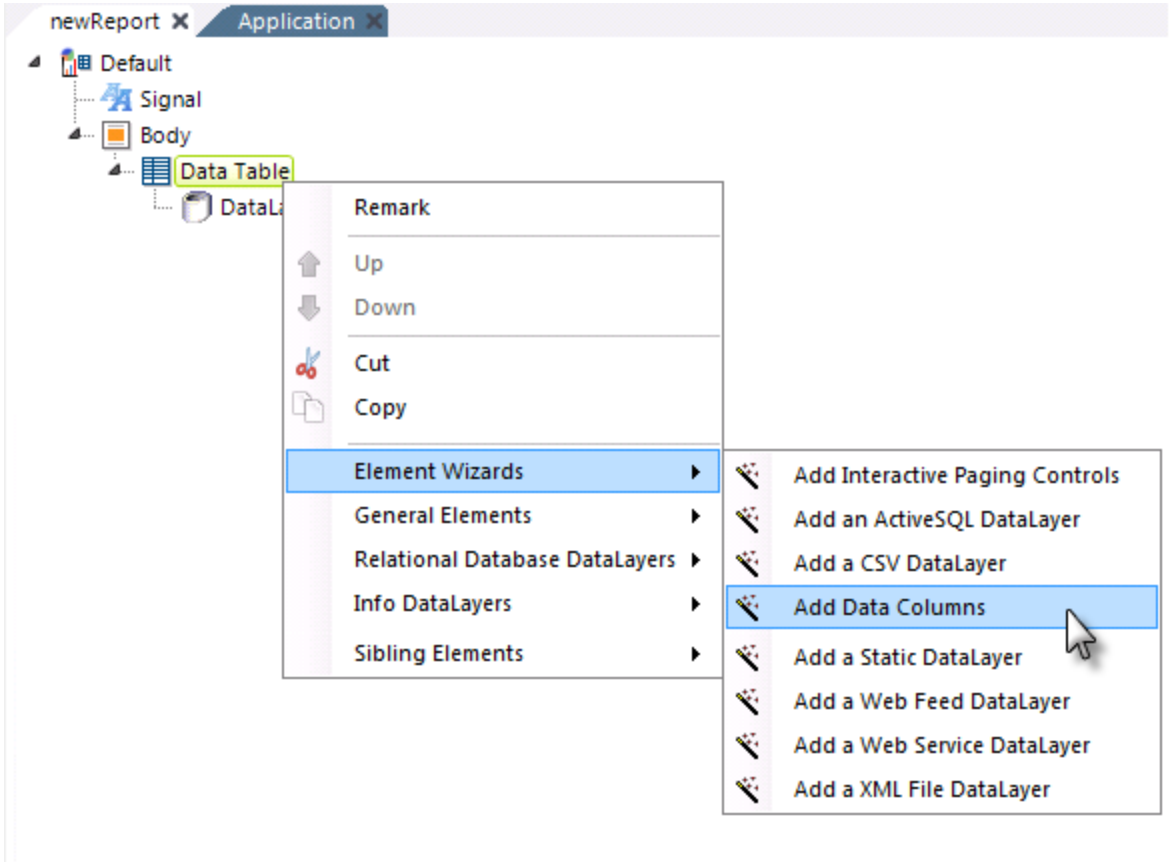
 Data may be returned in a hierarchical format, with parent-child relationships, such as JSON data. This data won't be fully available as standard datalayer XML and will create difficulties for further processing and analysis. *The Flattener* child element can be used to process this hierarchical data into the standard tabular set of rows and columns normally found in a datalayer. The following example shows DataLayer.REST in use:



As shown above, a **DataLayer.REST** element has been added beneath a Data Table. Its **Connection ID** and **Url Path** attributes have been set appropriately. Remember that its Url Path value will be appended to the Connection.REST element's Url Host value.



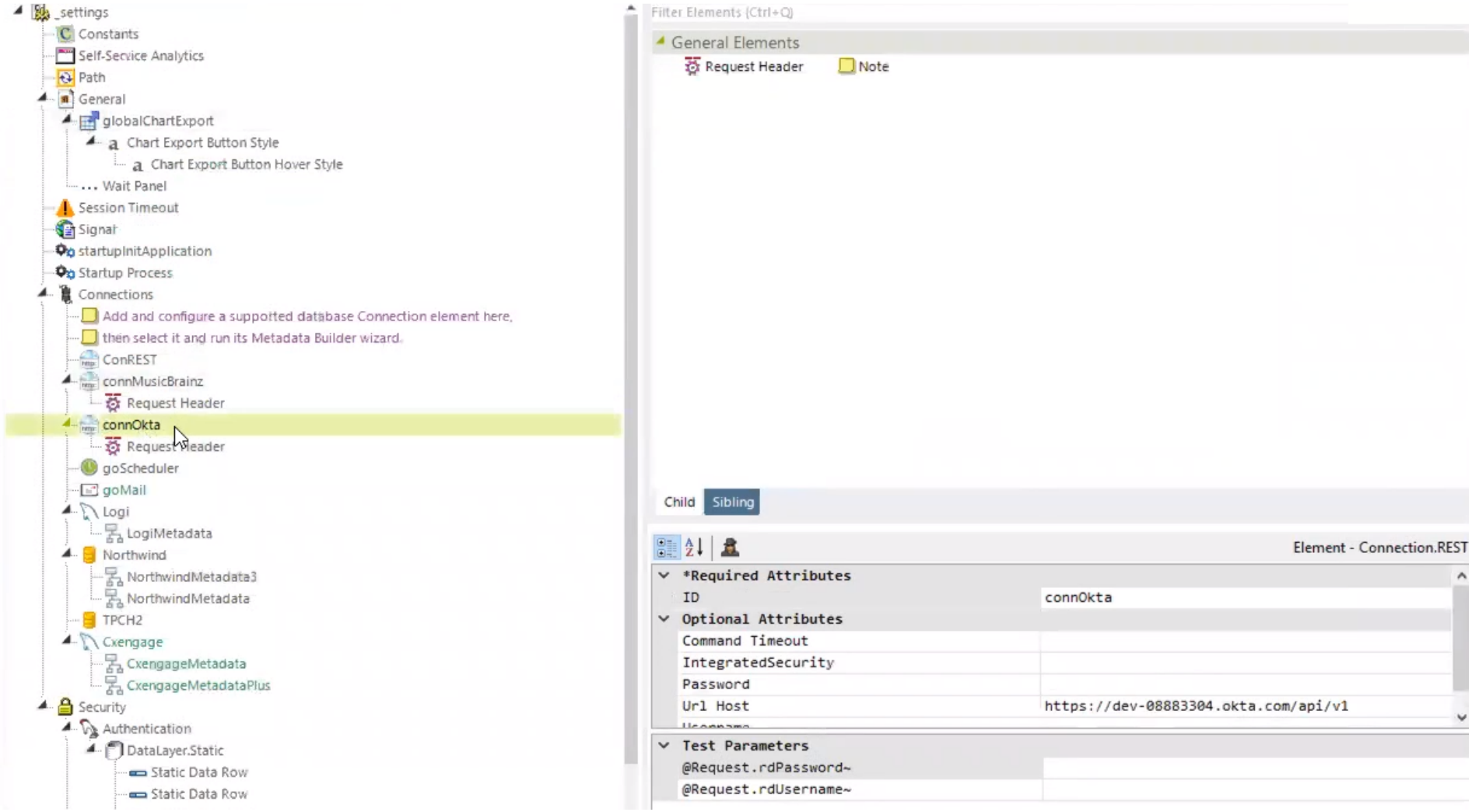
As shown above, one or more **XslTransform** elements can be added beneath the datalayer for each XSL transform to be applied to the data *before* it is retrieved into the datalayer. If a complete file path is not provided for the transform file, the engine will assume that the XSL file is in the _SupportFiles folder.



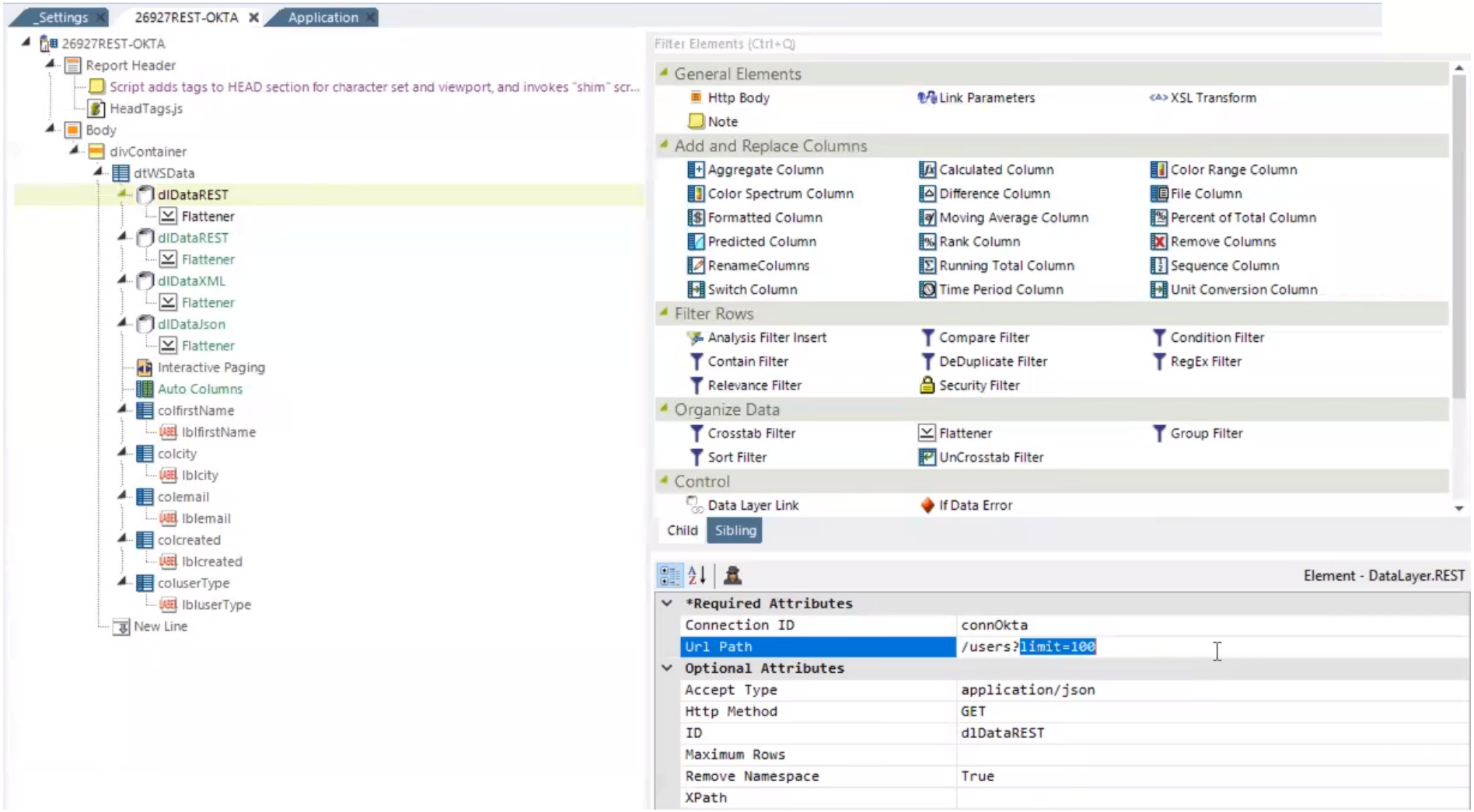
You may not have access to the web service's WADL file and, therefore, may not know the names of the data columns. In this case, Studio's **Add data columns** can be a tremendous help. Right-click the Data Table element, then select the options shown above in the pop-up menus. The wizard will interrogate the web service and insert the elements needed to display your Data Table columns.

Pagination

Info now supports pagination from a restful data source, allowing you to handle large amounts of data results with ease. DataLayer.REST automatically detects the link without any additional configuration, as long as that data source has that pagination mechanism and follows the standard RFC5988. We assemble all of the pages into a single data set and the engine consumes that data set to generate results.



Specify the page size by setting the limit in the URL Path in the datalayer, or, defer to the default page size (200). Limit the number of data rows received by entering a number in the 'Maximum Rows' field.



DataLayer.REST - Multi-Step Security Processing

You may encounter a security scenario where a two-step process is required, such as when a web service expects a "login" API call with credentials (as opposed to supplying credentials in a Connection.REST element) and returns an access ID, which must then be part of any other API calls. You can use this approach to handle this scenario:

1. Use Local Data with DataLayer.REST to authenticate and retrieve the web service's access ID. If the results are returned as JSON data, you'll need to use a Flattener element to make them available in tokens.
2. Use the resulting @Local token for the access ID with other DataLayer.REST elements to retrieve the desired data.

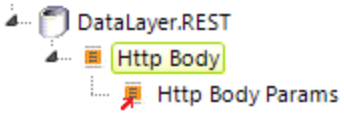
Some web services may require you to login and get an access ID, using a login URL, and then use that access ID in the Request Header of any subsequent queries, which go to to a different URL. In that case, you can use this approach:

1. Add *two* Connection.REST elements in your _Settings definition, one for the API login and one for API queries, each with their appropriate URL.
2. Add a Request Header child element beneath the "query" connection element.
3. Use Local Data with DataLayer.REST and the "login" connection element to authenticate and retrieve the web service's access ID. If the results are returned as JSON data, you'll need to use a Flattener element to make them available in tokens.
4. Then use Set Session Variables to assign the resulting @Local token for the access ID to a session variable, for example "AccessID".
5. Use the @Session.AccessID~ token to configure the "query" connection element's child Request Header element.
6. All other DataLayer.REST elements that make API calls will use the "query" connection element.

DataLayer.REST - Using Different HTTP Methods and Request Body Data

If the web service requires you to use a different HTTP method, such as *POST* instead of *GET*, you can select it from the option list in the datalayer's **Http Method** attribute. Some web services may want you to use a custom method that's not in the option list, like *PATCH*, and in that case you can just type it right into the attribute value.

In addition, some web services expect that request parameters be available in the "Request Body". DataLayer.REST has a child element that facilitates the passing of information in the request body for this purpose and the datalayer's Http Method must be set to *POST* in order to use it.



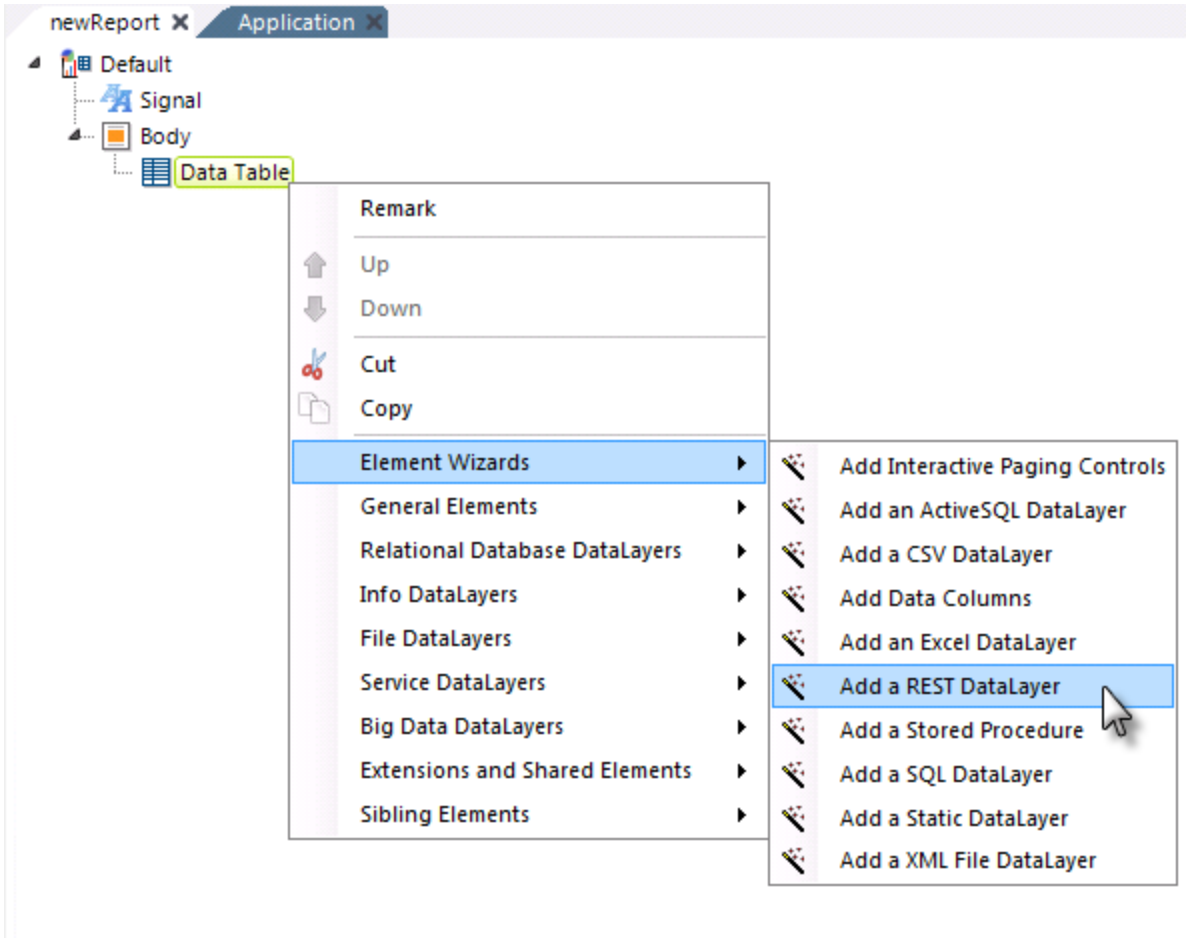
The **Http Body** element, shown above, has two attributes:

- **Content Type,** which sets the content type in the request header and defaults to *application/x-www-form-urlencoded* for this datalayer.
- **Http Body Content**, which is the request body data, entered as an XML or JSON document or as a URL-encoded set of name/value pairs.

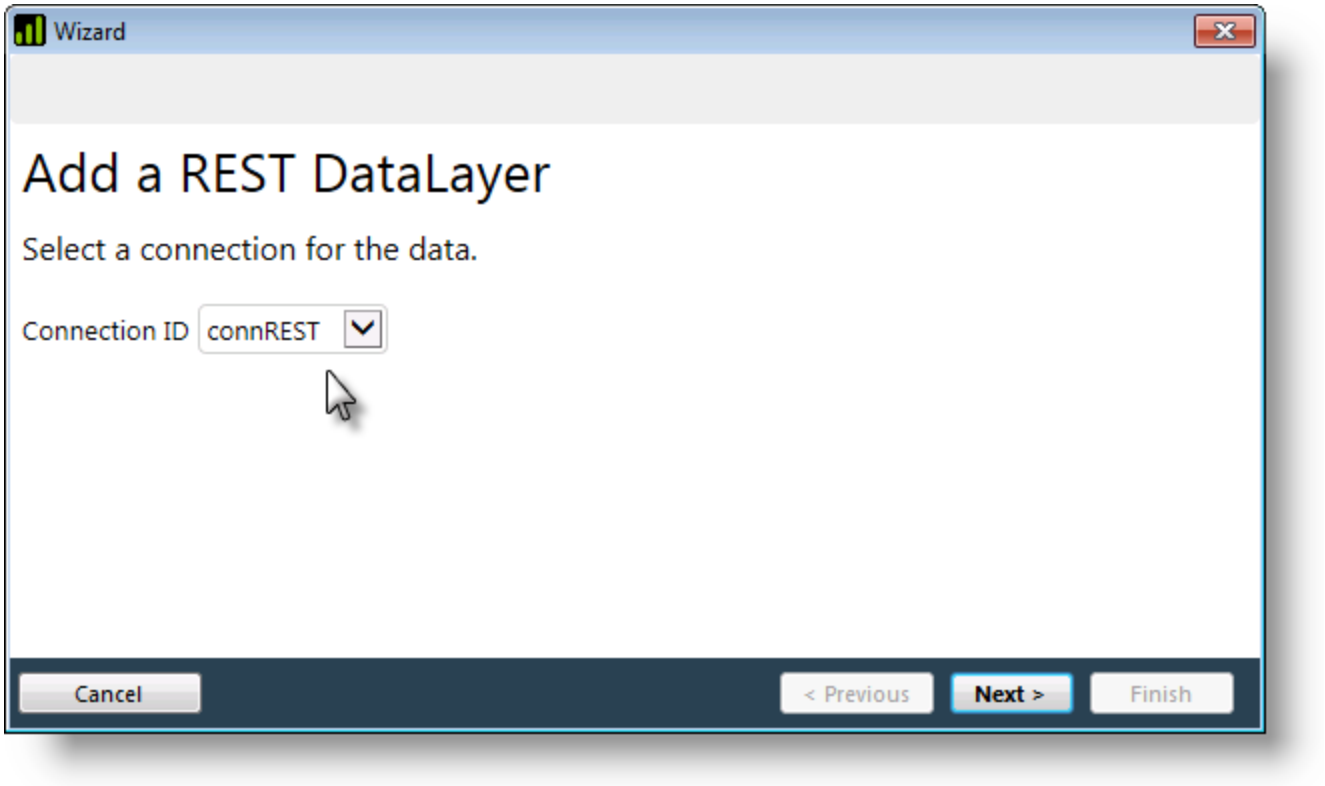
If you want to encode name/value pairs in the request body data to avoid possible issues with invalid characters, ensure that the Http Body element's Content Type attribute is blank or set to *application/x-www-form-urlencoded* and use the **Http Body Params** element to define the pairs. Tokens may be used in Http Body Param values.
For more information, our Web Service Sample Application (under the "Advanced" category) demonstrates the use of DataLayer.REST.

DataLayer.REST - Using Studio's DataLayer Wizard

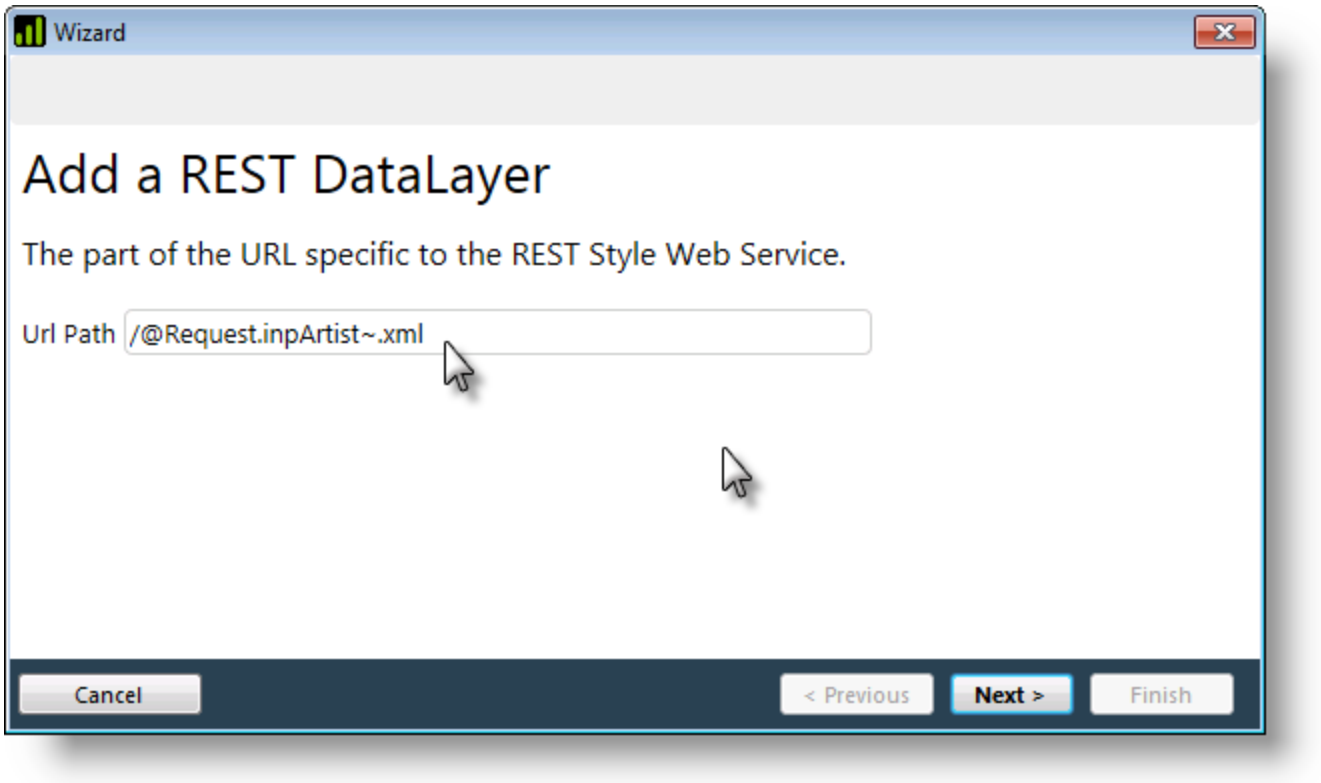
Logi Studio includes a wizard that can assist you in configuring DataLayer.REST. The wizard assumes that you have already added and configured a **Connection.REST** element in the _Settings definition.



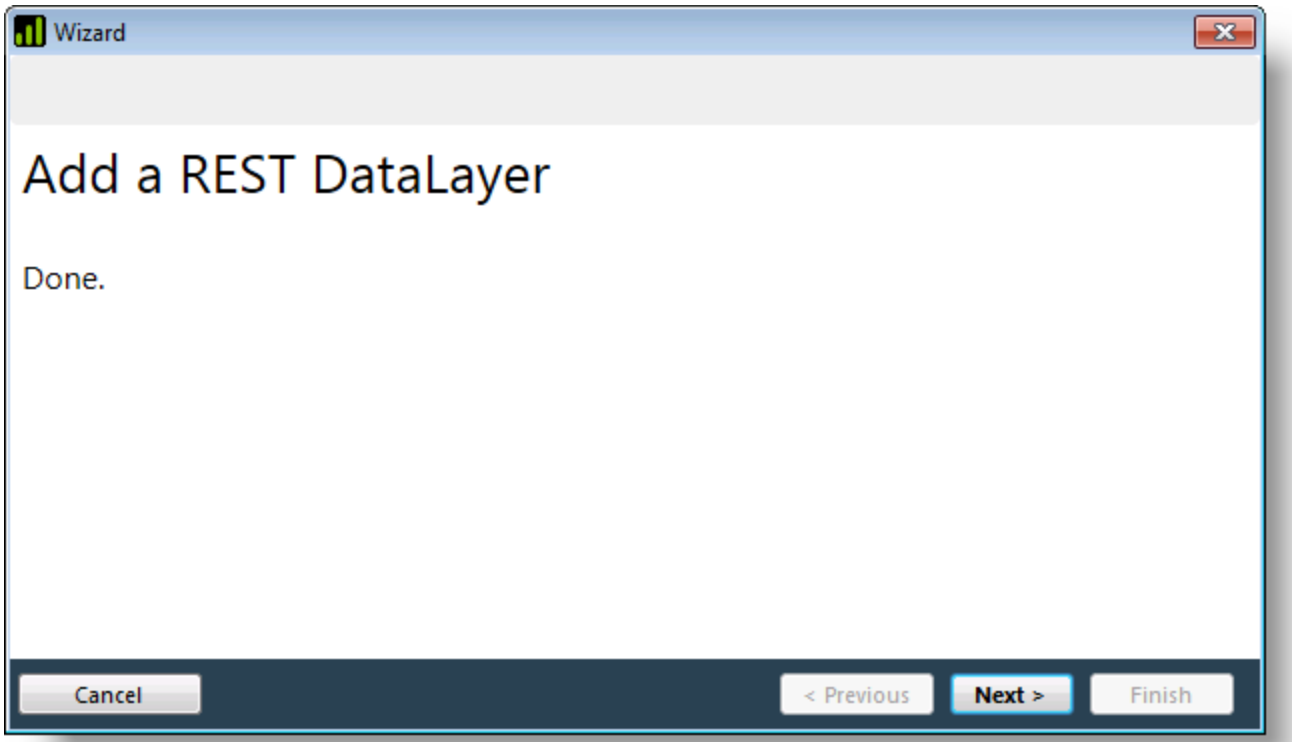
As shown above, the wizard can be started by selecting and then right-clicking the parent element under which you want to add the datalayer, and using the context menus to select "Add a REST DataLayer". The wizard will open; use it as follows:



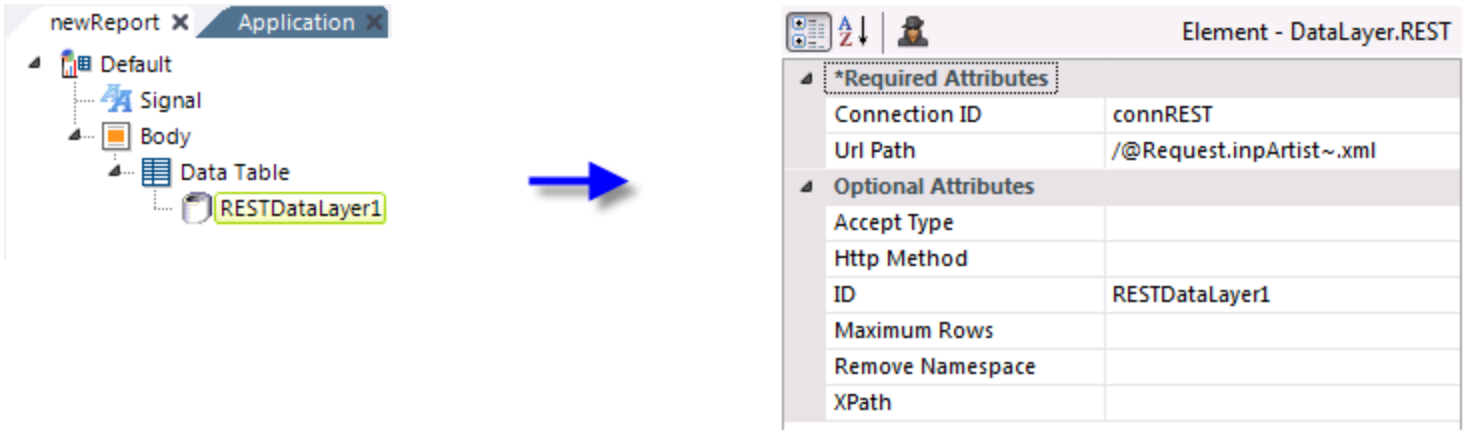
1. Select the ID of the Connection.REST element from the drop-down list of available connections. Click **Next** to continue.



2. Enter the Url Path for the web service. You can use tokens here.
3. Click **Next** to continue.



4. Click **Finish** to insert the datalayer.



5. The wizard will insert the datalayer and configure its attributes, as shown above. Other attributes, such as **XPath**, have to be configured manually.

DataLayer.SP

Developers working with SQL databases often prefer to use **Stored Procedures** rather than direct SQL statements in their applications for better performance and enhanced security.

The following topics introduce you to the DataLayer.SP element, which provides the capability of running stored procedures on SQL database servers:

- DataLayer.SP Attributes
- Retrieving Data with DataLayer.SP
- Stored Procedure Parameters
- Writing Data to the Database
- Using Studio's DataLayer Wizard

 Stored procedures executed using this element *must* return a rowset. Ensure that they end with a **SELECT** statement.

Using Oracle? Connection strings to Oracle databases must include the statement "PLSQLRSet=1" in order to use stored procedures.

DataLayer.SP - Attributes

The DataLayer.SP element has the following attributes:

Attribute	Description
Command	(Required) Specifies the **name** of the stored procedure (i.e. the name called to execute it).
Connection ID	Specifies the ID of a Connection element defined in the _Settings definition. If left *blank*, the datalayer will use the first connection in the _Settings definition. For clarity, developers are advised to enter an ID here in all cases.
ID	Specifies a unique element ID. Providing a value here is *highly recommended* for easier identification of data when debugging.
Maximum Rows	Specifies the maximum number of results rows to retrieve. Default: *no limit*.

DataLayer.SP - Retrieving Data with DataLayer.SP

The datalayer receives and caches the results returned by a SQL stored procedure. You can add child elements beneath the datalayer to affect the results, including:

- **Filtering**: Sort, group, or restrict the result data
- **Joining**: Apply SQL JOINs to the data in the datalayer
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled result values
- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report definitions

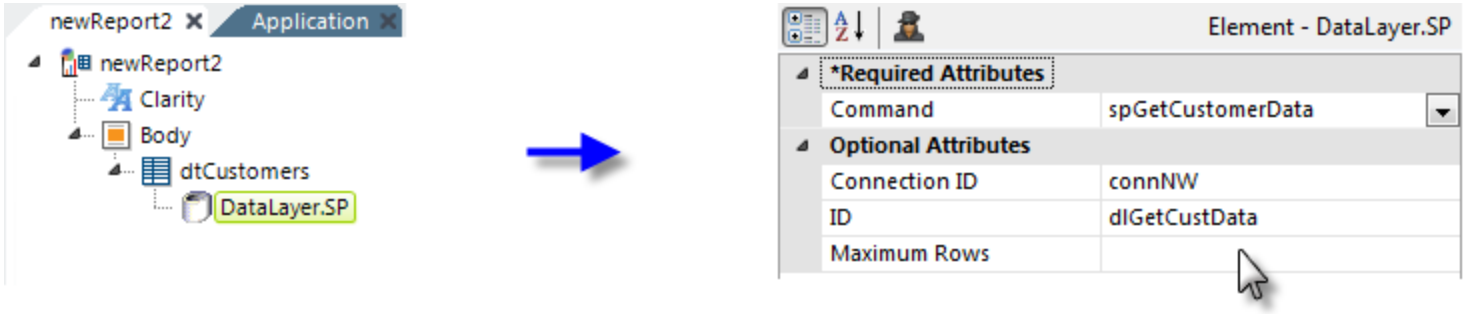
Data retrieved into the datalayer is cached in **XML** format, in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

The data retrieved with a datalayer is available using @Data **tokens**, in the format @Data.*ColumnName*~. The spelling of the column name is *case-sensitive*. The data is only available within the scope of the parent element of the datalayer, not throughout the entire report definition. The **DataLayer.Linked** element can be used to make the data reusable in another datalayer outside this scope.

The *Auto Columns* element can be used to quickly display in your report all the data in a datalayer.

The data retrieved into the datalayer can be viewed by turning on the Debugger Trace page in your _Settings definition (General element) and using the resulting link at the bottom of your report page to view the data.

In most respects, DataLayer.SP functions exactly as other datalayer elements do, however, stored procedures typically provide better performance than a standard SQL statement and have the added benefit of offering protection against SQL injection attacks.



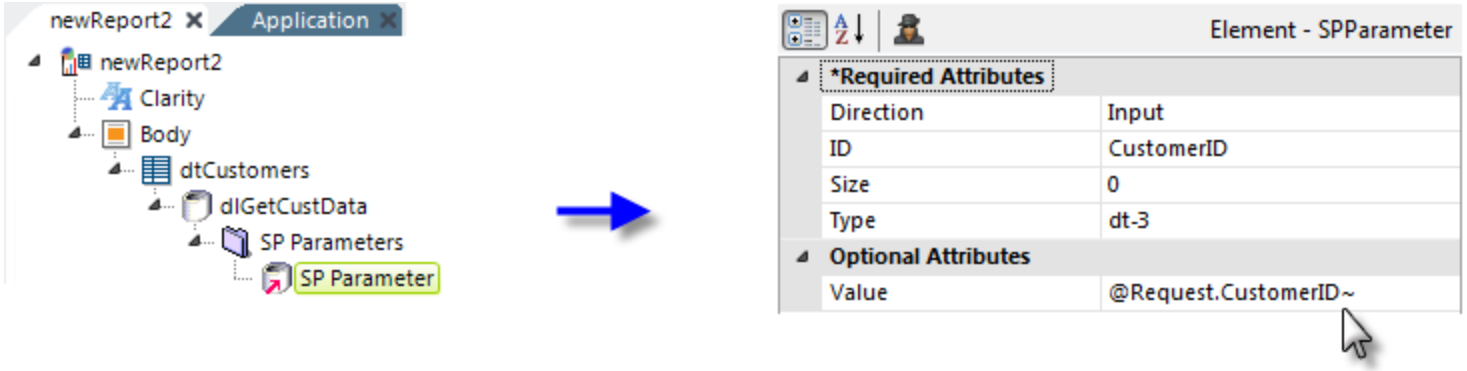
In the example shown above, a **DataLayer.SP** element has been added as a child element of a Data Table and its attributes set so that it calls the desired stored procedure on the database server.

The *only* text in the **Command** attribute value is the *name* of the stored procedure; you do not need to enter "EXECUTE" or other SQL commands here. For most major database servers, you can click the arrow at the end of this attribute to retrieve a list of available stored procedure names and then select the one you want. Input and output parameters are handled using special elements that are children of the datalayer and they're discussed in "DataLayer.SP - Stored Procedure Parameters" on the next page.

Result set data returned into the datalayer can be referenced using @Data tokens.

DataLayer.SP - Stored Procedure Parameters

You can pass values into, and receive them back from, stored procedures as parameters by using **SP Parameter** elements.



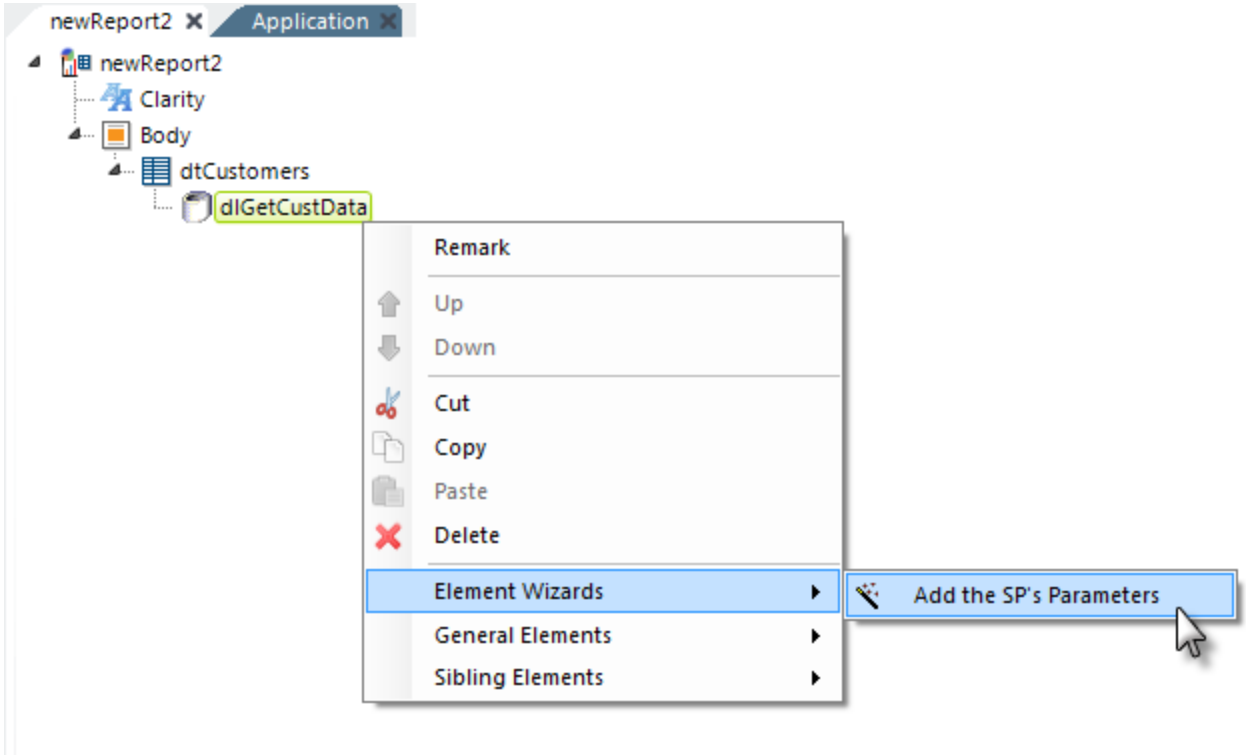
As shown above, an **SP Parameters** element has been added as the container for one or more **SP Parameter** elements.

The SP Parameter element has the following attributes:

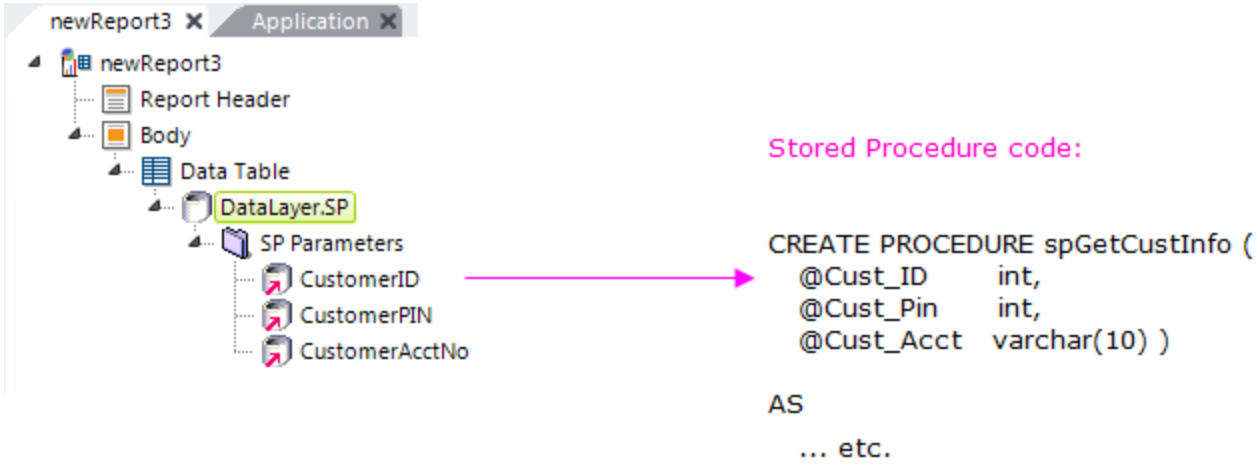
Attribute	Description
Direction	(Required) Specifies the *direction* or type of parameter. In Report and Process definitions, parameters can be used to send values to the SP; in Process definitions they can also be used to receive values returned from the SP. Options include: - *Input* type parameters are used to send values to the stored procedure but cannot be modified by it. - *Output* type parameters can be modified by the stored procedure and the returned values read. Output parameter elements should be placed *last* (at the bottom) of a collection of SP Parameter elements. (Process definitions only)

Attribute	Description
	- *Return* type parameters will provide the stored procedure's native Return value or the results of a RETURN statement in the procedure *IF you are using no other SP Parameters*. Otherwise it will return nothing. If you're using other SP Parameters, you can use this special token to get the SP return value: @Procedure.<*Procedure.SP.Element ID*>.RETURN_VALUE~ . (Process definitions only).
ID	Specifies an *arbitrary* ID for this parameter. Input parameters sent to the stored procedure are recognized sequentially, *not* by ID; the identifier assigned here is *not* associated with any parameter or local variable name in the Stored Procedure itself.
Size	Specifies the size or length of the parameter in bytes. If this is set to 0, the size will be automatically determined at runtime based on the length of the actual data used.
Type	Specifies the data type of the data used or expected. The data types used with stored procedures are discussed in detail in "Stored Procedure Data Types" on page 141.
Value	Specifies the actual value to be sent as an Input parameter; should be left *blank* for Output or Return parameters. Tokens can be used here to represent the value.

Multiple SP Parameter elements may be used if there are multiple parameters but input-type SP Parameter elements should be contiguous in the element tree and organized *sequentially* to match the order of the parameters declared in the stored procedure.



Studio includes a wizard that will automatically determine and add all of the parameters required for a named stored procedure. With the DataLayer.SP element selected in the Workspace editor, click the wizard link, shown above, in the Element Toolbox. *You must have a valid connection to the database defined at this time*. This example further illustrates the correlation between Input-type SP parameters and the stored procedures variables.



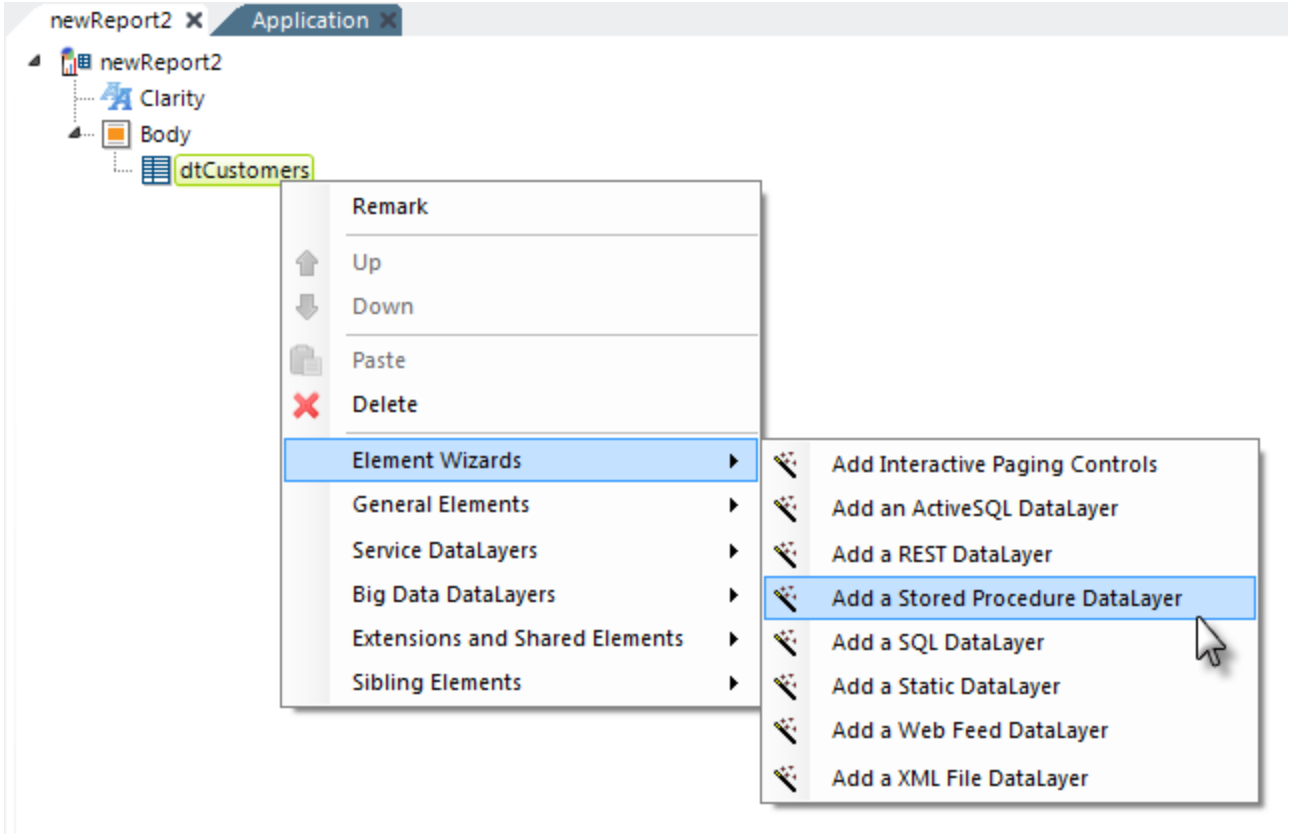
The SP Parameter element IDs in the procedure do not need to match the SP variables; they're assigned sequentially to variables in the stored procedure.

DataLayer.SP - Writing Data to the Database

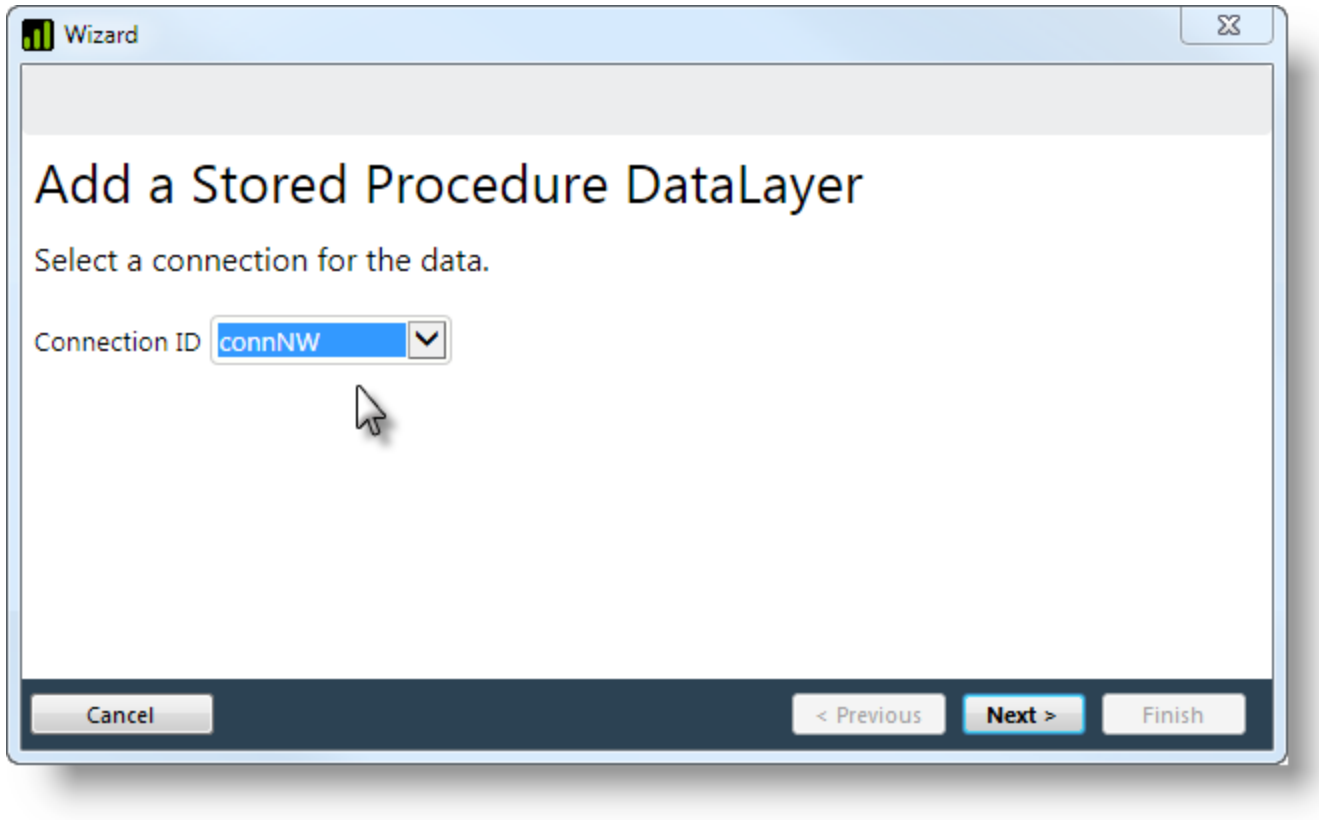
Logi Info applications are fully capable of writing data to a SQL database (the so-called "Writeback" capability) and can execute any valid SQL statement that does so, including INSERT, UPDATE, and DELETE. This is most commonly done in a Process Task using a **Procedure.SQL** element or by calling a stored procedure using a **Procedure.SP** element. Both of these techniques discussed in detail in *Process Tasks*.

DataLayer.SP - Using Studio's DataLayer Wizard

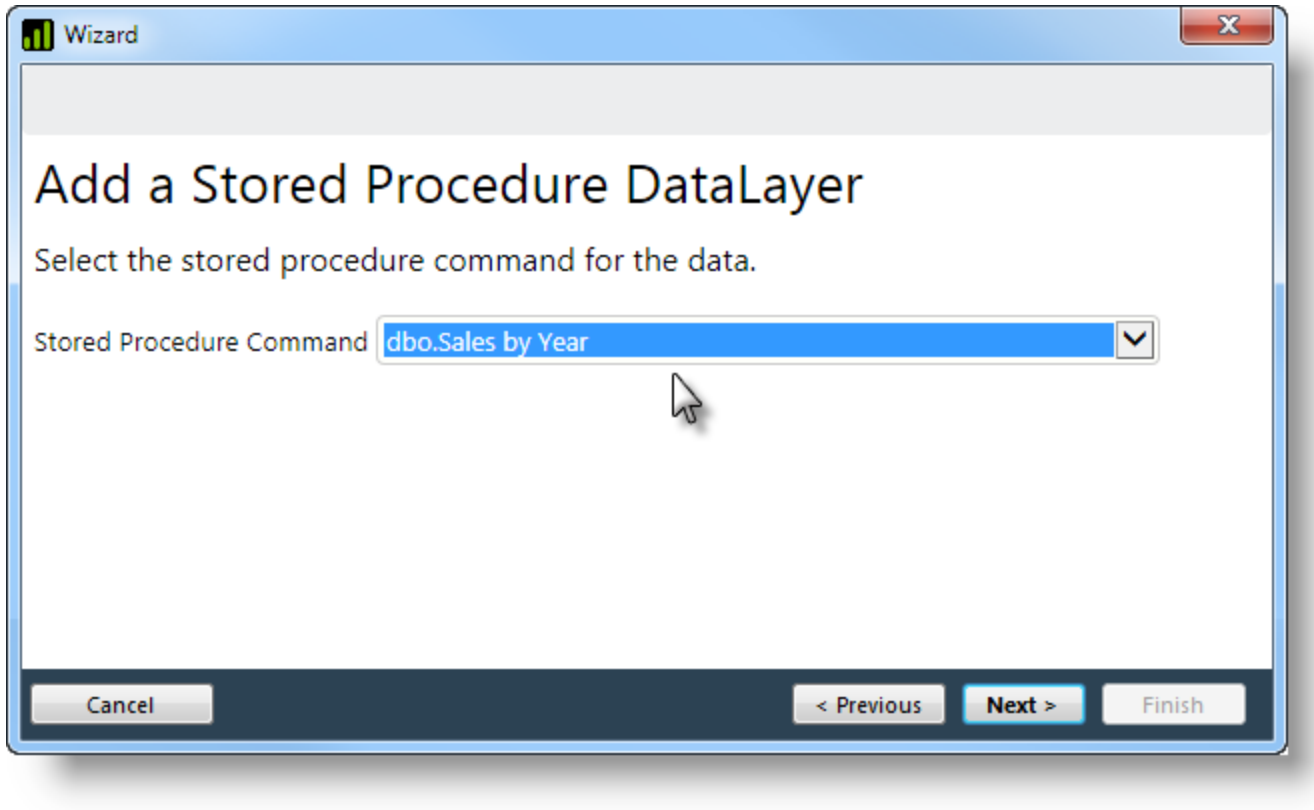
Logi Studio includes a wizard that can assist you in configuring DataLayer.SP. The wizard assumes that you have already added an appropriate database **Connection** element in the _Settings definition and configured it.



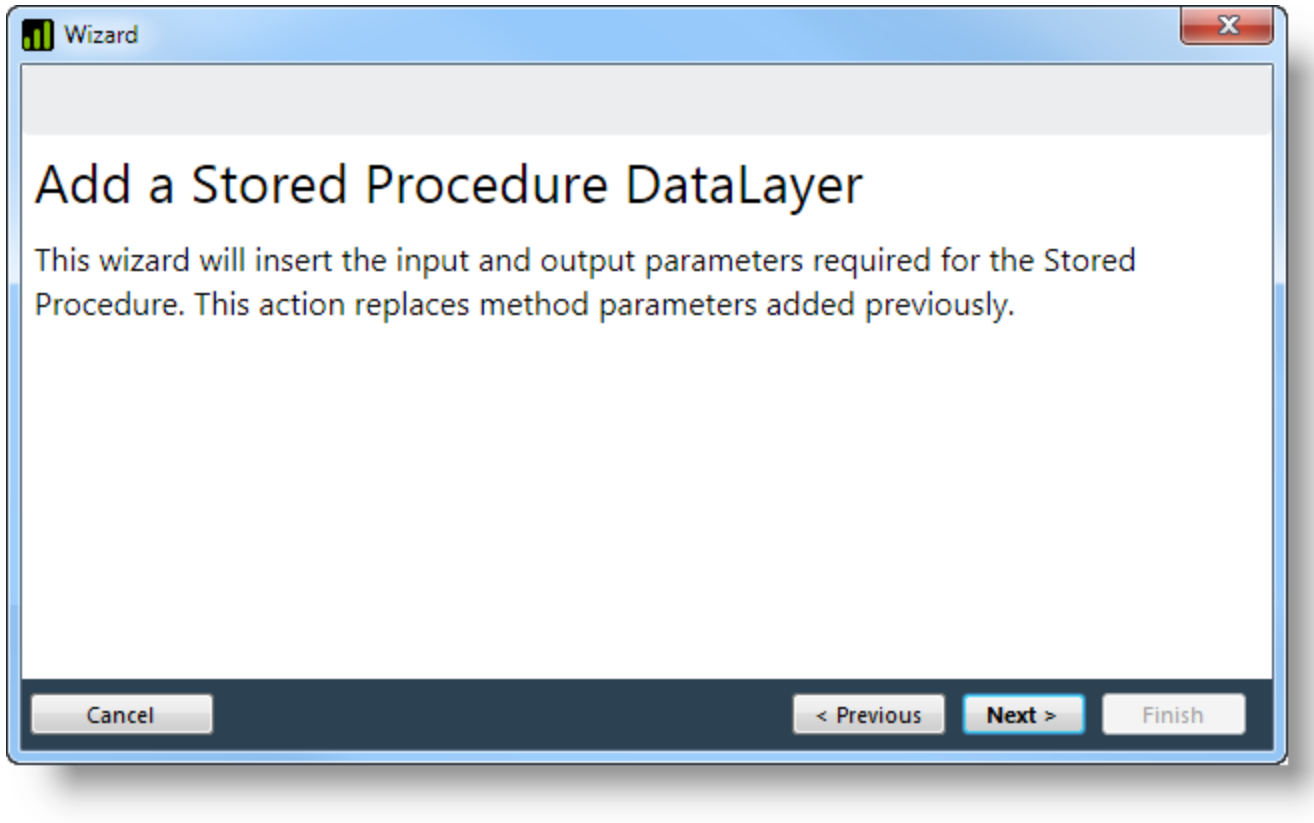
As shown above, the wizard can be started by selecting and then right-clicking the parent element under which you want to add the datalayer, and using the context menus to select "Add a Stored Procedure DataLayer". The wizard will open; use it as follows:



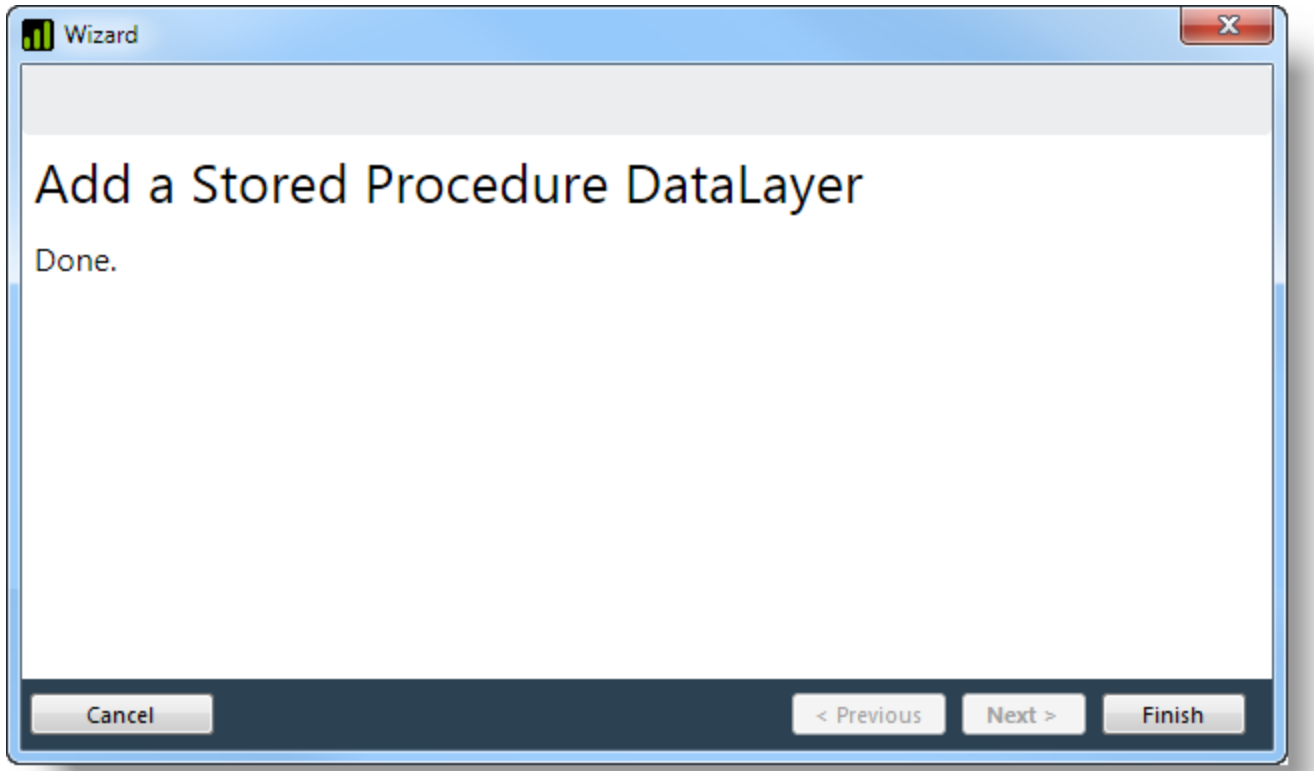
1. Select the ID of the Connection element from the drop-down list of available connections. Click **Next** to continue.



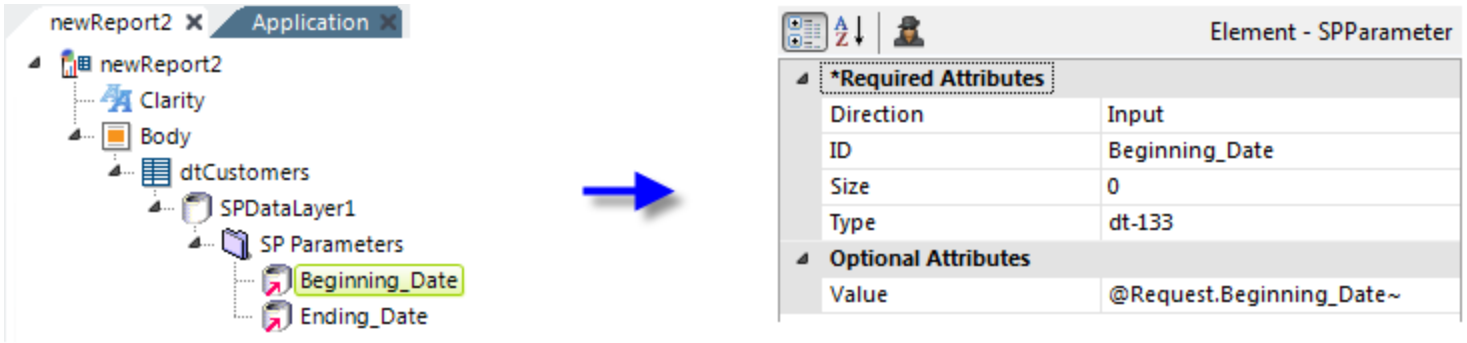
2. Select the desired stored procedure from the drop-down list of available stored procedures. Click **Next** to continue.



3. Click **Next** to have the wizard insert the DataLayer.SP and any SP Parameter elements it requires.



4. Click **Finish** to close the wizard.



5. The wizard has inserted the datalayer and configured its attributes, as shown above, and has also inserted and configured the required SP parameters. The input parameters have been configured assuming that @Request tokens will be used to provide their values.

Stored Procedure Data Types

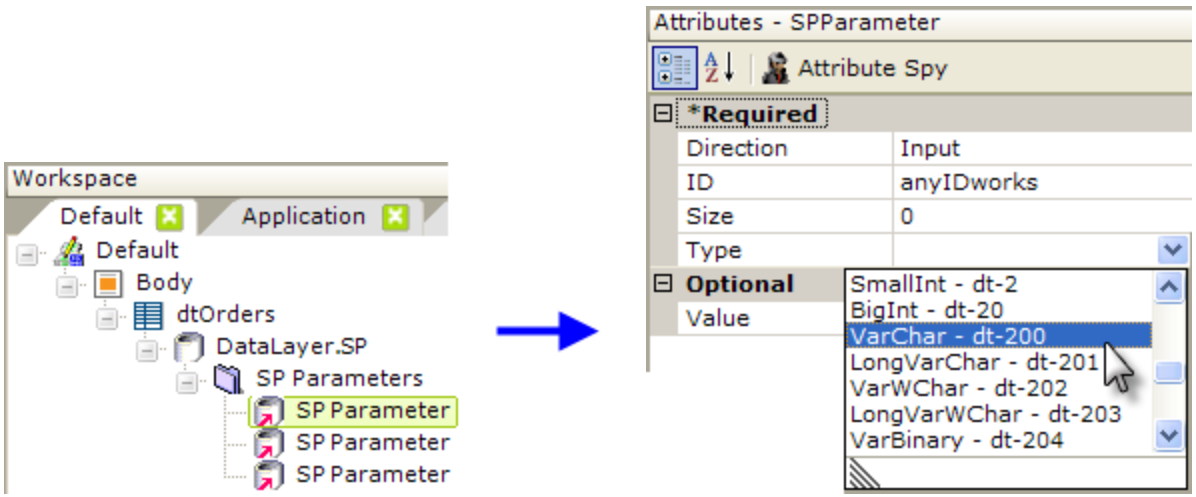
Logi applications have the ability to work with database server Stored Procedures (SPs). These are subroutines that are stored on the database server and offer benefits that include centralization, faster performance, and increased security. Typically Logi developers interact with stored procedures using the **DataLayer.SP** element and this topic describes the data types used with SPs.

Topics include:

- About Stored Procedures
- SP Parameter Data Types

About Stored Procedures

Like functions, SPs can be called with "input parameters", or arguments, that provide data to them, and they can return their results using output parameters. Logi products provide for this interaction with the **SP Parameters** container element, which is a child of DataLayer.SP, and its **SP Parameter** child elements.



As shown above, one SP Parameter element is used for each parameter required by the SP. Generally, the number of input parameters the SP expects must match the number of SP Parameter elements in the report definition, or an error will occur.

Logi products still include the **SP Auto Parameters** and **SP Input Parameters** elements for backward compatibility with older Logi applications, but they are deprecated and developers *should not* use them for new Logi apps.

SP Parameters can be configured to be **input** or **output** parameters, and they're identified by the SP by their *order*, not by their *IDs*. For example, if the SP lists input parameters as @name, @city, @state, then three SP Parameter elements are used and they must be placed in the definition in a top-to-bottom order with the "name" element first, "city" element second, and "state" element third. The IDs of the SP Parameter elements are irrelevant.

Additional information about using stored procedures can be found in "DataLayer.SP" on page 126.

Because data is being passed from the web server to the database server and possibly back, **data typing** is important. So each SP Parameter element is also configured to indicate its **data type**, which maps to the data type declared in the SP for it. The data types used by Logi applications follow the OLEDB data types and will work with most database servers, in either .NET or Java environments.

SP Parameter Data Types

The following data types are available when configuring SP Parameter elements:

Data Type	Description
BigInt	(dt-20) A 64-bit signed integer. The data type represents integers with values ranging from -*9,223,372,036,854,775,808* through *9,223,372,036,854,775,807*.

Data Type	Description
Boolean	(dt-11) A Boolean value, of either *true* or *false*
BSTR	(dt-8) A null-terminated character string of Unicode characters.
Char	(dt-129) A null-terminated character string of characters.
Currency	(dt-6) A currency value ranging from *-922,337,203,685,477.5808* through *922,337,203,685,477.5807* with an accuracy to a ten-thousandth of a currency unit.
Date	(dt-7) Date data, stored as a Double. The whole portion is the number of days since December 30, 1899, and the fractional portion is a fraction of a day.
DBDate	(dt-133) Date data in the format *yyyymmdd*.
DBTime	(dt-134) Time data in the format *hhmmss*.
DBTimeStamp	(dt-135) Date and time data in the format *yyyymmddhhmmss*.
Decimal	(dt-14) A fixed precision and scale numeric value between *-10^38 -1* and *10^38 -1*.
Double	(dt-5) A floating-point number within the range of *-1.79E +308* through *1.79E +308*.
Empty	(dt-0) No value.
FileTime	(dt-64) A 64-bit unsigned integer representing the number of 100-nanosecond intervals since January 1, 1601.

Data Type	Description
GUID	(dt-72) A globally unique identifier or "GUID", a 128-bit integer (16 bytes) that can be used across all computers and networks wherever a unique identifier is required. Such an identifier has a very low probability of being duplicated
Integer	(dt-3) A 32-bit signed integer, representing values that range from *-2,147,483,648* through *2,147,483,647.*
LongVarBinary	(dt-205) A long binary value, that represents an array of unsigned integers with values that range from *0*to *255.*
LongVarChar	(dt-201) A long null-terminated string value.
LongVarWChar	(dt-203) A long null-terminated Unicode string value.
Numeric	(dt-131) An exact numeric value with a fixed precision and scale, between *-10^38 -1* and *10^38 -1*.
Single	(dt-4) A floating-point number within the range of *-3.40E +38* through *3.40E +38.*
SmallInt	(dt-2) A 16-bit signed integer, representing values ranging from *-32768* through *32767*.
TinyInt	(dt-16) An 8-bit signed integer, representing values ranging from *-128* to *127*.
UnsignedBigInt	(dt-21) A 64-bit unsigned integer, representing values ranging from *0* to *18,446,744,073,709,551,615.*
UnsignedInt	(dt-19) A 32-bit unsigned integer, representing values ranging from *0* to *4,294,967,295*.

Data Type	Description
UnsignedSmallInt	(dt-18) A 16-bit unsigned integer, representing values ranging from *0* to *65535.*
UnsignedTinyInt	(dt-17) An 8-bit unsigned integer, representing values that range from *0* to *255.*
UserDefined	(dt-132) Allows the developer to extend the database server's scalar type system. UDTs can contain multiple elements and can have behaviors, differentiating them from the traditional alias data types which consist of a single system data type. Useful for creating date, time, currency, extended numeric types, and working with geospatial data and encoded or encrypted data.
VarBinary	(dt-204) A variable-length stream of binary data, e.g. an array of bytes.
VarChar	(dt-200) A null-terminated, variable-length stream of non-Unicode characters.
Variant	(dt-12) A special data type that can contain numeric, string, binary, or date data, and also the special values Empty and Null.
VarNumeric	(dt-139) A variable-length numeric value between *-10^38 -1* and *10^ 38 -1*.
VarWChar	(dt-202) A variable-length, null-terminated stream of Unicode characters.
WChar	(dt-130) A null-terminated stream of Unicode characters.

If in doubt about specific data types defined for your database server, consult its documentation for more information.

DataLayer.SQL

Logi Info developers working with SQL databases often run queries to retrieve data and issue other SQL commands.

The following topics introduce you to the DataLayer.SQL element:

- DataLayer.SQL Attributes
- Retrieving Data with DataLayer.SQL
- Using DataLayers to Retrieve SQL Data
- Using SQL Parameters
- Writing Data to the Database
- Using Studio's DataLayer Wizard

About DataLayer.SQL

The **DataLayer.SQL** element can be used as a child of Data Tables, charts, and many other elements.

The text of a SQL query can be entered directly into the DataLayer.SQL element's **Source** attribute, or generated using the special **SQL Query Builder** tool, which can be accessed by clicking the browse button at the end of the Source attribute. This tool allows developers to create a correctly-formatted SQL query using a graphical interface. You can also test your query in the tool, and view the resulting data.

All valid SQL queries can be executed using this element. The number of rows returned can be controlled using the Maximum Rows attribute.

Process Tasks can make use of a similar element, Procedure.SQL, to retrieve data and write it back to the database.

"DataLayer.SP" on page 126 allows you to use SQL Stored Procedures, which provides better protection against SQL injection attacks and better performance.

Developers interested in issuing queries to three-dimensional datasources, such as cubes, for use with two-dimensional visualizations, can make use of Studio's MDX Query Builder tool to formulate valid queries. For more information on the MDX Query Builder, see *Using Logi 12 Studio*.

DataLayer.SQL - Attributes

The DataLayer.SQL element has the following attributes:

Attribute	Description
Source	(Required) Specifies the SQL statement to be executed. Typically this is a SQL query, e.g. SELECT * FROM *MyTable*. However, multi-line SQL statements, including queries and statements like DECLARE, can be entered into the Attribute Zoom window for this attribute and will run correctly. Tokens may be used here so, for example, you could create a constant (in the _Settings definition) to hold your query text, then use a token like `@Constant.myQuery~` in this attribute. That would allow you to maintain the query in a single place, while using it in multiple datalayers. EXECUTE statements can be entered here as well, though use of stored procedures via the "DataLayer.SP" on page 126 element is recommended instead. When a connection to a non-OLAP datasource is being used, the Browse button for this attribute can be used to launch the SQL Query Builder wizard. If an OLAP connection is being used, the Browse button can be used to launch the MDX Query Builder. See the SQL Query Builder and MDX Query Builder sections of our *Using Logi 12 Studio* for more information on these topics.
ID	Specifies a unique element ID. Providing a value here is *highly recommended* for easier identification of data when debugging.
Connection ID	Specifies the ID of a database Connection element defined in the _Settings definition. If this value is left blank, the datalayer will use the *first* connection in the _Settings definition. For clarity, developers are advised to enter an ID here in all cases.
Handle Quotes Inside	Specifies how to handle any single-quotes found in token values used to form part of the SQL statement in the Source attribute. When *True*, it ensures SQL syntax validity by **doubling** any single-quotes found. For

Attribute	Description
Tokens	example, imagine the SQL statement "SELECT * FROM Customers WHERE CompanyName LIKE '%@Request.Name~%' ". If the value of the Request token is "Trail's Head", the embedded single-quote could be problematic. With the HandlesQuotesInTokens attribute set to *True*, the SQL statement sent to the database server will become "SELECT * FROM Customers WHERE CompanyName LIKE '%Trail''s Head%' ". Default: *False*
Maximum Rows	Specifies the maximum number of results rows to retrieve. Default: no limit.

DataLayer.SQL - Retrieving Data with DataLayer.SQL

The datalayer receives and caches the results returned by a SQL query or statement. You can add child elements beneath the datalayer to affect the results, including:

- **Filtering**: Sort, group, or restrict the result data
- **Joining**: Apply SQL JOINs to the data in the datalayer
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled result values
- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report definitions

Data retrieved into the datalayer is cached in **XML** format, in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

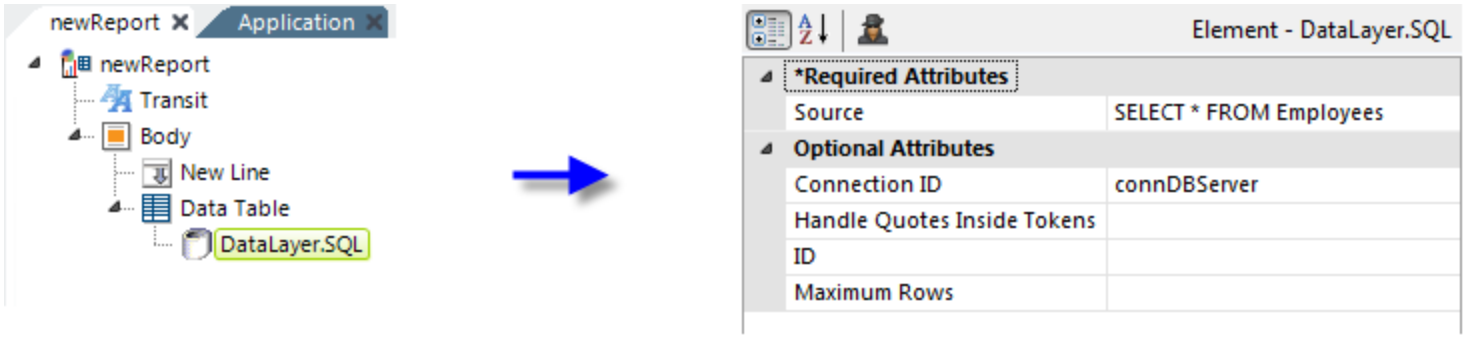
The data retrieved with a datalayer is available using @Data **tokens**, in the format @Data.*ColumnName*~. The spelling of the column name is *case-sensitive*. The data is only available within the scope of the parent element of the datalayer, not throughout the entire report definition. The **DataLayer.Linked** element can be used to make the data reusable in another datalayer outside this scope.

The *Auto Columns* element can be used to quickly display in your report all the data in a datalayer.

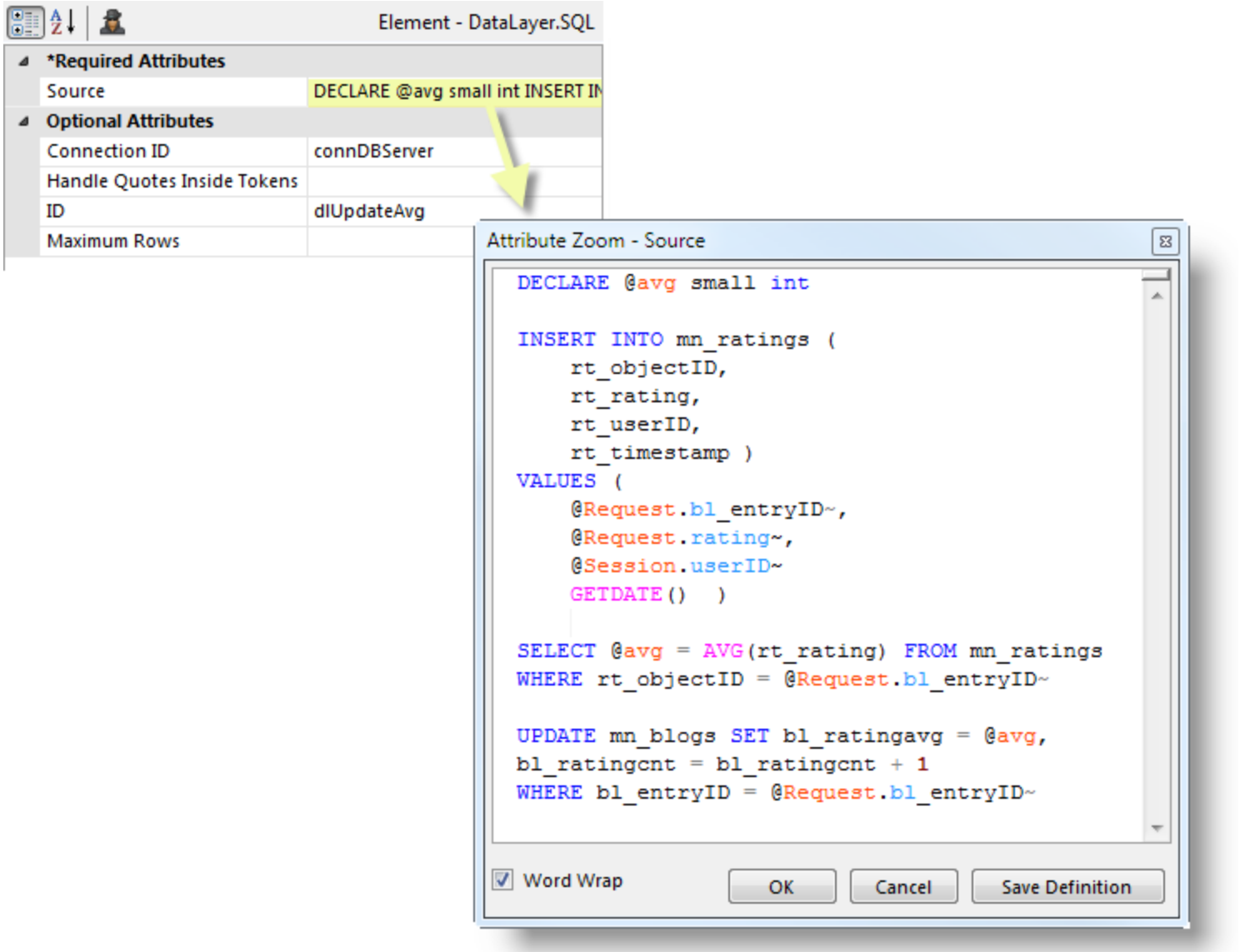
The data retrieved into the datalayer can be viewed by turning on the Debugger Trace page in your _Settings definition (General element) and using the resulting link at the bottom of your report page to view the data.

DataLayer.SQL - Using Datalayers to Retrieve SQL Data

DataLayer.SQL runs a SQL query to retrieve data from a SQL datasource. The element's **Source** attribute value is the actual SQL statement, and any valid SQL statement (or statements) will be run.



The example above shows a simple SQL query that returns data to a simple Data Table.


Element - DataLayer.SQL
*Required Attributes
Source
DECLARE @avg small int INSERT IN
Optional Attributes
Connection ID
connDBServer
Handle Quotes Inside Tokens
ID
dlUpdateAvg
Maximum Rows
Attribute Zoom - Source
DECLARE @avg small int
INSERT INTO mn_ratings (
rt_objectID,
rt_rating,
rt_userID,
rt_timestamp)
VALUES (
@Request.bl_entryID~,
@Request.rating~,
@Session.userID~
GETDATE())
SELECT @avg = AVG(rt_rating) FROM mn_ratings
WHERE rt_objectID = @Request.bl_entryID~
UPDATE mn_blogs SET bl_ratingavg = @avg,
bl_ratingcnt = bl_ratingcnt + 1
WHERE bl_entryID = @Request.bl_entryID~
Word Wrap
OK
Cancel
Save Definition

However, as shown in the example above, the Source attribute value (opened in the Attribute Zoom Window) can consist of *multiple* SQL statements. They will all be executed, as long as the syntax is correct, and any results will be returned to the datalayer. The Attribute Zoom window is opened for any attribute by double-clicking the attribute name.

All valid SQL statements can be executed. The **Handle Quotes Inside Tokens** attribute handles tokens that might have embedded single quotes, such as the text *Trail's Head*. When set to *True*, token values in the SQL Command will be wrapped in single-quotes, "doubling" them so the syntax will be valid. The default is *False*.

Tokens for string values should be wrapped in quotes to conform to your database SQL syntax. For example, if you'd use a query with a WHERE clause including quotes, like this:

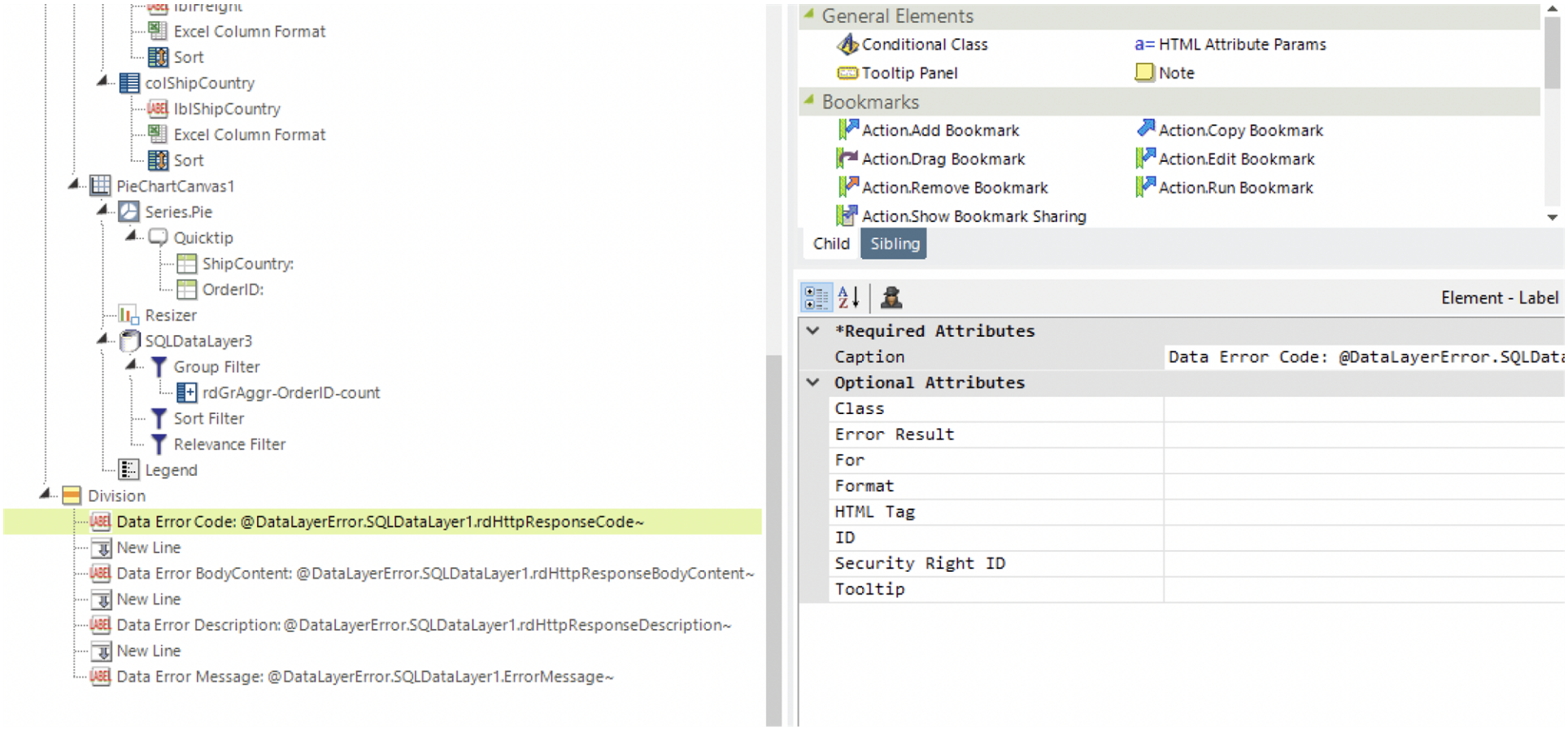
```
...WHERE user_lastname = 'Smith'
```

Then with tokens it would be:

```
...WHERE user_lastname = '@Request.UserName~'
```

The **If Data Error** element can be used beneath DataLayer.SQL to handle errors that may occur, by switching to an alternate datalayer. While the multi-statement capability makes it easy to use them, your best performance for complex SQL queries and best security will be found by using stored procedures and the **DataLayer.SP** element.

Or, use the token element `@DataLayerError` in conjunction with the **If Data Error** element to capture all of the error information.



Depending on the information you want to display on the page, you can choose from the following tokens:

- `ErrorMessage~`
- Restful data source only: `rdHttpResponseBodyContent~`

- Restful data source only: `rdHttpResponseBodyCode~`
- Restful data source only: `rdHttpResponseDescription~`

Use divisions to customize the error message to include images, buttons, and links.

See "Datalayer Error Handling" on page 36 for more information.

DataLayer.SQL - Using SQL Parameters

The **SQL Parameters** and **SQL Parameter** elements can be used to include tokenized parameters, if you prefer not to embed tokens directly into the SQL statement. This approach also allows you to enforce a data type for the parameter value and also offers protection against SQL Injection attacks.

To use parameters, you write your SQL statement using "placeholder" notation. The exact syntax depends on your database *and* the Connection element you're using. For example, if you're using **Connection.SQLServer**, you use this notation:

```
SELECT * FROM Customers WHERE State=@state AND City=@city
```

and the **ID** attribute of each SQL Parameter element must match a placeholder *name*. So, in the example, the value of an element with an ID of *state* will replace the @state placeholder at runtime.

Similarly, if you're using **Connection.Oracle**, you use this notation:

```
SELECT * FROM Customers WHERE State=:state AND City=:city
```

and, as before, the **ID** attribute of each SQL Parameter element must match a placeholder *name*.
If you're using **Connection.OLEDB**, you use this notation:

```
SELECT * FROM Customers WHERE State=? AND City=?
```

Unlike the previous examples, these are *positional* parameters so, at runtime, placeholders in the statement will be replaced, starting with the first placeholder, by the values specified in the SQL Parameter elements, based on the elements' *top-to-bottom order* in the definition. The element IDs are ignored.

The example above illustrates the relationship, in this scenario, between SQL Parameter element order and placeholders in the statement.

SQL Parameters will not work when using **Connection.ODBC**. Consult your database documentation for its specific place-holder syntax.

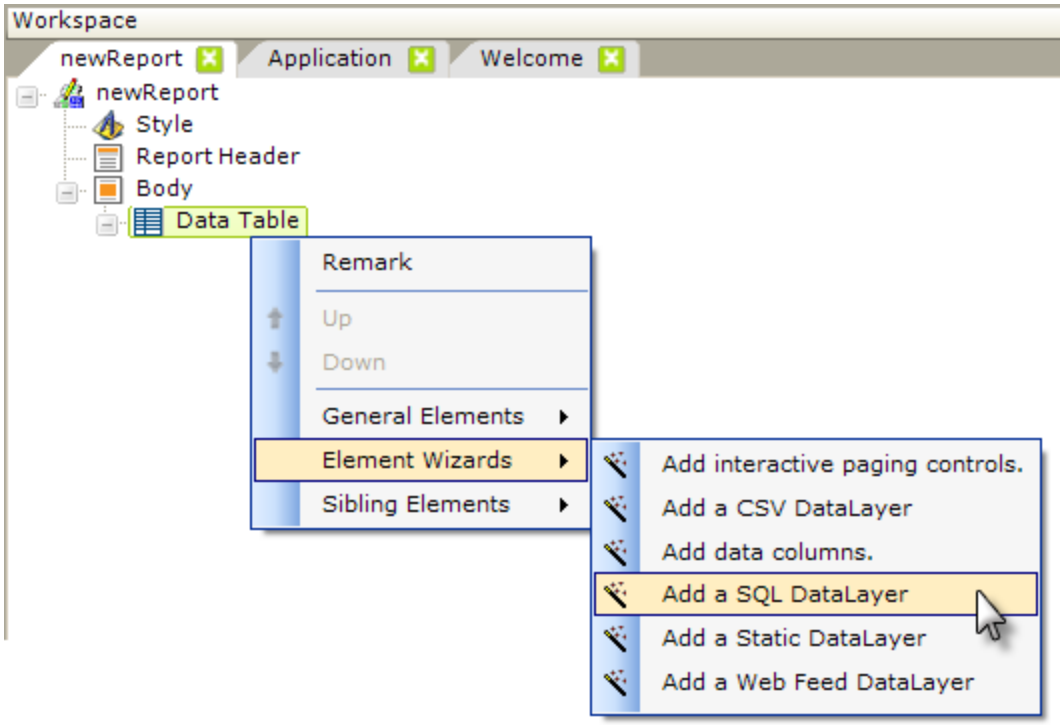
DataLayer.SQL - Writing Data to the Database

Logi Info applications are fully capable of writing data to a SQL database (the so-called "Writeback" capability) and can execute any valid SQL statement that does so, including INSERT, UPDATE, and DELETE.

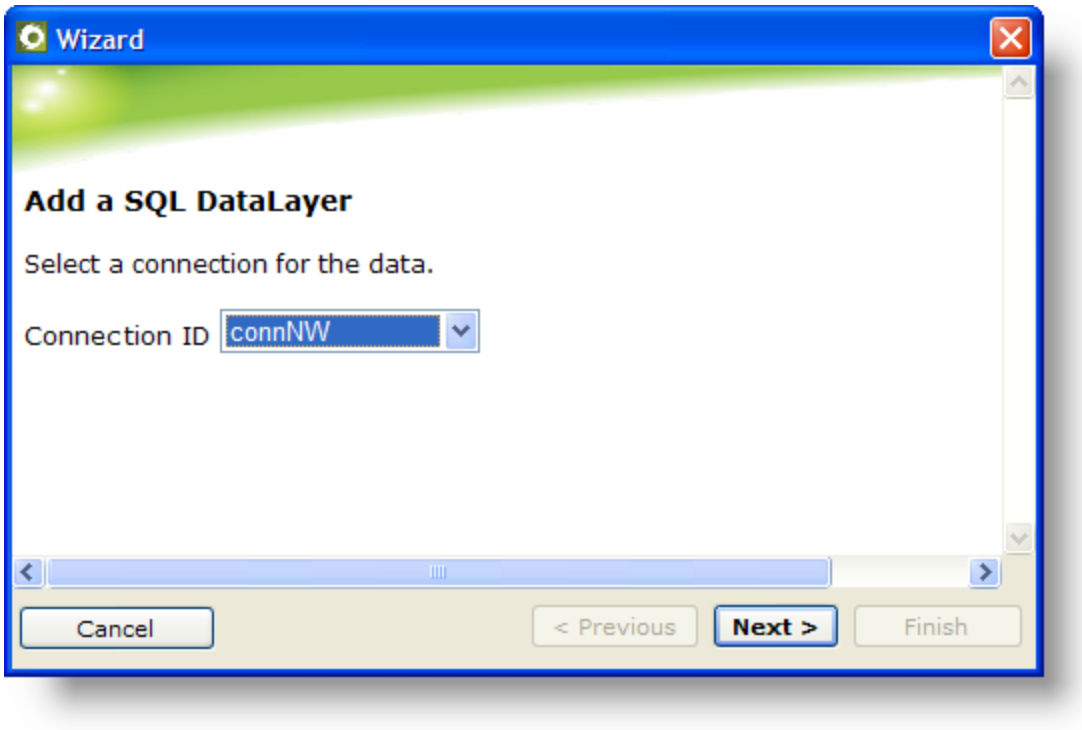
This can be done in combination with a SELECT operation, as shown in the data retrieval example above. More commonly, it's done independently in a Process Task using a **Procedure.SQL** element or by calling a stored procedure using a **Procedure.SP** element. Both of these techniques are discussed in detail in *Process Tasks*.

DataLayer.SQL - Using Studio's DataLayer Wizard

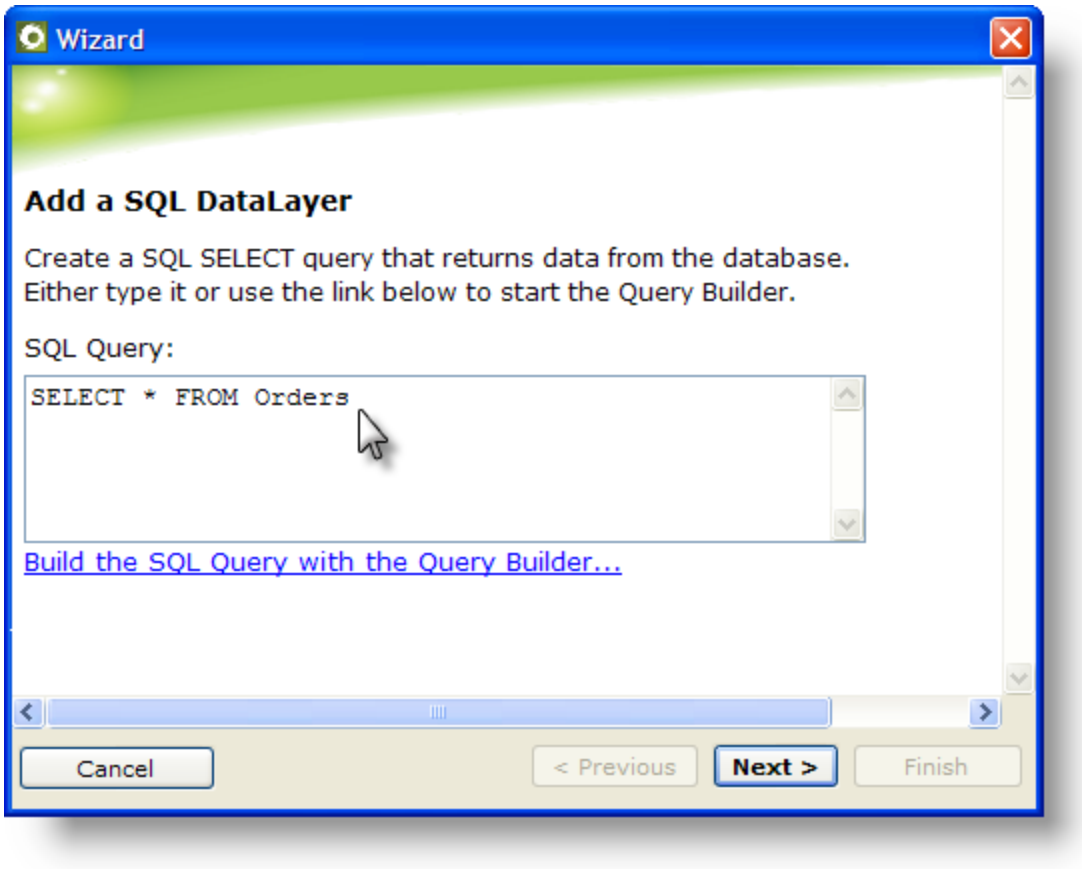
Studio includes a wizard that can assist you in configuring DataLayer.SQL. The wizard assumes that you have already added an appropriate database **Connection** element in the _Settings definition and configured it.



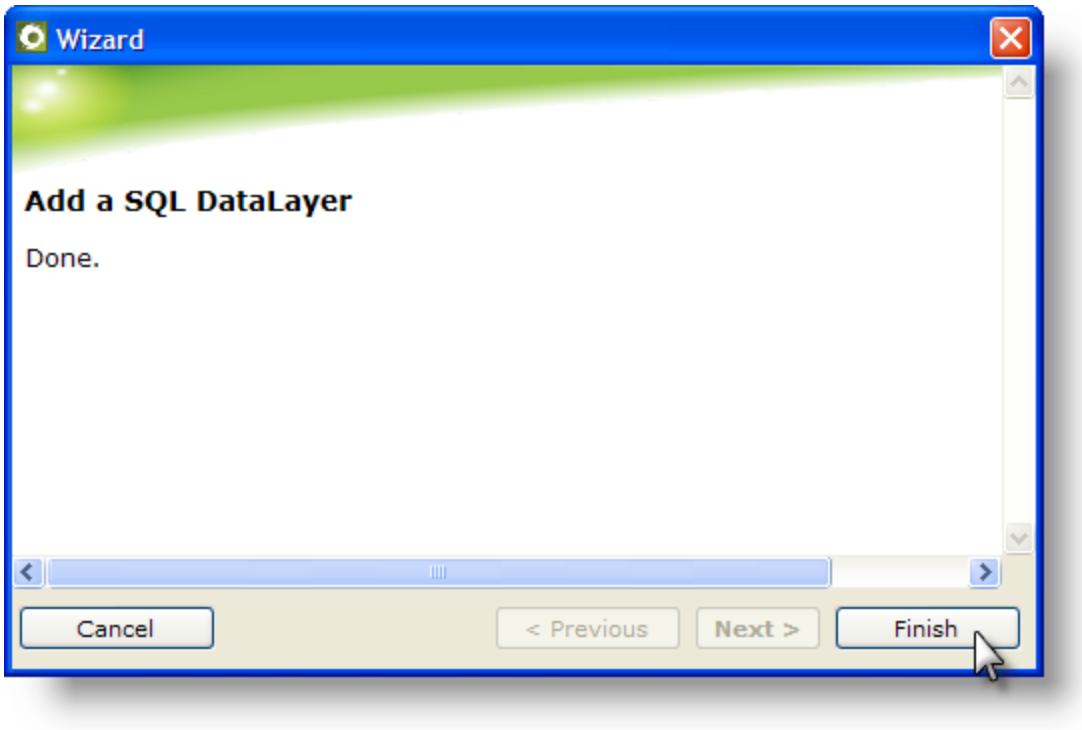
As shown above, the wizard can be started by selecting and then right-clicking the parent element under which you want to add the datalayer, and using the context menus to select "Add a SQL DataLayer". The wizard will open; use it as follows:



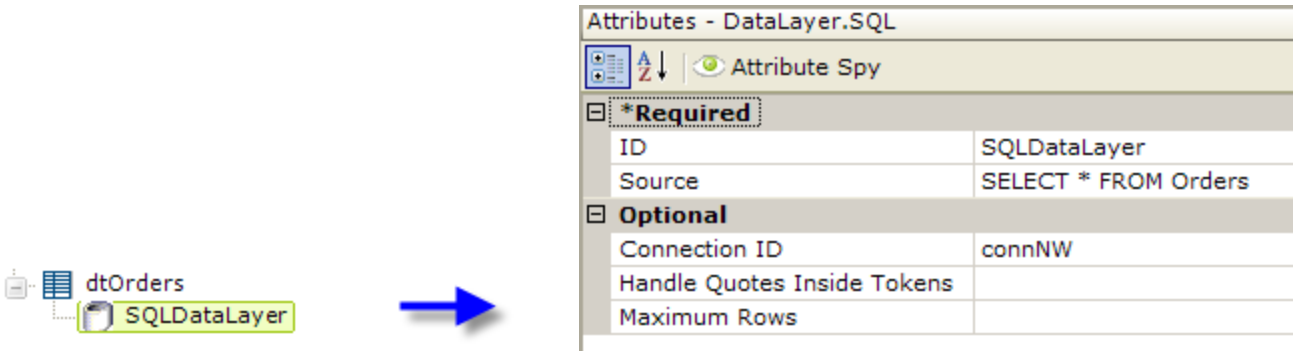
1. Select the ID of the Connection element from the drop-down list of available connections. Click **Next** to continue.



2. Enter the SQL query that will be used to retrieve data, either by typing it in directly or by building it using the Query Builder tool (see *Using Logi 12 Studio*). If you attempt to use the Query Builder and receive an error, type in a simple query instead and proceed with the wizard; you can adjust the query later. Click **Next** to continue.



3. Click **Finish** to close the wizard.



4. The wizard has inserted the datalayer and configured its attributes, as shown above.

DataLayer.Static

The **DataLayer.Static** element gives developers the ability create a datalayer and "hard-code" data rows directly into it. This can be useful for providing fixed options for Input controls, and for testing and debugging

The following topics introduce you to the DataLayer.Static element:

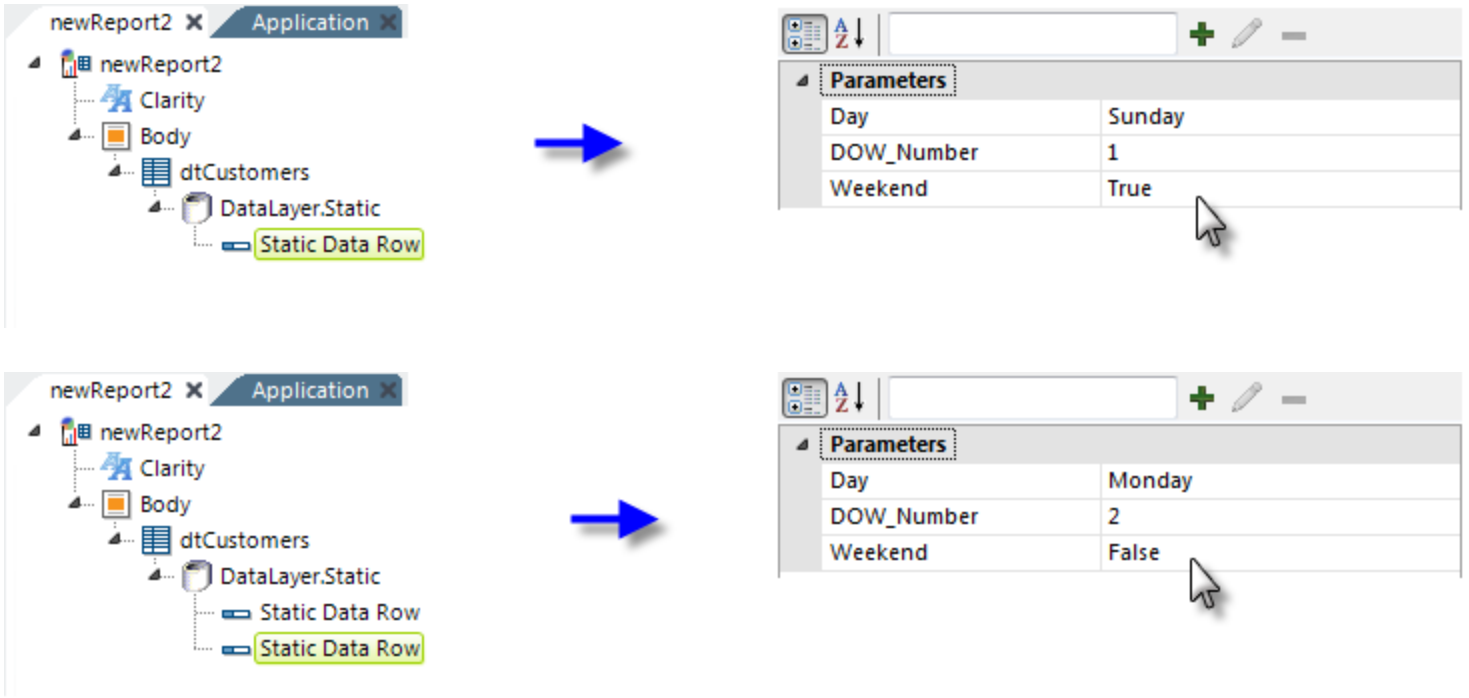
- Working with DataLayer.Static
- Using Studio's DataLayer Wizard

DataLayer.Static Attributes

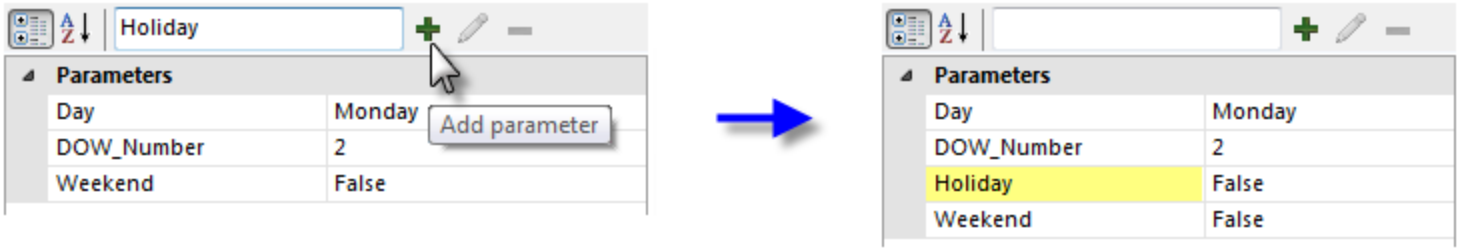
The **DataLayer.Static** element has no attributes other than an **ID**. It's not required but we recommend, for easier debugging, that you *always* provide a unique ID for datalayers.

DataLayer.Static - Working with DataLayer.Static

In order to actually create the data, developers must add one or more **Static Data Row** elements beneath a DataLayer.Static element. Each Static Data Row element represents a *single row* of data in the datalayer.

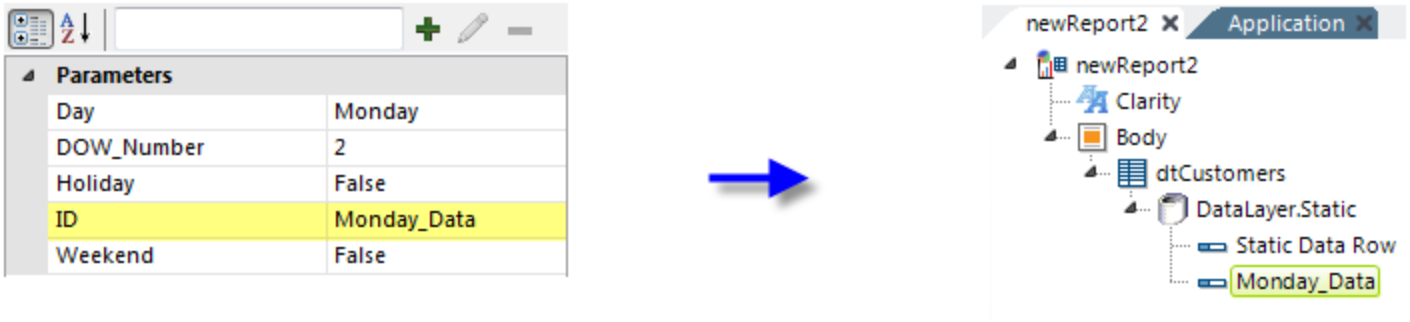


As shown above, **Static Data Row** elements have dynamic attributes, which the developer creates. Each attribute represents a *column* in the data row; the attribute name assigned by the developer becomes the column name and the value entered becomes the column data. Naturally, if multiple Static Data Row elements are being added, they all need to have the same number of attributes, with identical names.



Dynamic attributes are **added** by entering the attribute name in the input box at the top of the Attributes Panel, as shown above, and clicking the "**+**" icon. Values are then added by entering them in the usual fashion. An attribute and its value, if any, are **deleted** by selecting it and clicking the red "**-**" icon.

Dynamic attribute names can be **edited** by selecting them and clicking the "Pencil" icon. This will place the name into the top text box, where it can be edited. The icon will change to a "Diskette" icon and clicking it again will save the edited attribute name. Attribute values can be edited directly, as usual, by placing the cursor on them



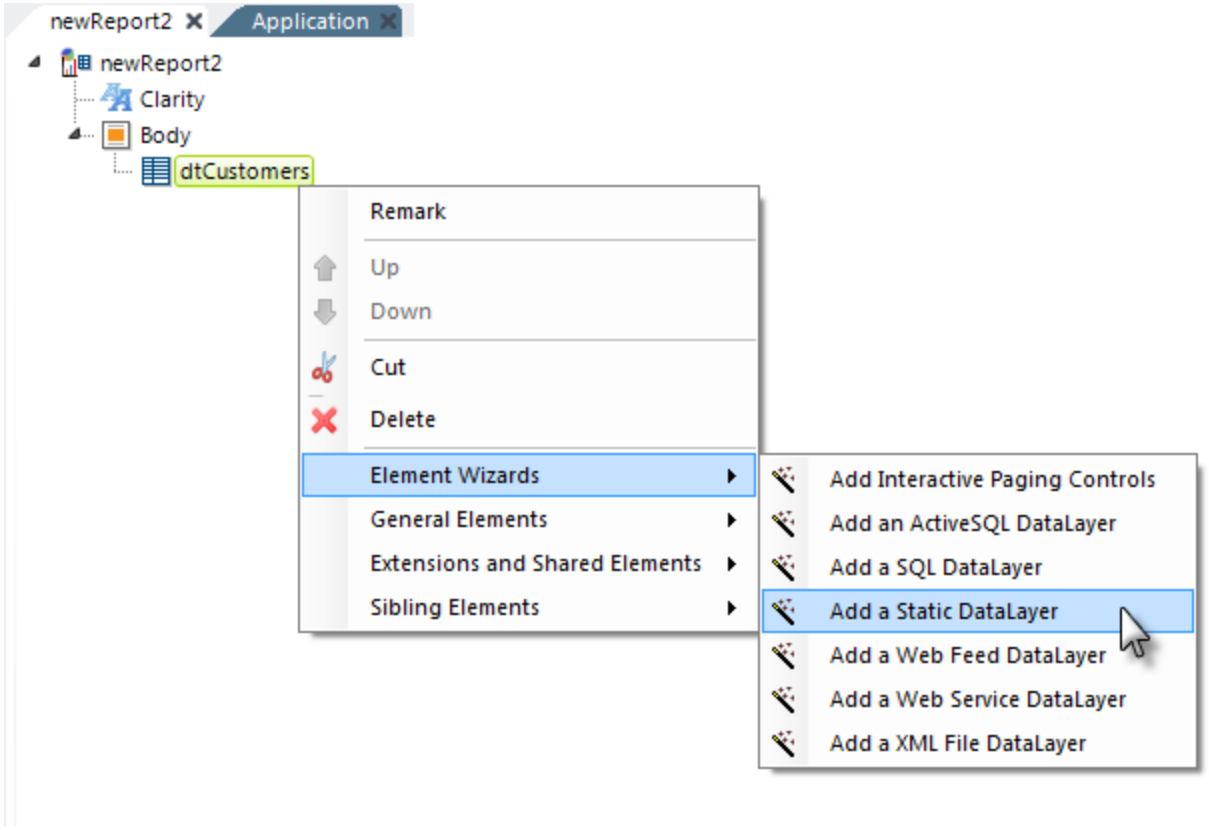
If an attribute named "ID" is created, as shown above, its value will replace the words "Static Data Row" next to the element icon in the Element Tree. This may be more desirable than a big stack of anonymous Static Data Row elements, as it allows developers to visually identify each static data row element separately in the tree.

Developers can retrieve the values from each column in the datalayer by using the @Data token.

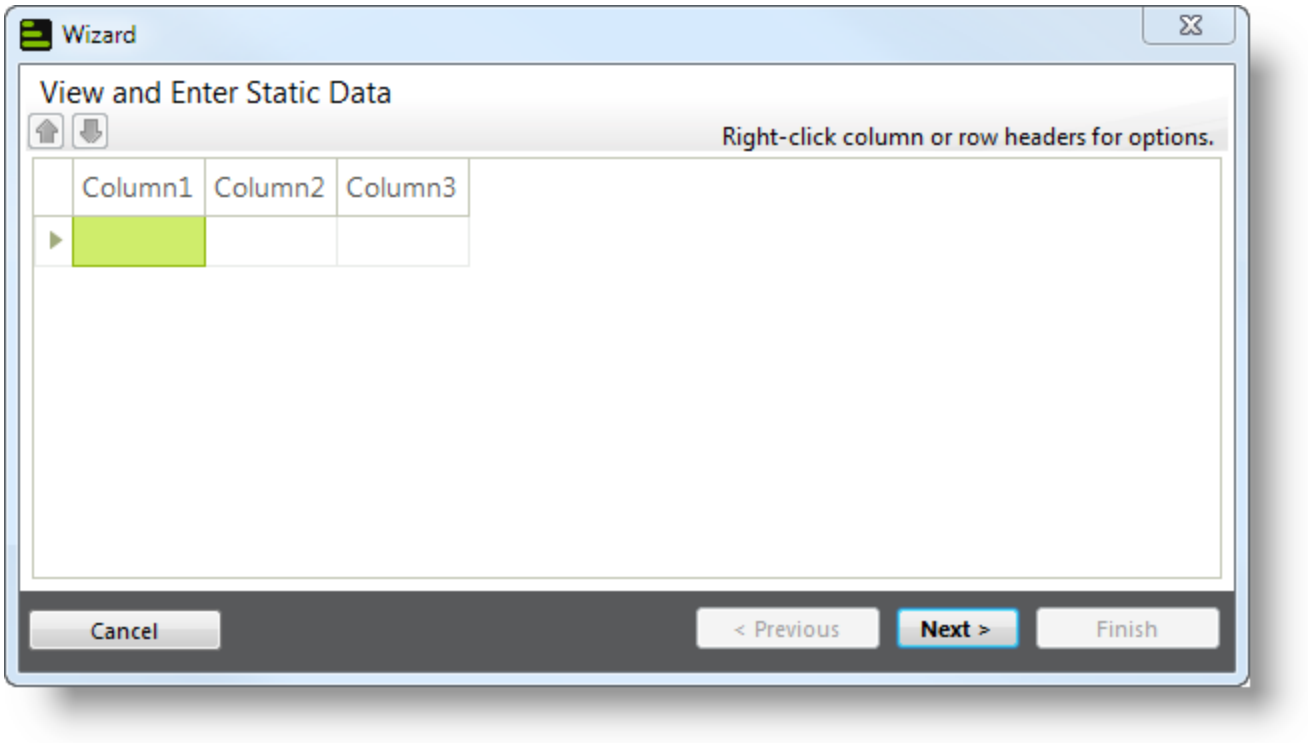
Data in a DataLayer.Static can be manipulated with all of the usual child elements, such as **Sort Filter**, and can be joined using the **Join** element to other datalayers.

DataLayer.Static - Using Studio's DataLayer Wizard

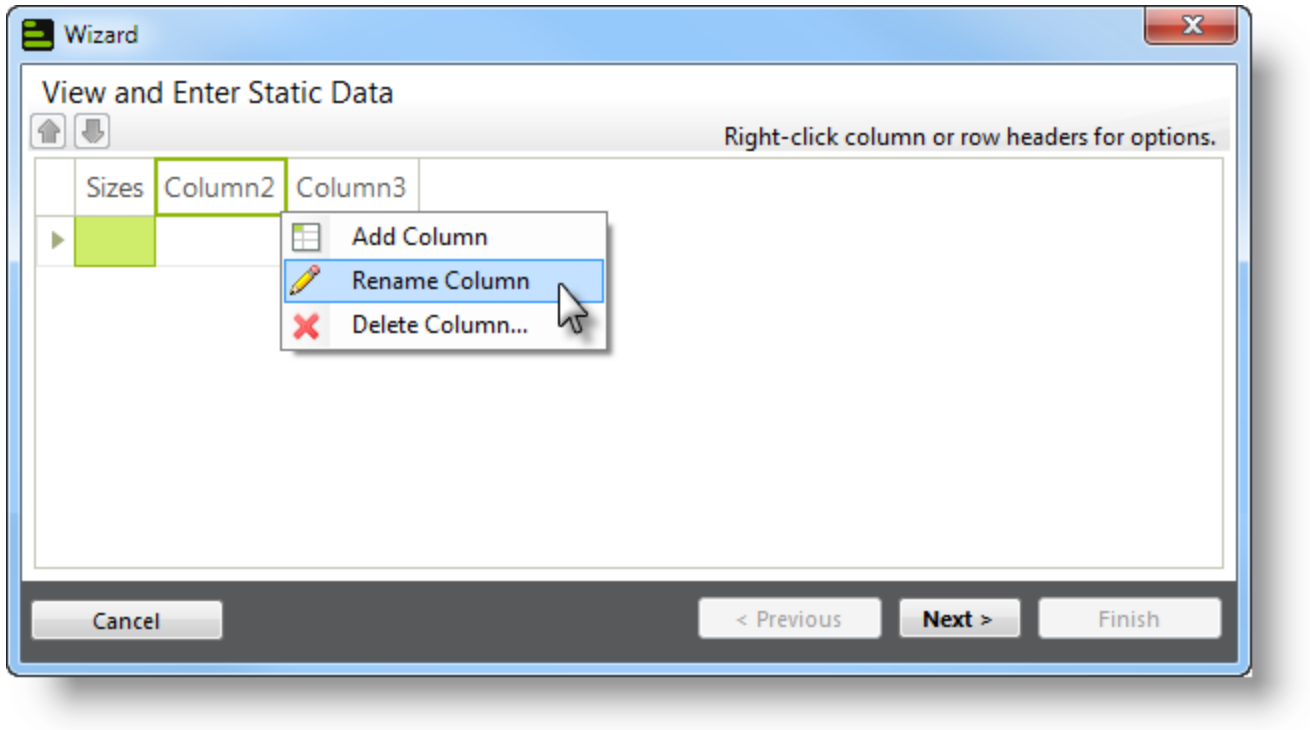
Logi Studio includes a wizard that assists you in configuring DataLayer.Static and creating static data.



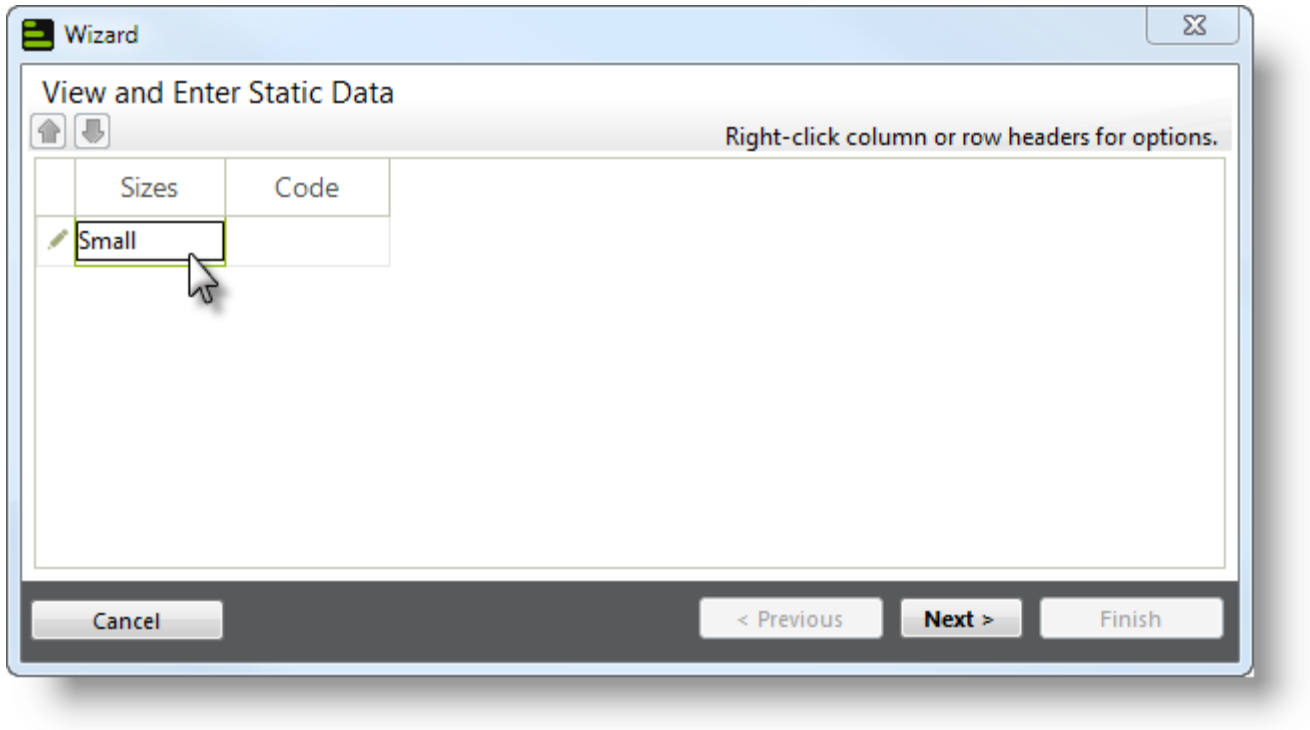
As shown above, the wizard can be started by selecting and then right-clicking the parent element under which you want to add the datalayer, and using the context menus to select "Add a Static DataLayer".



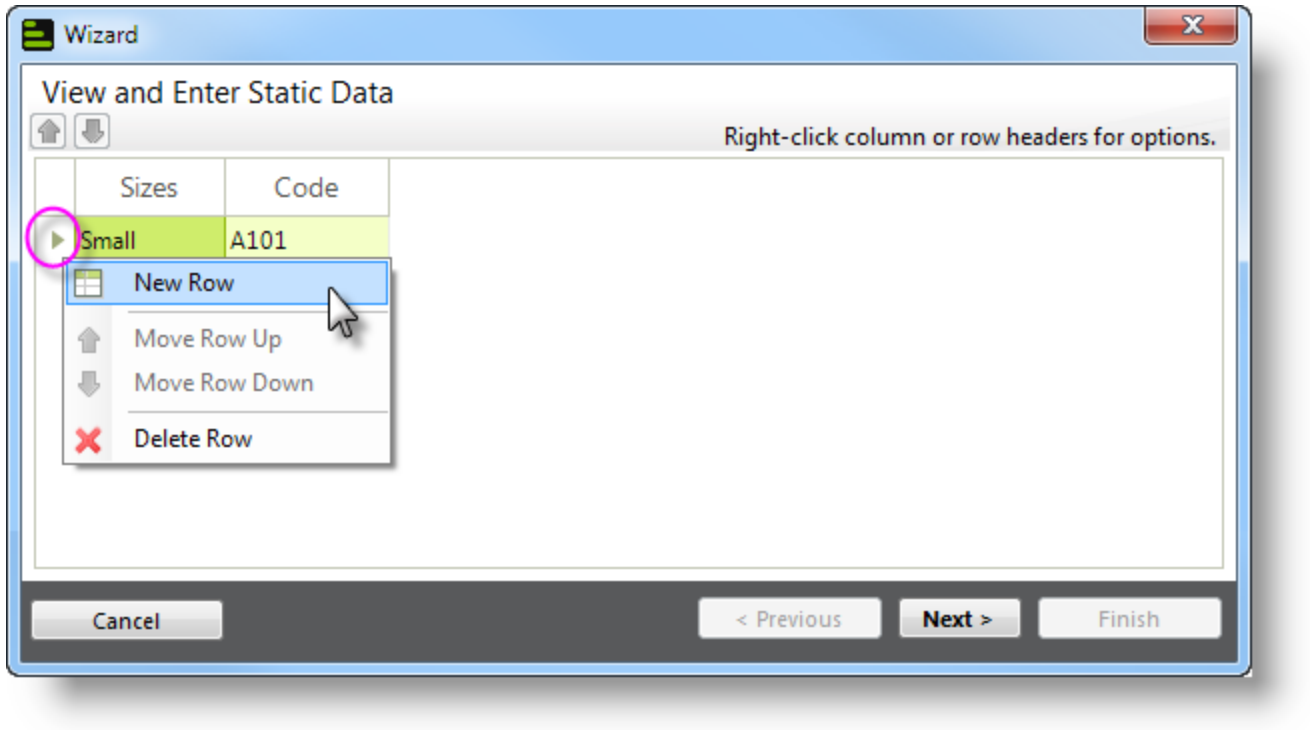
The wizard will open, as shown above. Use the wizard as follows:



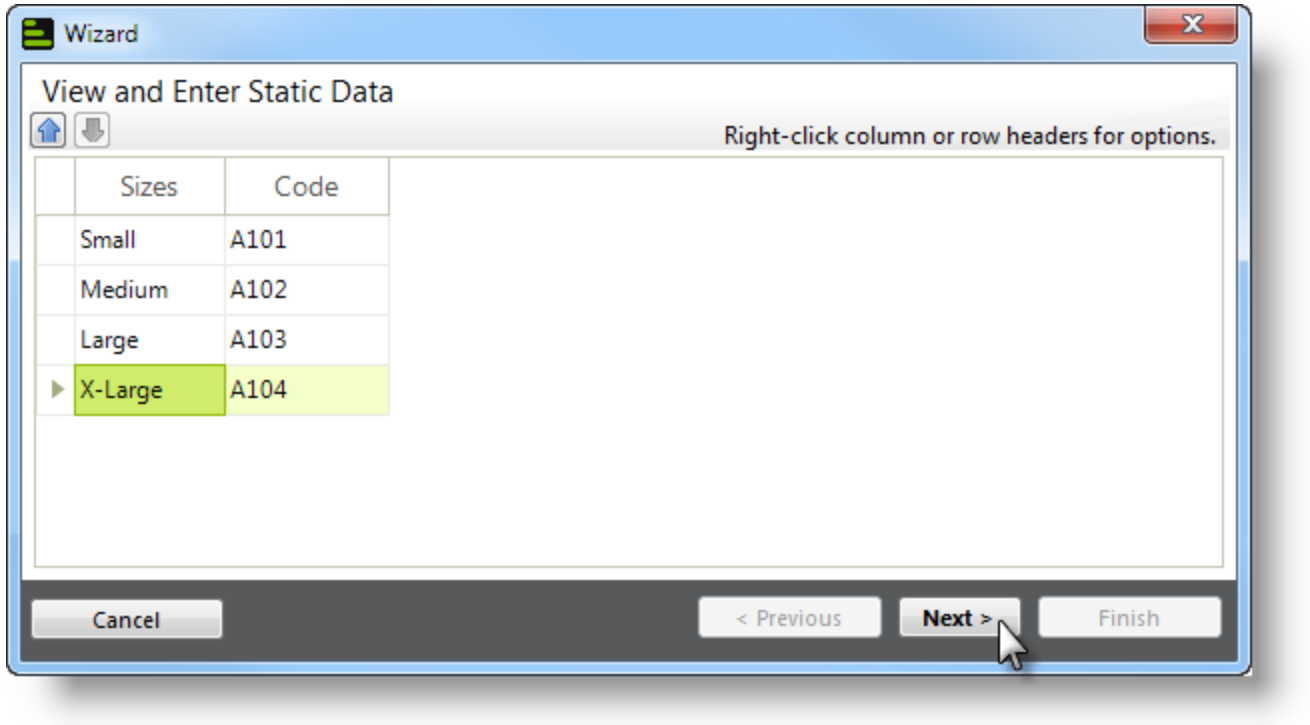
1. Right-click the column headers, one-by-one, select **Rename Column** from the pop-up menu, and change the column names to names appropriate for your purpose. You can drag the column header borders to widen the columns, if desired. Add or delete columns as necessary, until you have just those you need.



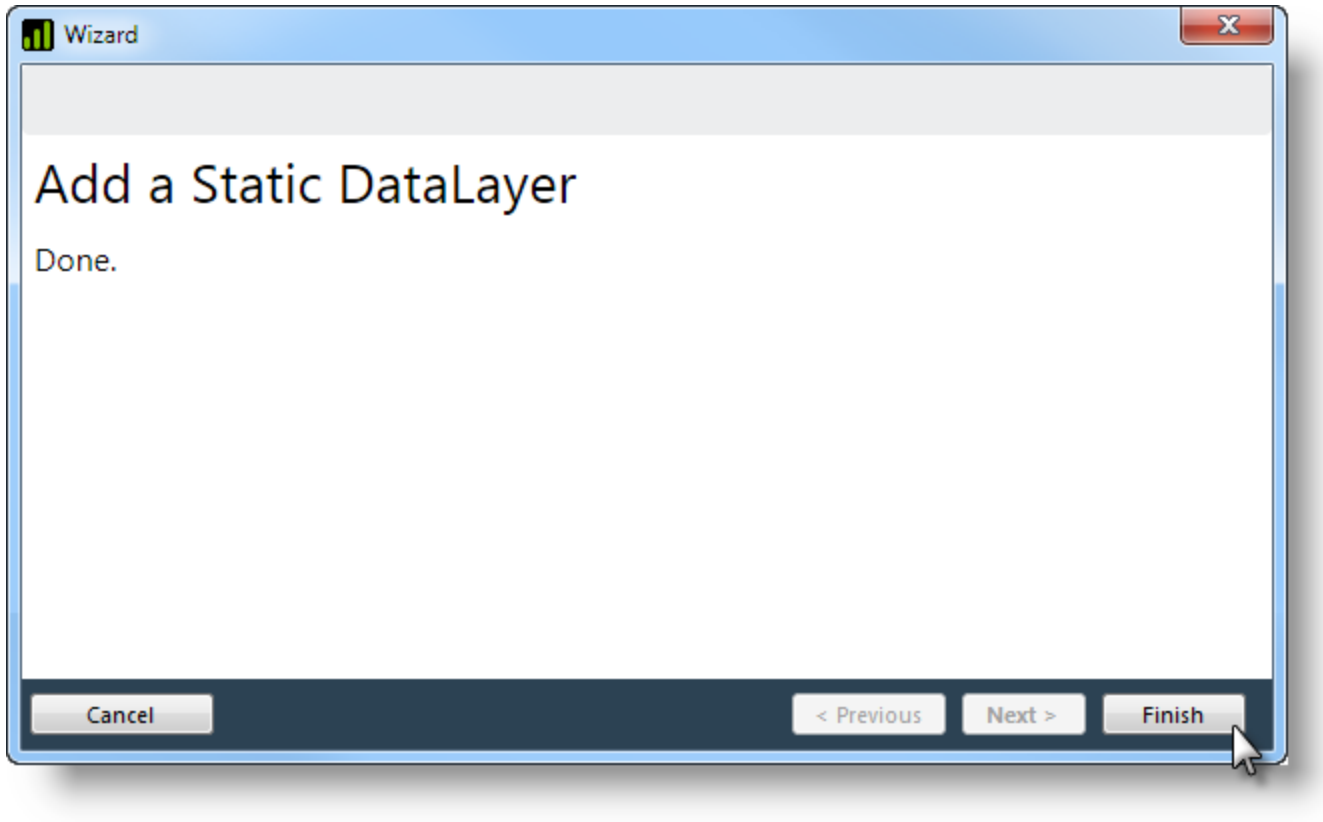
2. Click the first column and enter the desired data value. Press **Tab** to move to the next column and enter its data, then repeat for all columns. *Do not press Enter*!



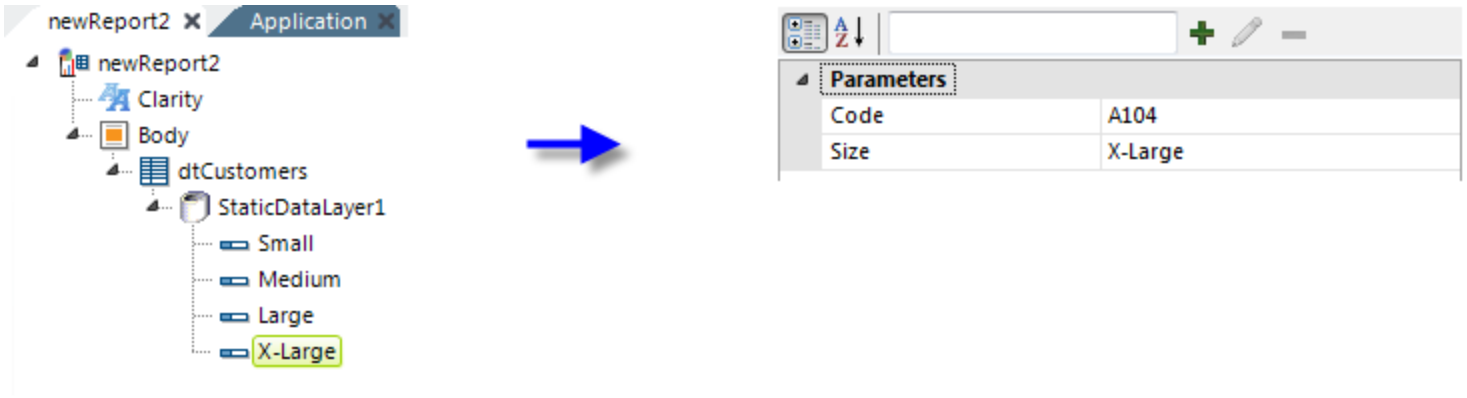
3. Right-click the small arrow icon to the left of the first column and select **New Row** from the pop-up menu that appears. Repeat Steps 2 and 3 as many times as necessary to enter data for all static rows.



4. When the data for all rows and columns has been entered, click **Next** to insert the elements.



5. Click **Finish** to close the wizard.



The Element Tree will now include your new datalayer and supporting elements, as shown above.

DataLayer.WebFeed

The **DataLayer.WebFeed** element retrieves data from an RSS or Atom web feed, such as those available on news sites, blogs, and other frequently-updated web information sources, in a convenient form.

The following topics introduce the DataLayer.WebFeed element:

- DataLayer.WebFeed Attributes
- Working with DataLayer.WebFeed
- Using Studio's DataLayer Wizard

DataLayer.WebFeed - Attributes

The DataLayer.WebFeed element has the following attributes:

Attribute	Description
ID	Specifies an optional element ID. Recommended for easier identification when debugging.
Web Feed URL	Specifies the address (URL) of a RSS or Atom web feed, e.g. `https://news.google.com/rss`
Include Headers	Specifies whether or not feed-level elements (management information) is included with the usual news item information. Default: *False*

DataLayer.WebFeed - Working with DataLayer.WebFeed

The datalayer connects directly to the web feed and retrieves the information. Unlike most datalayer elements, this one does not requred a Connection element. Feed information is retrieved and cached as rows and columns, with one row per news item. Data retrieved into the datalayer is cached in XML format.

The data retrieved with a datalayer is available using @Data tokens, in the format @Data.*ColumnName*~. The spelling of the column name is *case-sensitive*. The data is only available within the scope of the parent element of the datalayer, not throughout the entire report definition. The DataLayer.Linked element can be used to make the data reusable in another datalayer outside this scope.

"Column names" may vary depending on the feed, however the column names for news items will all start with "item_" and for header values with "channel_". Examples of possible valid "column names" available through the @Data token are:

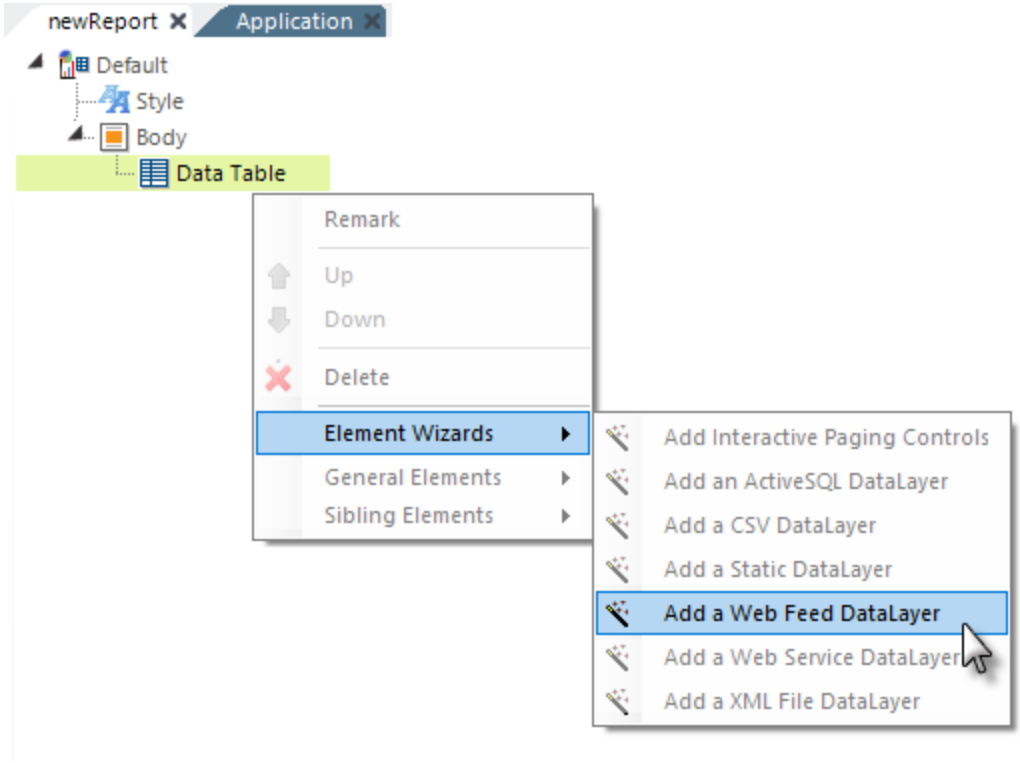
Column	Description
item_title	The title of the new item
item_link	A link to the item. If the feed is a news aggregator, this link may have embedded in it the URL of the source for this item
item_guid	A unique identifier used by the feed for this item
item_category	A categorization, specific to the feed, of this item
item_pubDate	A publication timestamp for this item

Column	Description
item_description	Text of the actual item, possibly in HTML format
channel_pubDate	A publication timestamp for this item
channel_ lastBuildDate	Timestamp for the last assembly of the feed page
channel_ webMaster	Email address of the feed page web master
channel_image_url	URL for images associated with feed
channel_copyright	HTML copyright message
channel_language	Language designator abbreviation
channel_image_ link	URL for image associated with item

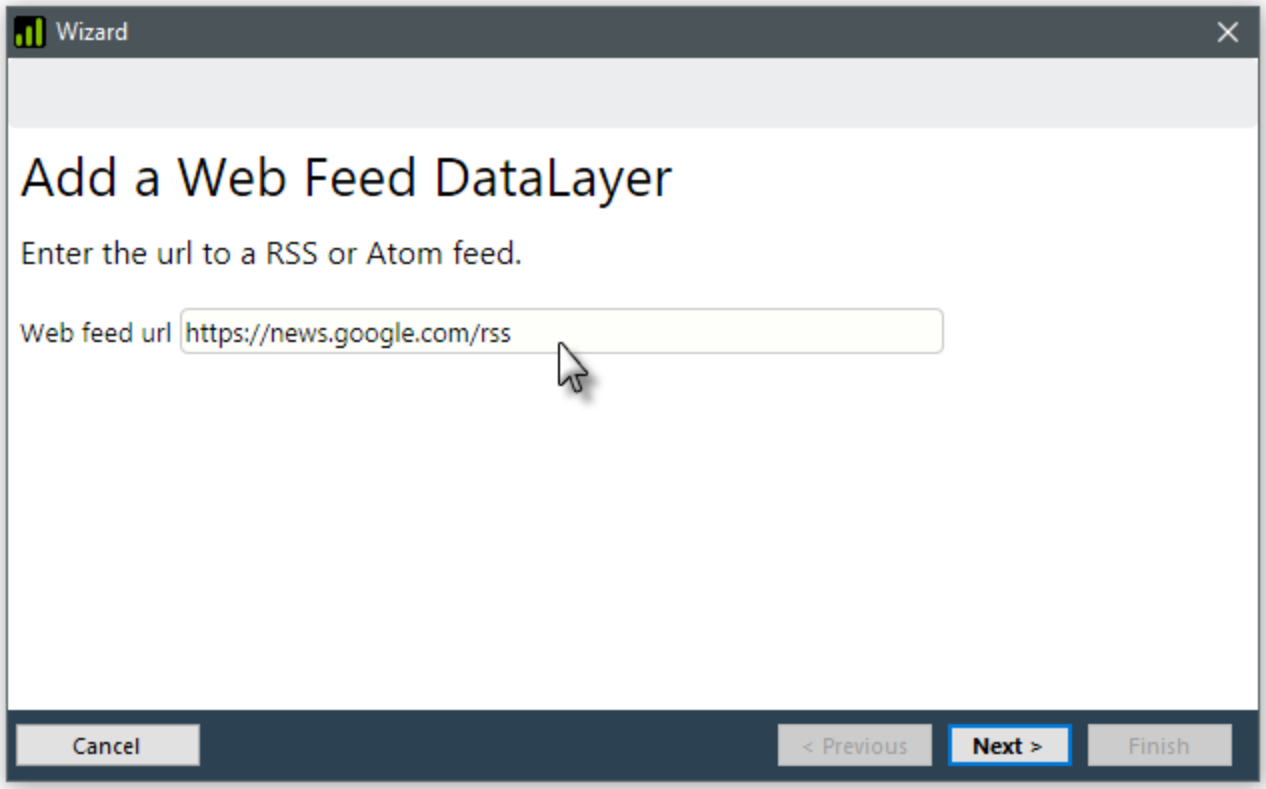
The *Auto Columns* element can be used to quickly see which "column names" are being returned by the feed. You can view the data retrieved into the datalayer by turning on the Debugger Link in the _Settings definition (General element) and using the resulting icon at the bottom of the report page to view the Debugger Trace Page. A link on the trace page will display the retrieved data.

DataLayer.WebFeed - Using Studio's DataLayer Wizard

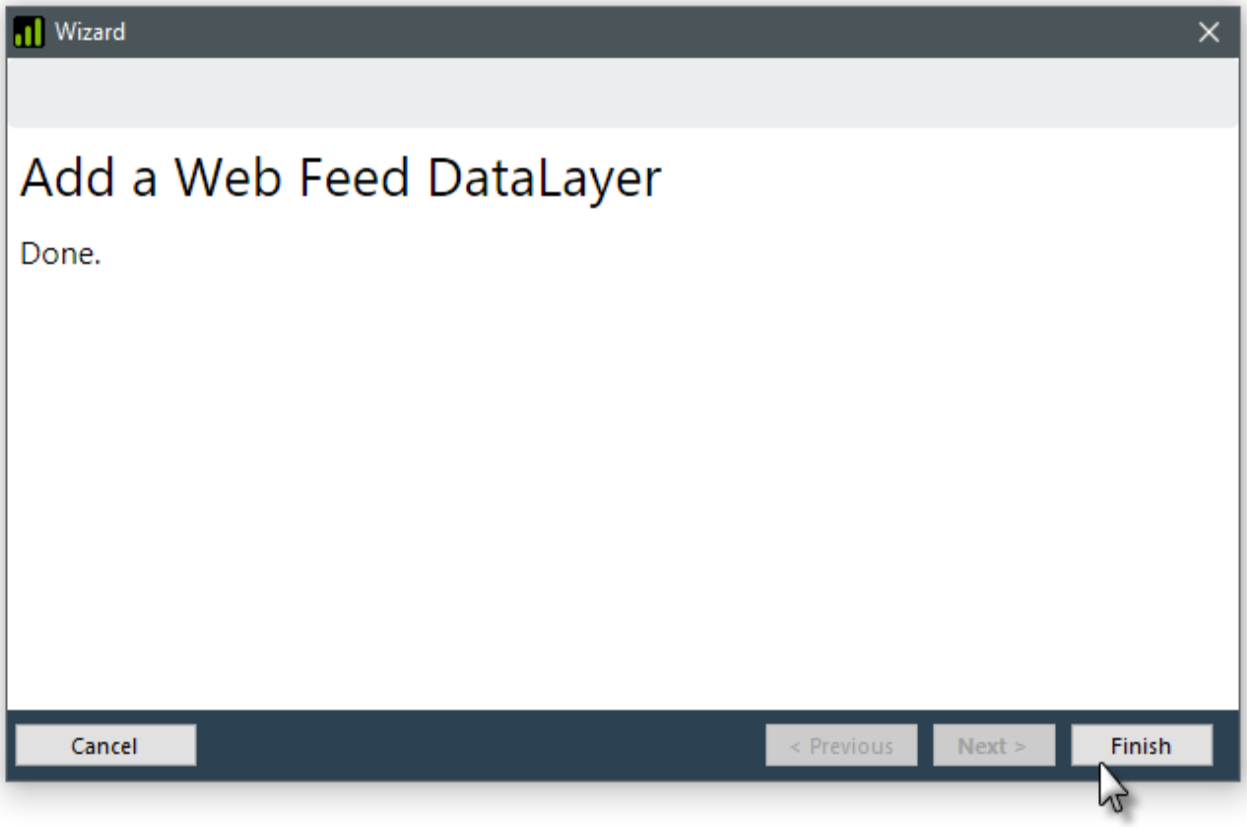
Logi Studio includes a wizard that can assist you in configuring DataLayer.Web Feed.



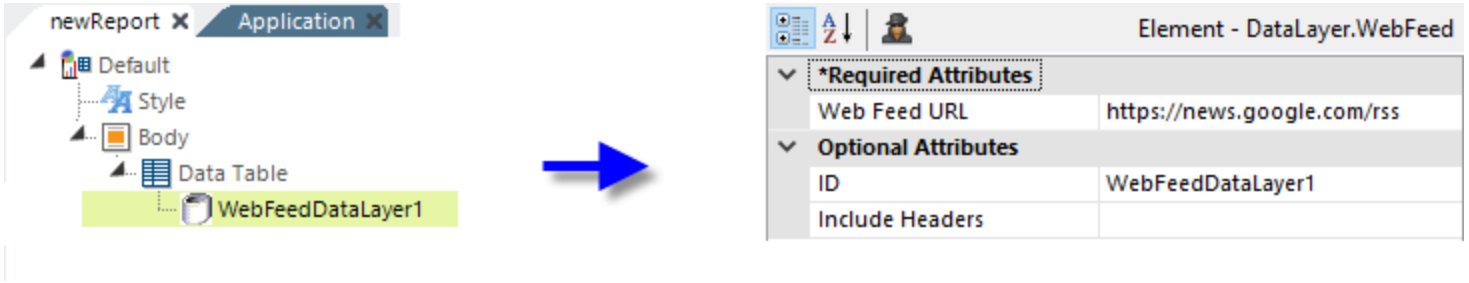
As shown above, the wizard can be started by selecting and then right-clicking the parent element under which you want to add the datalayer, and using the context menus to select "Add a Web Feed DataLayer". The wizard will open; use it as follows:



1. Enter the URL for the web feed. Click **Next** to continue.



2. Click **Finish** to close the wizard.



3. The wizard has inserted the datalayer and configured its attributes, as shown above.

DataLayer.Web Scraper

The **DataLayer.Web Scraper** element "scrapes", or retrieves, content from *within* web pages and organizes it into the row-column datasets usually found in datalayers. This topic discusses this datalayer.

The following sections are covered in this topic:

- Attributes
- Working with DataLayer.Web Scraper


The DataLayer.Web Scraper element is *not* available in Logi Java applications. More information about this interesting element can be found in "Web Page Screen Scraping" on page 188.

Attributes

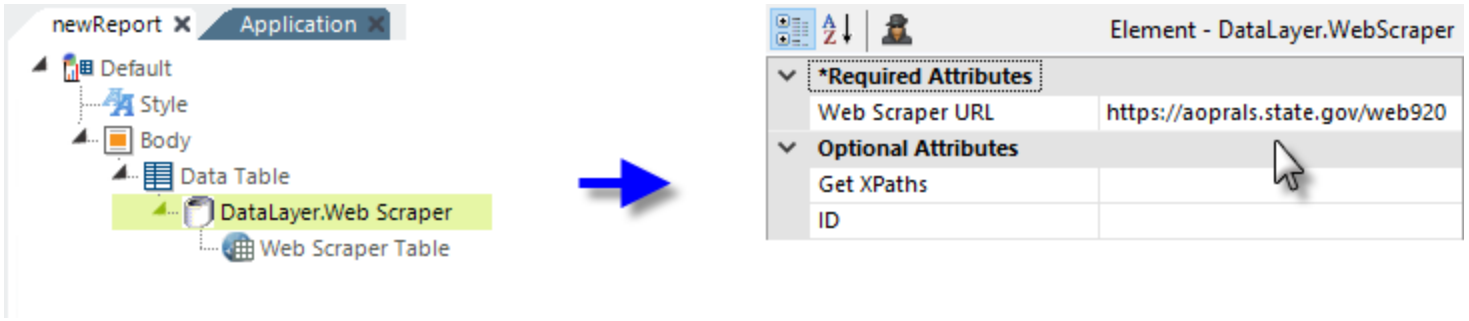
The **DataLayer.Web Scraper** element has the following attributes:

Attribute	Description
ID	Specifies an optional element ID. Recommended for easier identification when debugging.
Web Scraper URL	(Required) Specifies the address of the web page from which data will be retrieved; can be either an Internet HTTP address or a fully-qualified file path and name. A wide variety of file types are valid, including .htm, .html, .xhtml, and .mht, among others.
Get XPaths	(Development only) When this attribute is used, a document of the XML nodes on the target web page is created, for use as a reference during development, but data is not returned into the datalayer in an accessible form. If

Attribute	Description
	the attribute value is set to *All*, the document that's created lists all XML nodes in the target web page and includes the XPath strings that can be used to access each node. If set to a specific *XPath string*, the document that's created lists only the XML nodes in the web page that match that string. If left *blank*, no nodes document is created and data is returned into the datalayer; however either a **Web Scraper Rows** or **Web Scraper Table** element is required as a child of the datalayer for proper operation.

Working with DataLayer.Web Scraper

In most respects, **DataLayer.Web Scraper** functions exactly as other datalayer elements do and its data can be filtered and conditioned using appropriate elements. However, there is no need to use a Connection element in the _Settings definition with this element. The DataLayer.Web Scraper element directly accesses the Internet or file system to read the specified web page or file.



As shown above, a **DataLayer.Web Scraper** element is added as a child element to a Data Table or other data container element. Its attributes are set so that it accesses the desired web page or file.

A **Web Scraper Rows** or **Web Scraper Table** (shown) child element must be used with this datalayer.

When using **Web Scraper Rows**, the rows of text identified will be retrieved into the datalayer and its columns will be named based on the element IDs of its child **Web Scraper Column** elements, which are used to identify individual nodes in the text rows.

When using **Web Scraper Table**, the rows of the table identified will be retrieved in the datalayer and the columns will be named based on the table column headers.

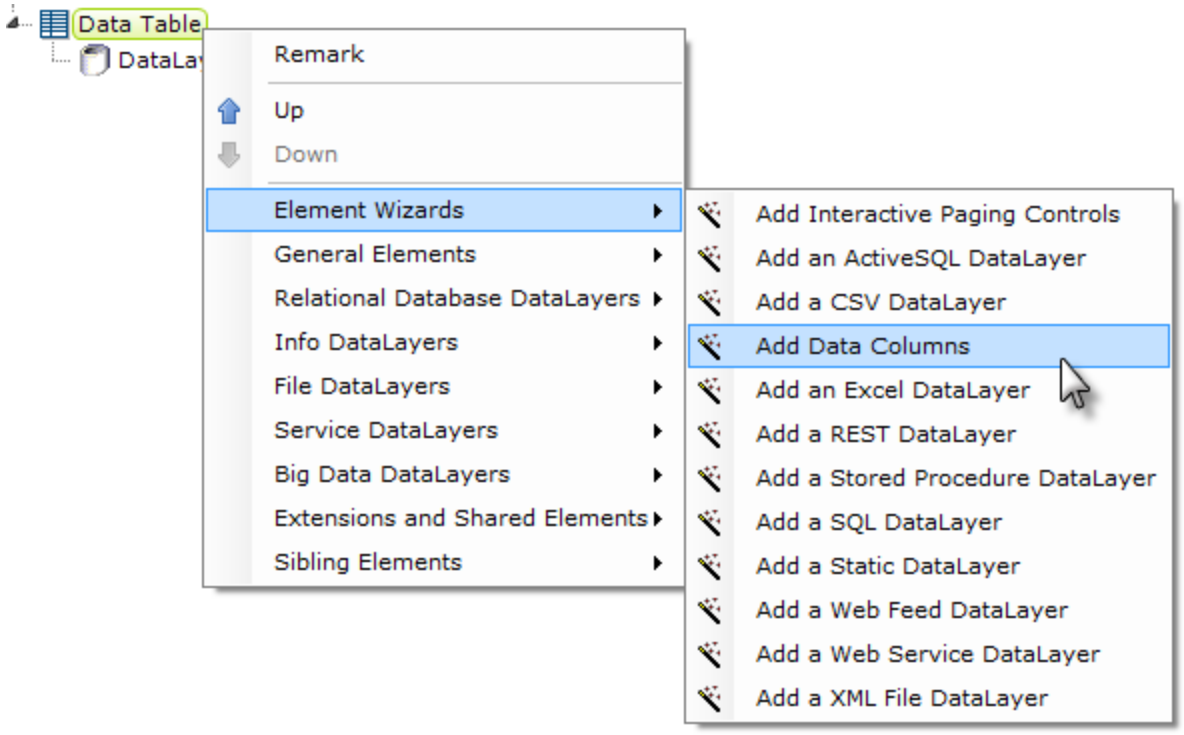
The datalayer reads and caches the data from the site or file. You can add child elements beneath the datalayer to affect its contents, including:

- **Filtering**: Sort, group, or restrict the data
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled data values
- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report definitions

Data retrieved into the datalayer is cached in **XML** format, in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

The data read with the datalayer is available using **@Datatokens**, in the format @Data.*ColumnName*~. The spelling of the column name is *case-sensitive*. The data is only available within the scope of the parent element of the datalayer, not throughout the entire report definition. The **DataLayer.Linked** element can be used to make the data reusable in another datalayer outside this scope.

The following three paragraphs *do not apply* if you're using the **Get XPaths** attribute to create a XML nodes reference document: The *Auto Columns* element can be used to quickly display in your report all the data in a datalayer. The data retrieved into the datalayer can be viewed by turning on the Debugger Link in your _settings definition (General element) and using the resulting iconat the bottom of your report page to view the Debugger Trace Page. A link on the trace page will display the retrieved data.



Studio includes a wizard that will add elements for the columns in the datalayer to your definition. You can select the desired columns before the operation begins. With the datalayer's parent **Data Table** or similar element selected in the Definition Editor, click the wizard link, shown above, in the Element Toolbox.

Web Page Screen Scraping

The **DataLayer.Web Scraper** element "scrapes", or retrieves, content from *within* web pages and organizes it into the row/-column datasets usually found in datalayers. This provides an easy way for developers to automate the collection and use of data published on other web sites or in HTML files.

The following topics discuss techniques for using the datalayer:

- Using "Get XPaths" and Web Scraper Table
- Using Web Scraper Rows and Web Scraper Column

 The linked topics illustrate techniques by referring to public websites. While we make an effort to stay aware of them, these sites may change their coding from time to time without our knowledge, invalidating the actual data and values depicted, so you may not be able to achieve the identical results. The techniques themselves, however, remain valid.

About "Web Page Scraping"

"Screen scraping" has been around for a long time; it's the process of **extracting** data from the text on a **display** screen, usually based on fields or screen regions. A key issue is that the data is intended to be *viewed* and is therefore neither documented nor structured for convenient parsing. In the age of the Internet, the term has also come to apply to the same process with web pages. In many cases, complicated parsing code is required.

However, the Logi approach makes the process fairly easy. Generally, in a Logi application, the target HTML web page is **read** (the entire page, not just the portion displayed), **transformed** into XML, and then parsed based on **XPath** identifiers. Data formatted into tables, and even in less-structured blocks of text, can readily be parsed and served up to the report developer in the standard Logi datalayer row/column structure. The data can then be further processed (filtered, grouped, aggregated, etc.) using all the usual datalayer child elements. This is all done using the **DataLayer.Web Scraper** element and its child elements.

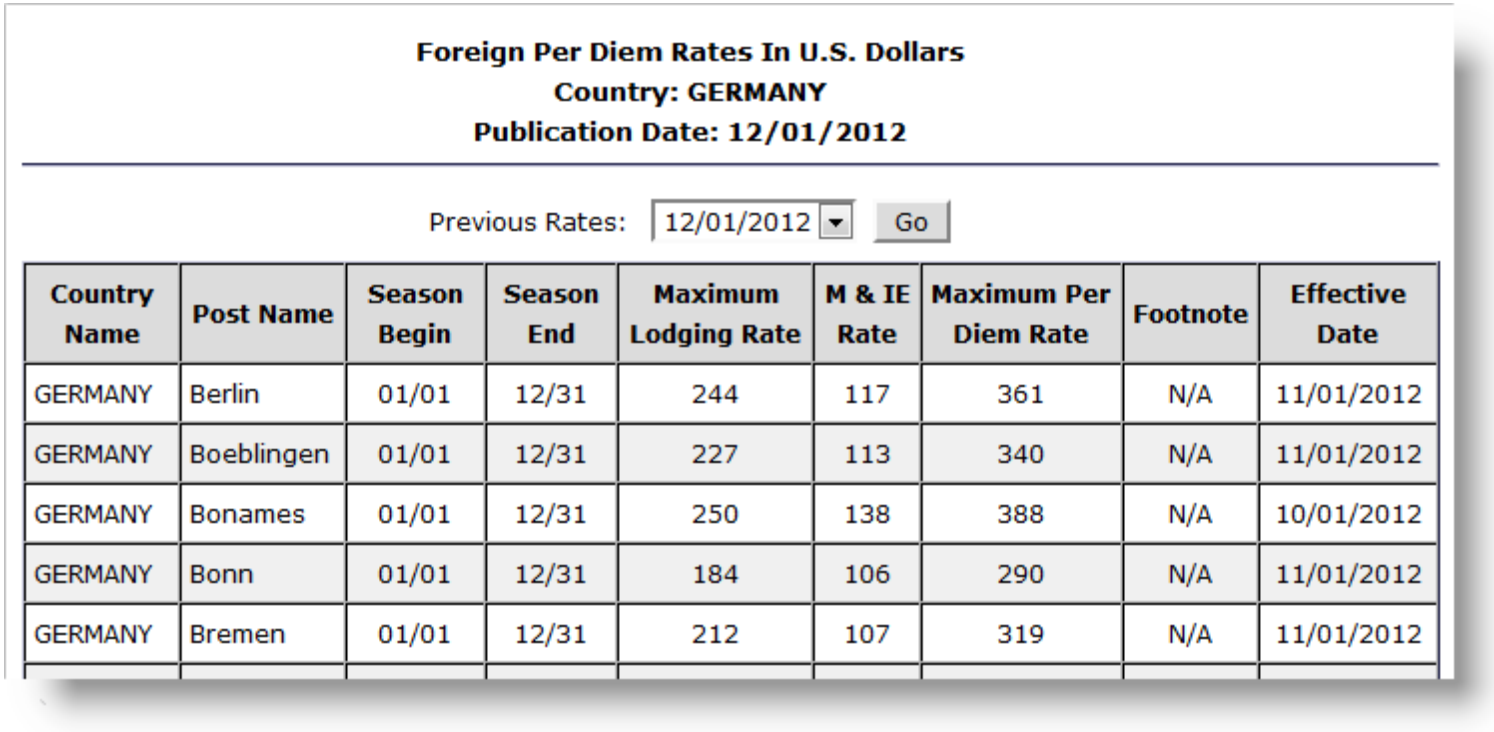
The DataLayer.Web Scraper element is *not* available in Logi Java applications.

For purposes of convenience, this topic refers to "HTML files" but file extensions are not really important; if the file will render to a web page, the datalayer will use it. Pages with .htm, .html, xhtml, .mht and other extensions are all valid.

One of the keys to this process is specifying the XPath **parsing identifiers** and DataLayer.Web Scraper provides a mechanism for assisting developers in selecting them, as discussed in "Using "Get XPaths" and Web Scraper Table" on the next page.

Using "Get XPaths" and Web Scraper Table

The DataLayer.Web Scraper element has an attribute named **Get XPaths**. During development, when this attribute is used, the target HTML page is read and transformed into XML, then a **reference document** is created that lists all of the XPath identifiers and their values. Looking through this topic, the developer can quickly see, for example, the XPath tag name for a table with the desired data.

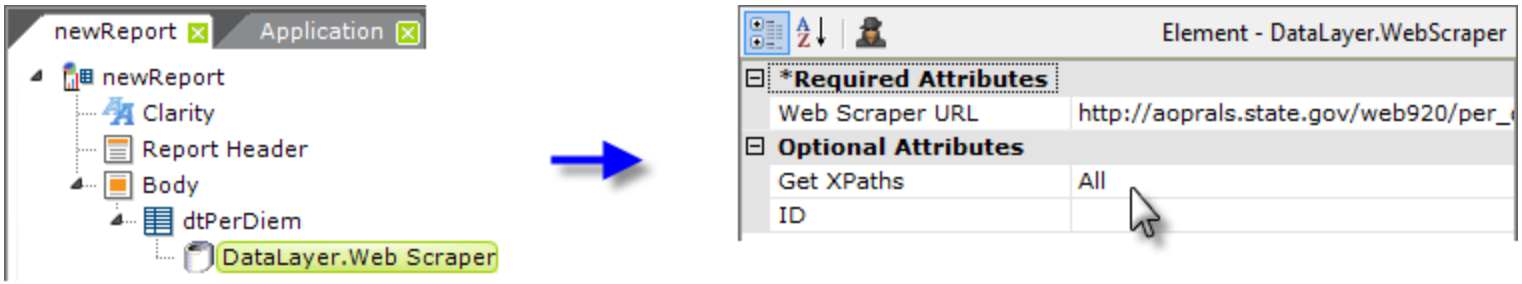


Foreign Per Diem Rates In U.S. Dollars
Country: GERMANY
Publication Date: 12/01/2012

Previous Rates: 12/01/2012 Go

Country Name	Post Name	Season Begin	Season End	Maximum Lodging Rate	M & IE Rate	Maximum Per Diem Rate	Footnote	Effective Date
GERMANY	Berlin	01/01	12/31	244	117	361	N/A	11/01/2012
GERMANY	Boeblingen	01/01	12/31	227	113	340	N/A	11/01/2012
GERMANY	Bonames	01/01	12/31	250	138	388	N/A	10/01/2012
GERMANY	Bonn	01/01	12/31	184	106	290	N/A	11/01/2012
GERMANY	Bremen	01/01	12/31	212	107	319	N/A	11/01/2012

The example above shows part of the U.S. State Department web site page that publishes per diem rates. The following example shows how to use DataLayer.Web Scraper to get the XPath tags needed to parse this page.



As shown above, add a **Data Table** element with a **DataLayer.Web Scraper** child element the report definition. Set the datalayer's **Web Scraper URL** attribute value to the URL of the target web page:

```
http://aoprals.state.gov/web920/per_diem_action.asp?MenuHide=1&CountryCode=1089
```

and set its **Get XPaths** attribute to *All*, which causes the development reference document to be created.

XPath by ID	XPath by Tag Name	Name	Value
	/html	html	U.S. Department of State<!DOCTYPE html PUBLIC "-//W3C//DTD X...
	/html/head	head	U.S. Department of State
	/html/head/title	title	U.S. Department of State
	/html/head/meta	meta	
	/html/head/meta/@name	@name	keywords
	/html/head/meta/@content	@content	
	/html/head/meta[2]	meta	
	/html/head/meta[2]/@name	@name	description
	/html/head/meta[2]/@content	@content	
	/html/head/link[3]/@rel	@rel	stylesheet
	/html/head/link[3]/@href	@href	http://www.state.gov/css/print.css
	/html/head/link[3]/@type	@type	text/css
	/html/head/link[3]/@media	@media	print
	/html/body	body	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" ...
//*[@id='wrapper']	/html/body/div	div	U.S. Department of StateSkip NavigationSecretary ClintonRema...
//*[@id='wrapper']	/html/body/div/@id	@id	wrapper
//*[@id='masthead']	/html/body/div/div	div	U.S. Department of StateSkip NavigationSecretary ClintonRema...
//*[@id='masthead']	/html/body/div/div/@id	@id	masthead
//*[@id='logo-assets']	/html/body/div/div/div	div	U.S. Department of State
//*[@id='logo-assets']	/html/body/div/div/div/@id	@id	logo-assets
//*[@id='logo-assets']/p	/html/body/div/div/div/p	p	
//*[@id='logo-assets']//p	/html/body/div/div/div/p/a	a	
//*[@id='logo-assets']//p	/html/body/div/div/div/p/a/@href	@href	#
//*[@id='logo-assets']//p	/html/body/div/div/div/p/a/@title	@title	U.S. Department of State - Great Seal

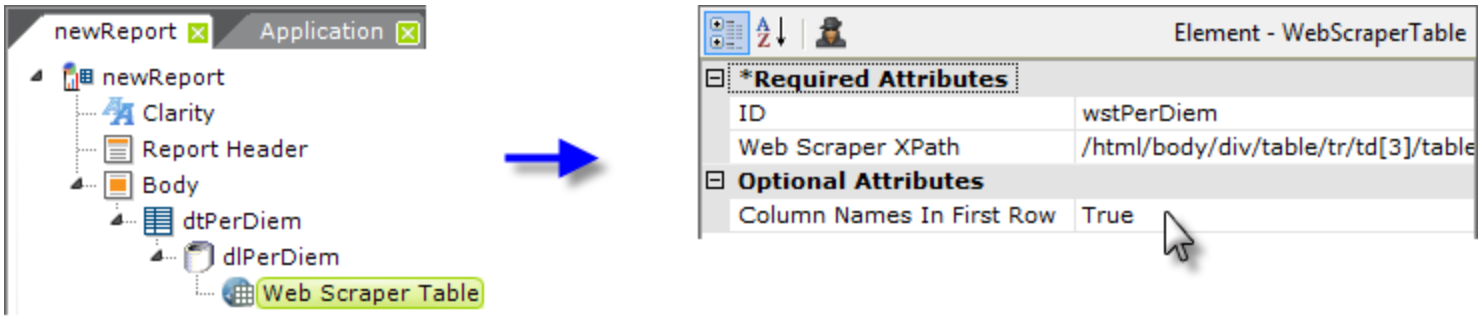
Preview the report in Studio, and examine the XML nodes reference document, shown above.

XPath by ID	XPath by Tag Name	Name	Value
//*[@id='wrapper']/table	/html/body/div/table	table	function Visible(parentb,parenta) { ID89b.style.display...
//*[@id='wrapper']//table	/html/body/div/table/tr/td/table	table	Office of Allowances Organization Cha...
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[5]/td/div/table	table	Organization Chart
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[8]/td/div/table	table	Foreign Per Diem Rates
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[8]/td/div/table[2]	table	Other Per Diem Rates
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[11]/td/div/table	table	Allowance Rates (Sec. 920)
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[11]/td/div/table[2]	table	Allowances By Location
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[11]/td/div/table[3]	table	Allowances By Type
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[11]/td/div/table[4]	table	Biweekly Updates
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[14]/td/div/table	table	DSSR Table of Contents
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[14]/td/div/table[2]	table	Search the DSSR
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[17]/td/div/table	table	Contact Us
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[17]/td/div/table[2]	table	Frequently Asked Questions
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[17]/td/div/table[3]	table	Summary of Allowances
//*[@id='AutoNumber3']	/html/body/div/table/tr/td/table/tr[20]/td/div/table	table	Reports
//*[@id='wrapper']//table	/html/body/div/table/tr/td/table[2]	table	
//*[@id='AutoNumber1']	/html/body/div/table/tr/td[3]/table	table	Select by Location Select by Al...
//*[@id='AutoNumber1']//tr[2]	/html/body/div/table/tr/td[3]/table/tr[2]/td/div/form/table	table	Previous Rates: 12/01/201211/01...
//*[@id='AutoNumber1']//tr[2]	/html/body/div/table/tr/td[3]/table/tr[2]/td/div/form/table[2]	table	Country NamePost NameSeason BeginSeason EndMaximum...
//*[@id='AutoNumber1']//tr[2]	/html/body/div/table/tr/td[3]/table/tr[2]/td/div/table	table	
	/html/body/table	table	Updates | Frequent Questions...

Locate the desired nodes by **filtering** out everything else, in this case everything*except* tables. To do this, go back to viewing the definition and enter `//table`into the **Get XPaths** attribute and then preview the definition again. Look through the Values column: tables are identified by their *headings*, not their row data, so the target table has to have distinctive headings. The table we want is shown above highlighted.

//*[@id='AutoNumber1']	/html/body/div/table/tr/td[3]/table	table	Select by Location Select by Al...
//*[@id='AutoNumber1']//tr[2]	/html/body/div/table/tr/td[3]/table/tr[2]/td/div/form/table	table	Previous Rates: 12/01/201211/01...
//*[@id='AutoNumber1']//tr[2]	/html/body/div/table/tr/td[3]/table/tr[2]/td/div/form/table[2]	table	Country NamePost NameSeason BeginSeason EndMaximum...
//*[@id='AutoNumber1']//tr[2]	/html/body/div/table/tr/td[3]/table/tr[2]/td/div/table	table	
	/html/body/table	table	Updates | Frequent Questions...

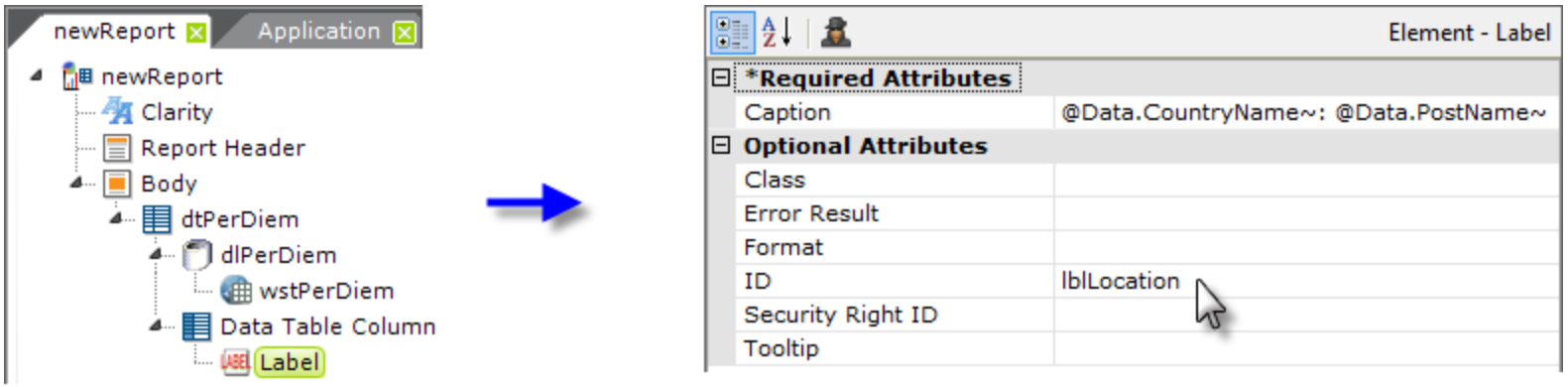
Next find, select, and copy the table's entire**XPath by Tag Name** value, highlighted above.



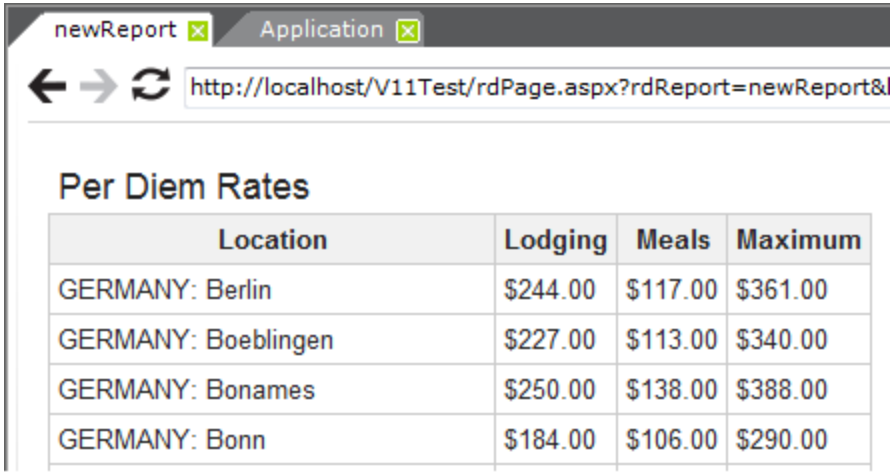
Next, back in the definition, clear the datalayer's **Get XPath** attribute value. Then add a **Web Scraper Table** child element beneath the datalayer and paster the XML tag copied from the reference document into its **Web Scraper XPath** attribute, as shown above, right. The complete value is:

```
/html/body/div/table/tr/td[3]/table/tr[2]/td/div/form/table[2]
```

And also set the **Column Names In First Row** attribute value to *True*.



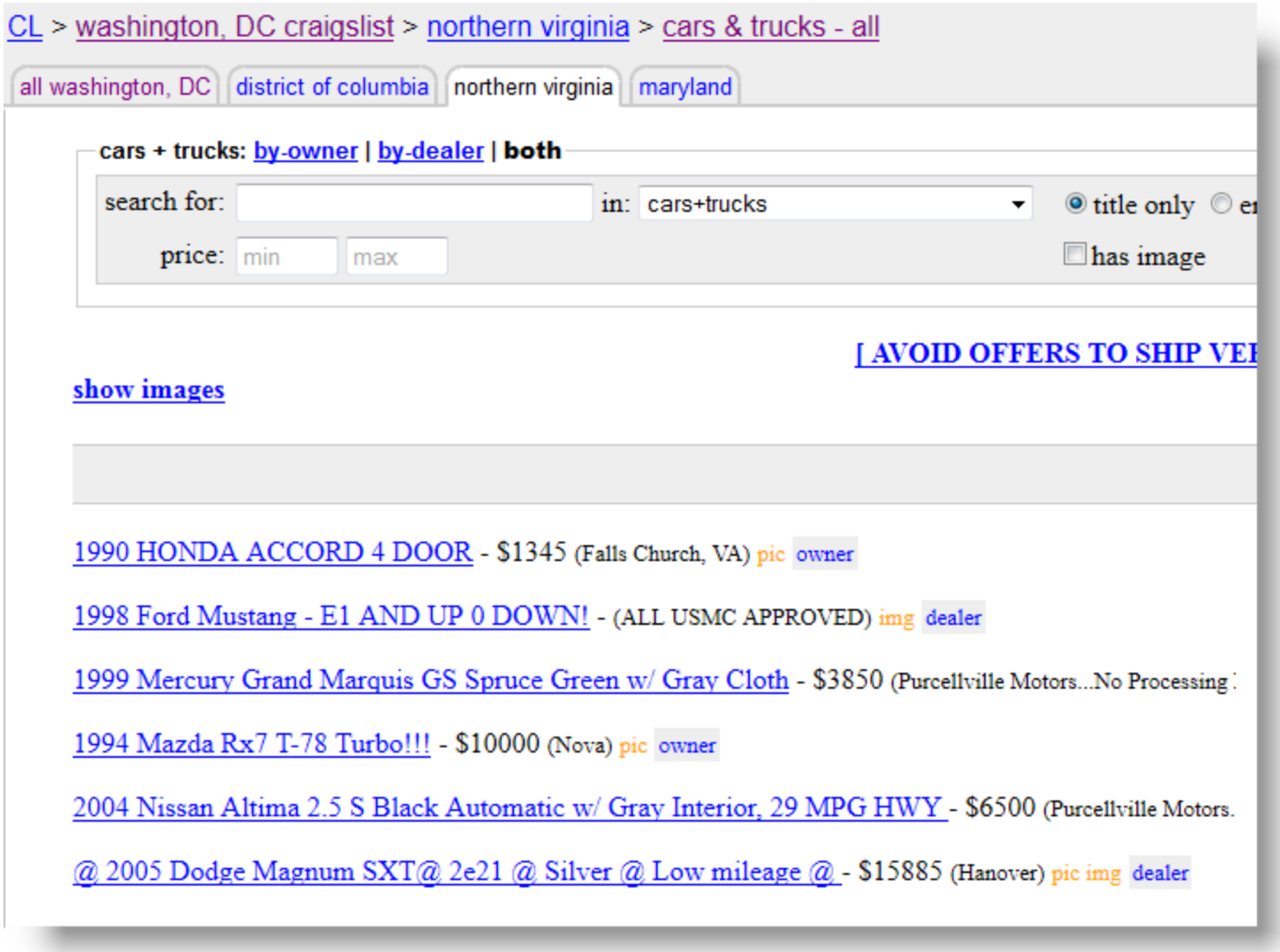
Finally, as shown above, add **Data Table Column** and **Label** elements to display the data:The **column names** in the datalayer, for use with @Data tokens, are the names seen in the XPath reference document values, with any embedded spaces removed. In this example, that's CountryName, PostName, SeasonBegin, SeasonEnd, etc.



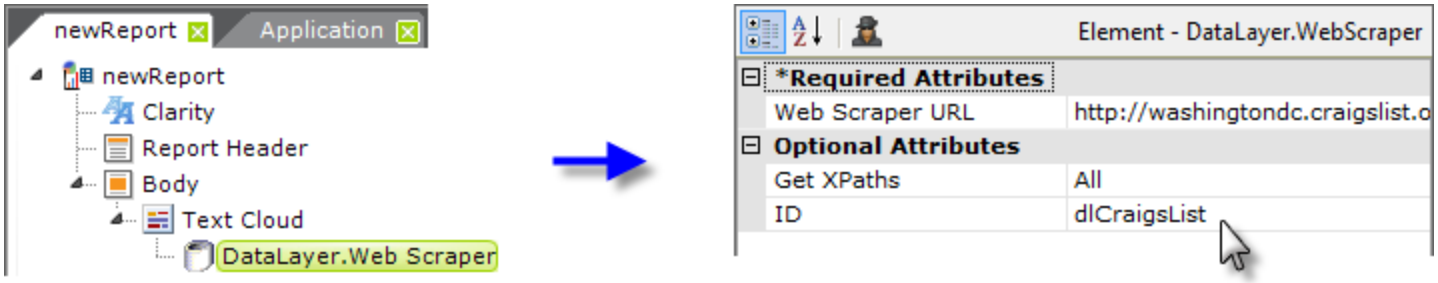
When previewed, as shown above, the report display the table with the desired data from the target web site.

Using Web Scraper Rows and Web Scraper Column

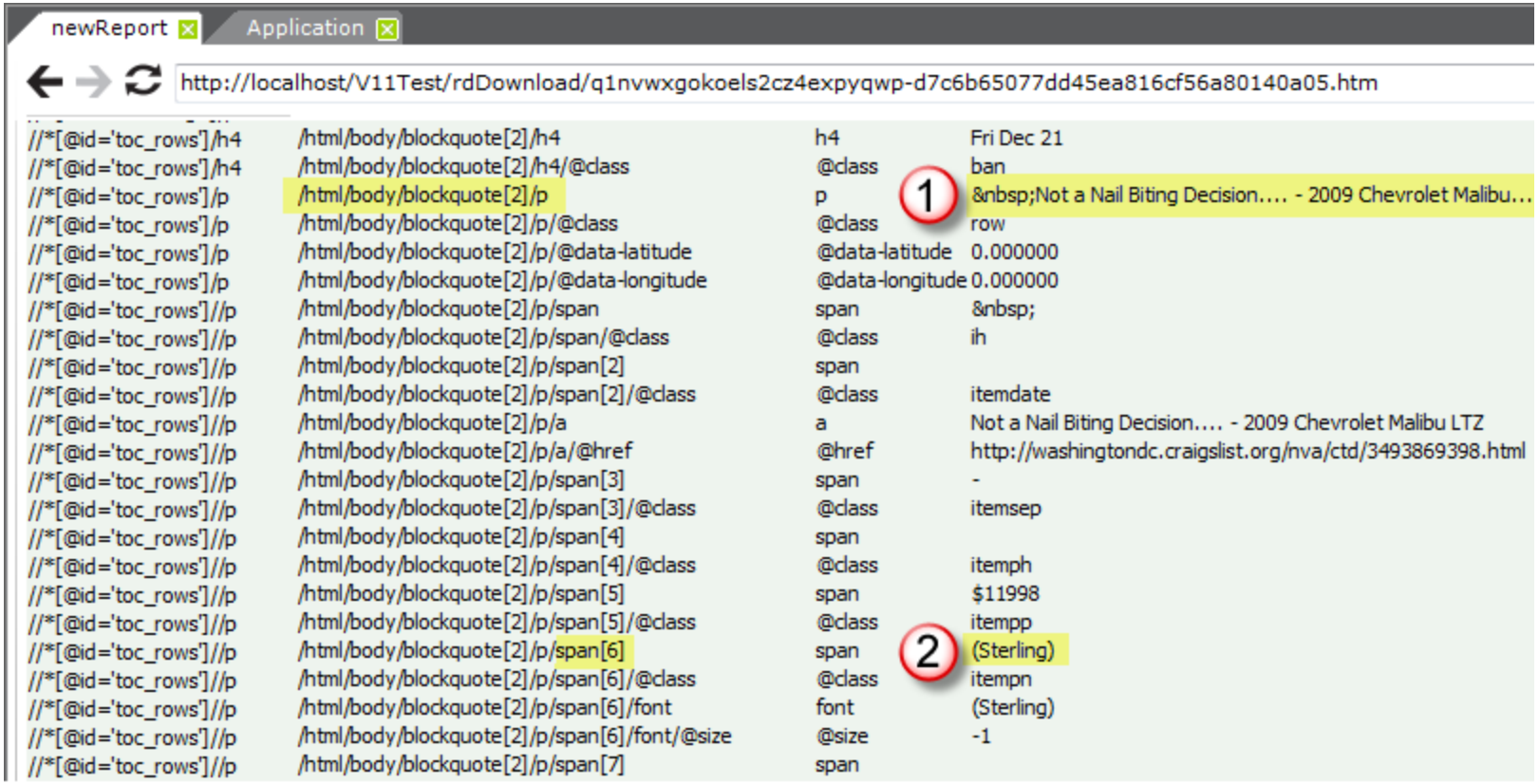
What if the target data on a web page is *not* in a table? In that case, two other elements, **Web Scraper Rows** and **Web Scraper Column**, can be used with DataLayer.Web Scraper to extract the data.


CL > washington, DC craigslist > northern virginia > cars & trucks - all
all washington, DC
district of columbia
northern virginia
maryland
cars + trucks: by-owner | by-dealer | both
search for:
in: cars+trucks
title only
price: min max
has image
[AVOID OFFERS TO SHIP VE
show images
1990 HONDA ACCORD 4 DOOR - $1345 (Falls Church, VA) pic owner
1998 Ford Mustang - E1 AND UP 0 DOWN! - (ALL USMC APPROVED) img dealer
1999 Mercury Grand Marquis GS Spruce Green w/ Gray Cloth - $3850 (Purcellville Motors...No Processing
1994 Mazda Rx7 T-78 Turbo!!! - $10000 (Nova) pic owner
2004 Nissan Altima 2.5 S Black Automatic w/ Gray Interior, 29 MPG HWY - $6500 (Purcellville Motors.
@ 2005 Dodge Magnum SXT@ 2e21 @ Silver @ Low mileage @ - $15885 (Hanover) pic img dealer

Part of the **Craig's List** web site, where cars are advertised for sale, is shown above. Our goal in this example is to display data from this site, in a **Text Cloud** chart, indicating the most common locations specified in these ads.



As shown above, first add a **Text Cloud** element, and beneath it, a **DataLayer.Web Scraper** element to the report definition. The Text Cloud element has three required attributes: an element **ID**, and a **Cloud Label Column** and **Cloud Size Column**. Give it an ID and, for now, enter two dummy values ("a" and "b", for example) in the two Column attributes. Set the datalayer's attributes as shown above. The complete target URL is: `http://washingtondc.craigslist.org/nva/cta/`



//*[@id='toc_rows']/h4	/html/body/blockquote[2]/h4	h4	Fri Dec 21
//*[@id='toc_rows']/h4	/html/body/blockquote[2]/h4/@class	@class	ban
//*[@id='toc_rows']/p	/html/body/blockquote[2]/p	p	Not a Nail Biting Decision.... - 2009 Chevrolet Malibu...
//*[@id='toc_rows']/p	/html/body/blockquote[2]/p/@class	@class	row
//*[@id='toc_rows']/p	/html/body/blockquote[2]/p/@data-latitude	@data-latitude	0.000000
//*[@id='toc_rows']/p	/html/body/blockquote[2]/p/@data-longitude	@data-longitude	0.000000
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span	span	
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span/@class	@class	ih
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[2]	span	
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[2]/@class	@class	itemdate
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/a	a	Not a Nail Biting Decision.... - 2009 Chevrolet Malibu LTZ
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/a/@href	@href	http://washingtondc.craigslist.org/nva/ctd/3493869398.html
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[3]	span	-
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[3]/@class	@class	itemsep
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[4]	span	
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[4]/@class	@class	itemph
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[5]	span	$11998
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[5]/@class	@class	itempp
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[6]	span	(Sterling)
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[6]/@class	@class	itempn
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[6]/font	font	(Sterling)
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[6]/font/@size	@size	-1
//*[@id='toc_rows']//p	/html/body/blockquote[2]/p/span[7]	span	

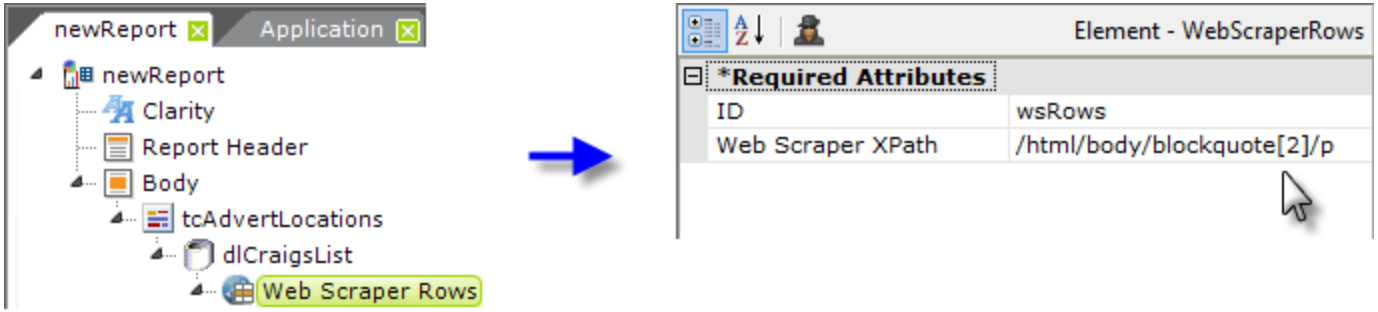
As before, preview the site and the XML nodes reference document, shown above, will be displayed. Scroll down through it, carefully looking for the start of the first actual car advertisement data block (this may not match the first ad you saw when you browsed the site). You'll have to scroll through all the stuff in the page header, search tools, title, etc.

When the first occurrence of the desired ad data block (1) is located, copy its XPath by Tag Name value (highlighted in the second column). This will value be used in the **Web Scraper Rows** element.

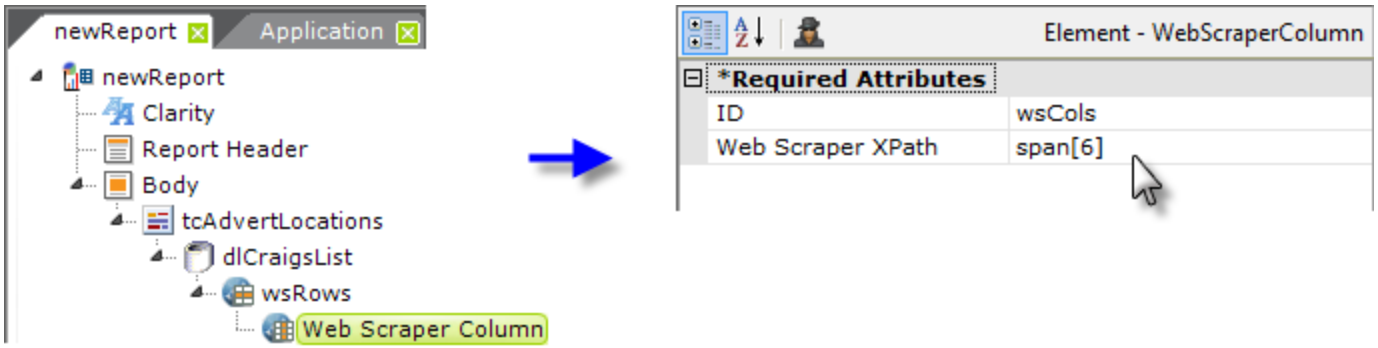
Next, because we're building a Text Cloud based on the locations mentioned in the ads, look downward in the Value column until you find the first occurrence of the location data (2). Note the part of its XPath by Tag Name value that is *relative* to the first XPath value we found:

```
1st value: /html/body/blockquote[2]/p
2nd value: /html/body/blockquote[2]/p/span[6]
Relative value: span[6]
```

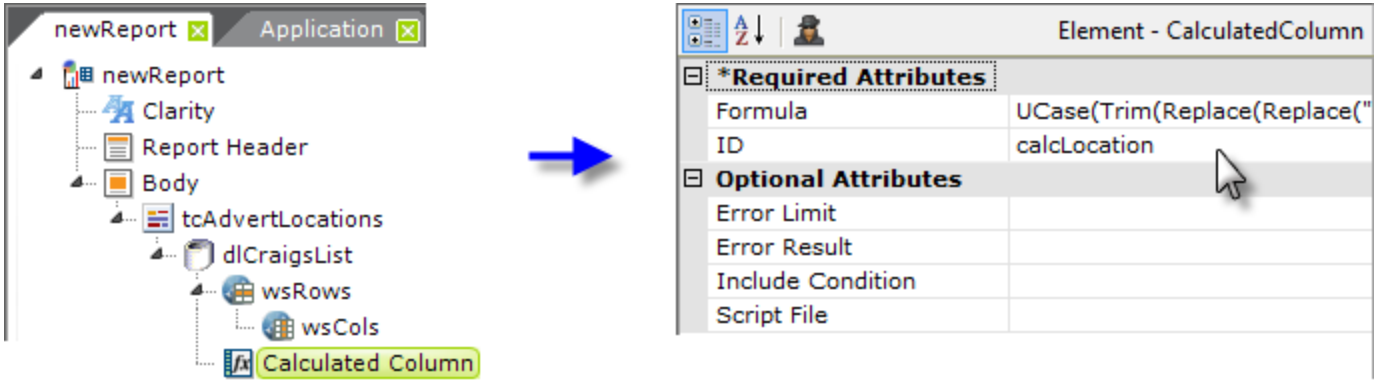
This relative value will be used in the **Web Scraper Column** element.



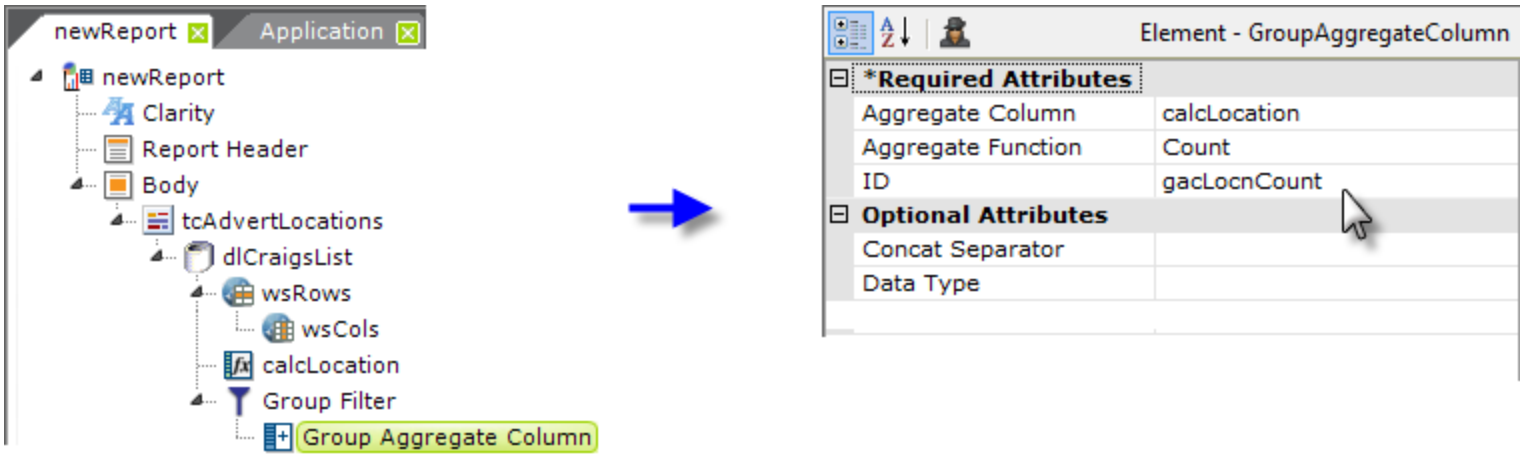
Back in the definition, clear the datalayer's Get XPaths attribute, and add a **Web Scraper Rows** element beneath the datalayer. Its **Web Scraper XPath** attribute is set to the first XPath value located earlier, as shown above.



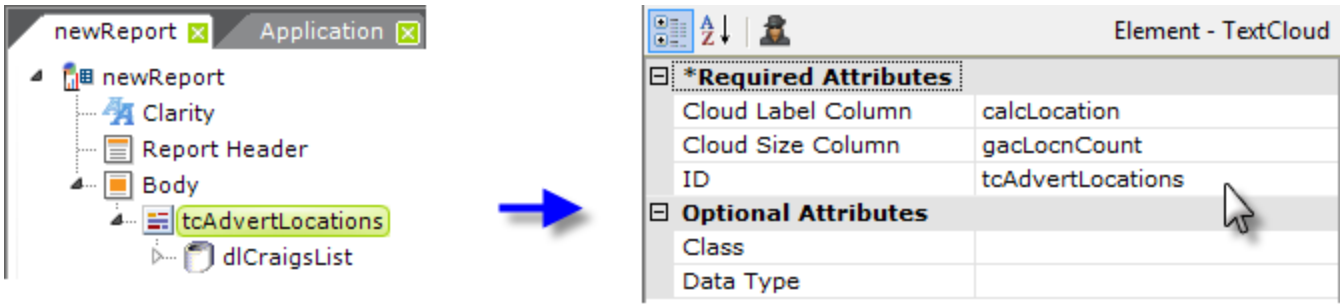
Beneath the Web Scraper Rows element, add a **Web Scraper Column** element. One of these is needed for every column to be returned in the datalayer. As shown above, its **Web Scraper XPath** attribute is set to the *relative* value noted earlier. This column in the datalayer will hold the location values.



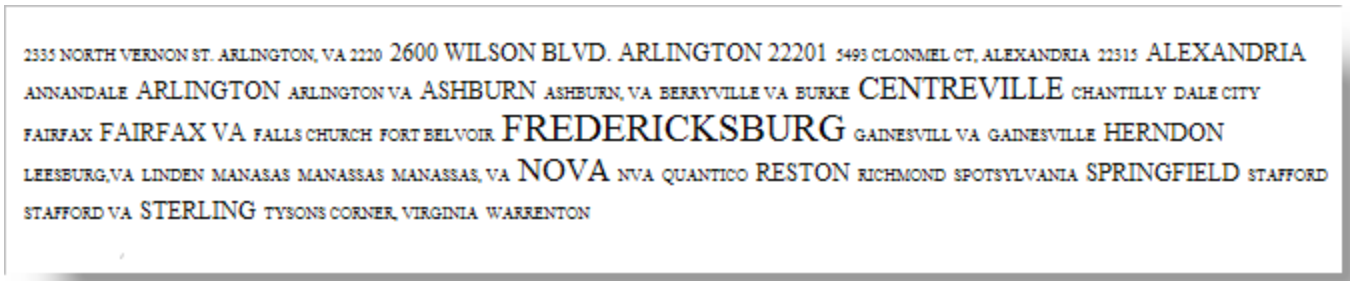
Next, add a **Calculated Column** element to clean up the data, adjust its case, etc. to bring some consistency to the data,



and **grouping elements** to group it by location and produce a count or frequency value we can use in the Text Cloud.



Finally, return to the Text Cloud element's **Column** attributes and set them to the proper datalayer columns, as shown above.



When browsed, the final report produces the text cloud shown above, with larger fonts denoting the locations from which cars are offered for sale on Craig's List more frequently.

Once this report has been defined, it will retrieve the latest data from the web page each time it is run. If the web site is structurally redesigned, however, then the report may have to be updated with new XPath values.

DataLayer.Web Service

The **DataLayer.Web Service** element gives developers the ability to retrieve information from a SOAP-style web service and can handle values returned as either an XML *dataset object* or a *string*. The Logi Server Engine generates an XML rowset from the returned data and developers can use tokens to retrieve records from each column.

The following topics introduce the DataLayer.Web Service element:

- DataLayer.Web Service Attributes
- Working with DataLayer.Web Service
- Using Studio's DataLayer Wizard
- Multi-Step Web Service Access

We also offer "DataLayer.REST" on page 111 for use with web services that use a RESTful protocol.

DataLayer.Web Service - Attributes

The DataLayer.Web Service element has the following attributes:

Attribute	Description
Connection ID	(Required) The ID of a **Connection.Web Service** element defined in the _Settings definition. If this value is left blank, the datalayer will use the *first* connection in the _Settings definition. For clarity, developers are advised to enter an ID here in all cases.
ID	An optional unique element ID. Recommended for easier identification when debugging.
XPath	Specifies a standard XPath string that will be used to select a set of matching nodes. All of the matching nodes are then used to generate the resulting datalayer. If an XPath is defined, it will be processed *before* any user defined XSL file. Tokens may be used in this attribute value.

The **Connection.Web Service** element sets the parameters required to link the application to a web service; user account credentials are specified in its attributes.

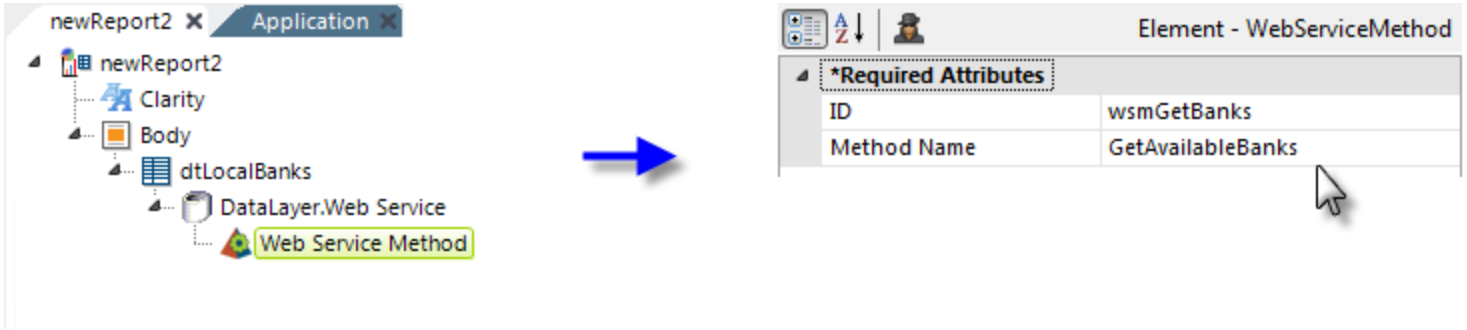
 If you're trying to communicate with a web service that requires the **TLS 1.1** or 1.2 protocol, you will need to use Logi Info v12.2-SP4 or later (earlier versions only support TLS 1.0). In addition, Info Java applications must use Oracle JDK 1.8 or OpenJDK 8 to make the protocol work.

DataLayer.Web Service - Working with DataLayer.Web Service

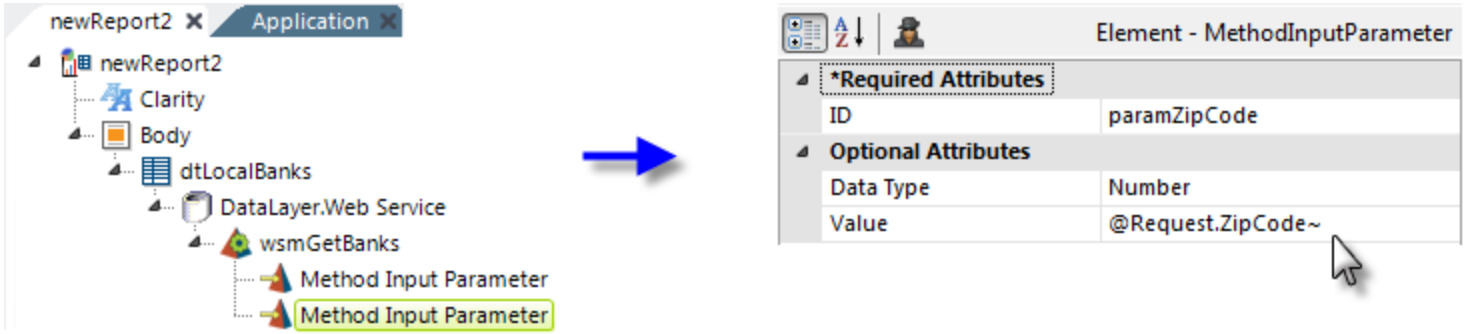
A web service can expose one or more of its **methods** so that they can be called by developers from an application. Web service methods return XML as either a *dataset object* or serialized *string* type. In both cases, the XML data may require special formatting for use within Logi reporting products. Logi Studio includes an **XslTransform** element which can be quite useful for converting XML in a datalayer into a format the Logi engine understands.

The information needed to configure the values of the Logi elements used with a web service can be found in the web service's **WSDL** document.

The following example illustrates how to use the **DataLayer.Web Service** element and its child elements:



1. Add a **DataLayer.Web Service** element to the definition, as shown above, and configure its attributes per the attribute table in "DataLayer.Web Service - Attributes" on the previous page.
2. Add a **Web Service Method** element beneath the datalayer, and give it a unique ID.
3. Set its **Method Name** attribute value to the name of the web service method that is to be called.

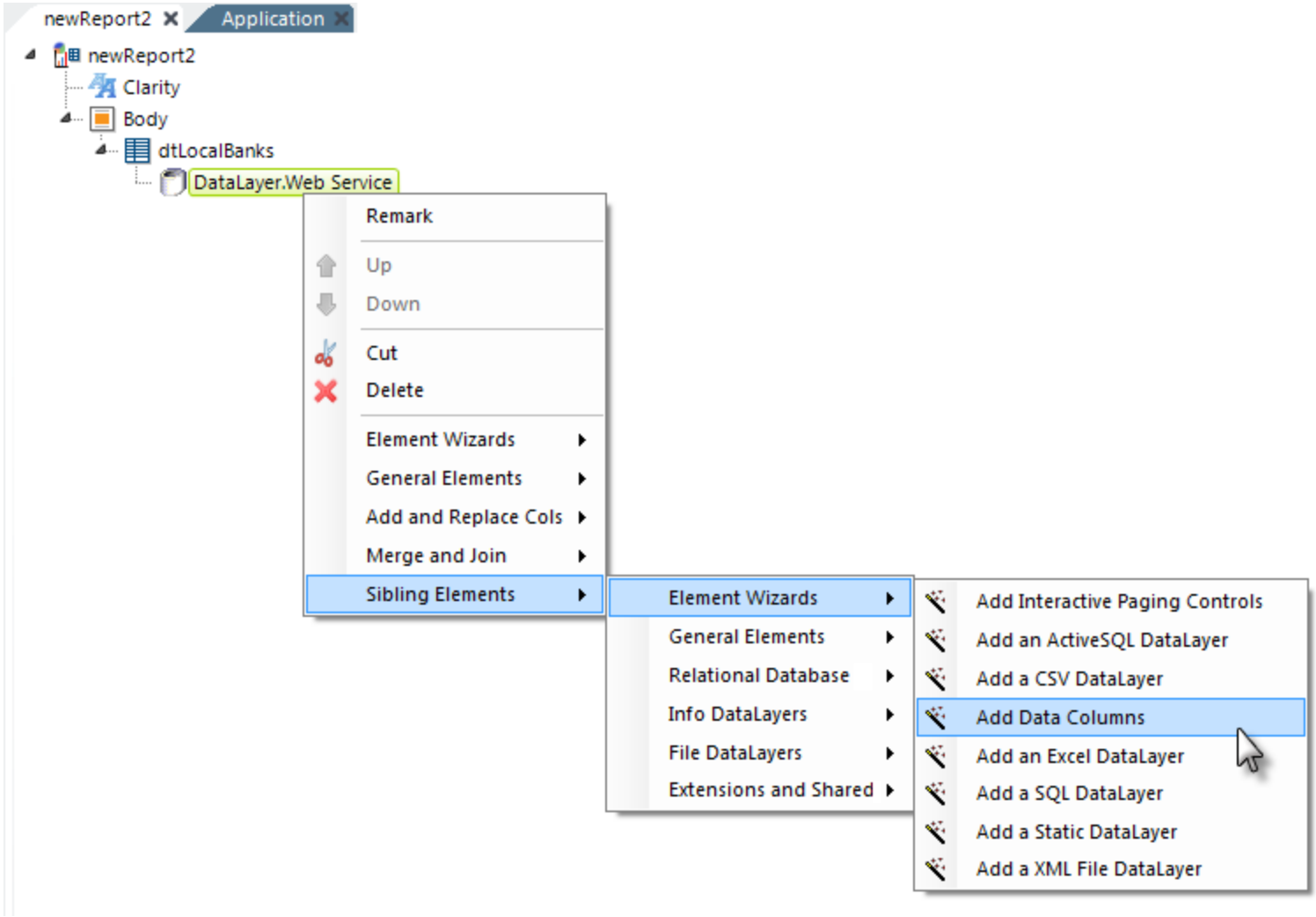


4. As shown above, add one or more **Method Input Parameter** elements beneath the Web Service Method element, based on the number of parameters you must pass to the web service method.

   If you're developing a **.NET** Logi application, these elements *must* be in the **order** that the parameters are expected by the web service.
   If you're developing a **Java** Logi application, the element ID attribute *must* be set to the **name** of the corresponding parameter.
   Consult the WSDL document to confirm correct parameter order and names.

5. Set each parameter's **Data Type** attribute value to the correct data type.
6. Set each parameter's **Value** attribute to the value (hard-coded or as a token) that you want to pass to the method.

Now when you run your report, the data from the web service should be retrieved into the datalayer.



When working with web services and Data Tables, you may not be sure what the column names of the returned data will be; this is a perfect time to use the **Add data columns** wizard found in Studio. Right-click the datalayer element, then select the options shown above in the pop-up menus. The wizard will interrogate the web service and insert the elements needed to display your Data Table columns.

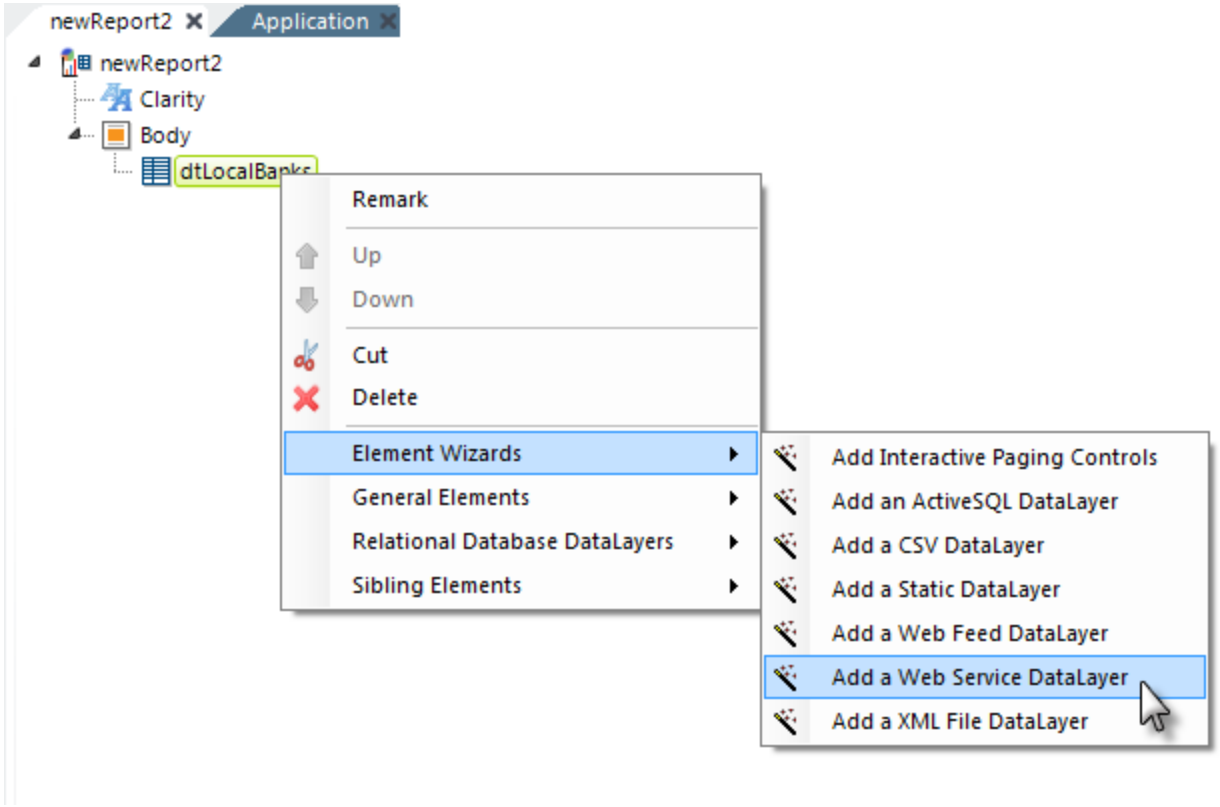
Working with Complex Data Types

Some web services may require you to send data, such as a user name and password, to them using complex data types. These requirements should be handled by writing a plug-in, as our DataLayer.Web Service element is not capable of creating complex structures for transmission to a web service.

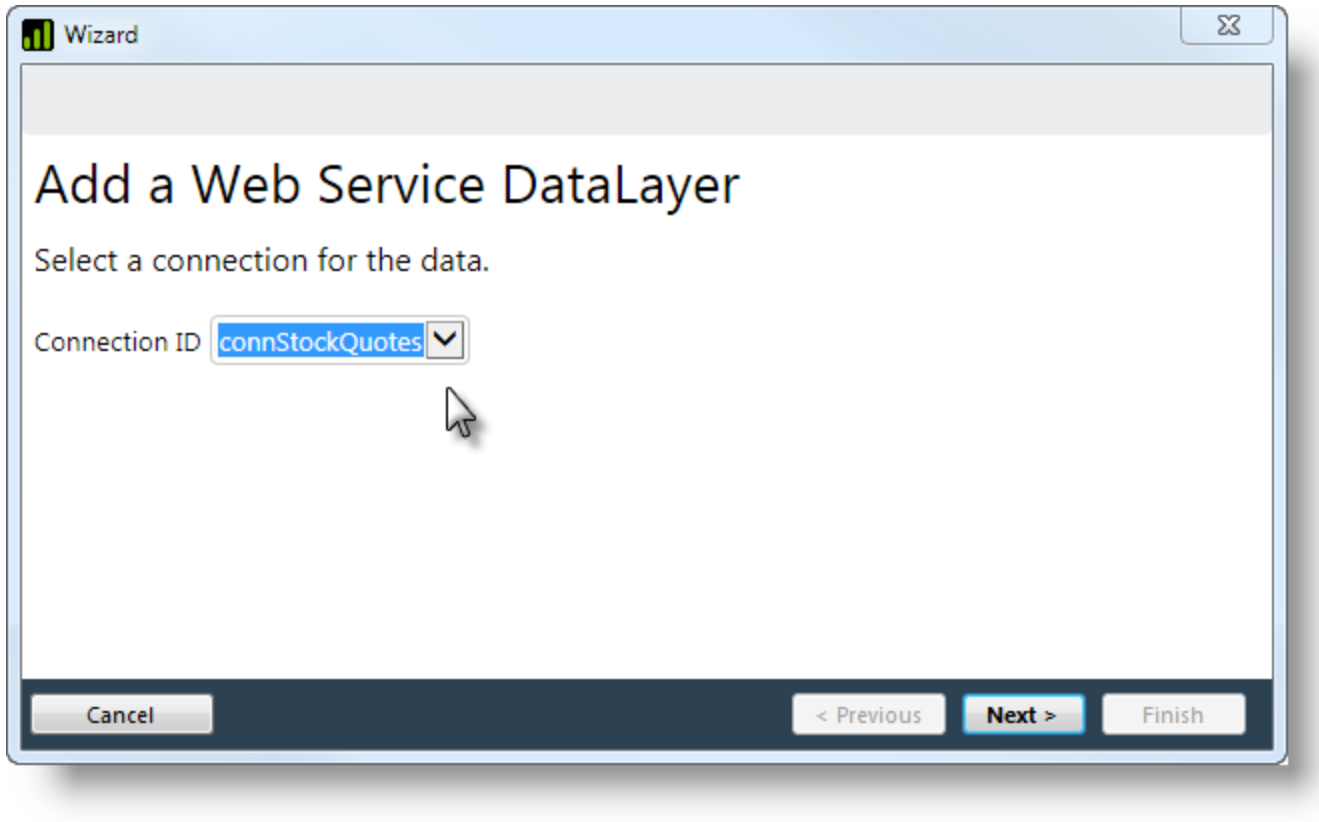
You can see an example of this type of plug-in in action in our Web Service with Complex Header sample plug-in.

DataLayer.Web Service - Using Studio's DataLayer Wizard

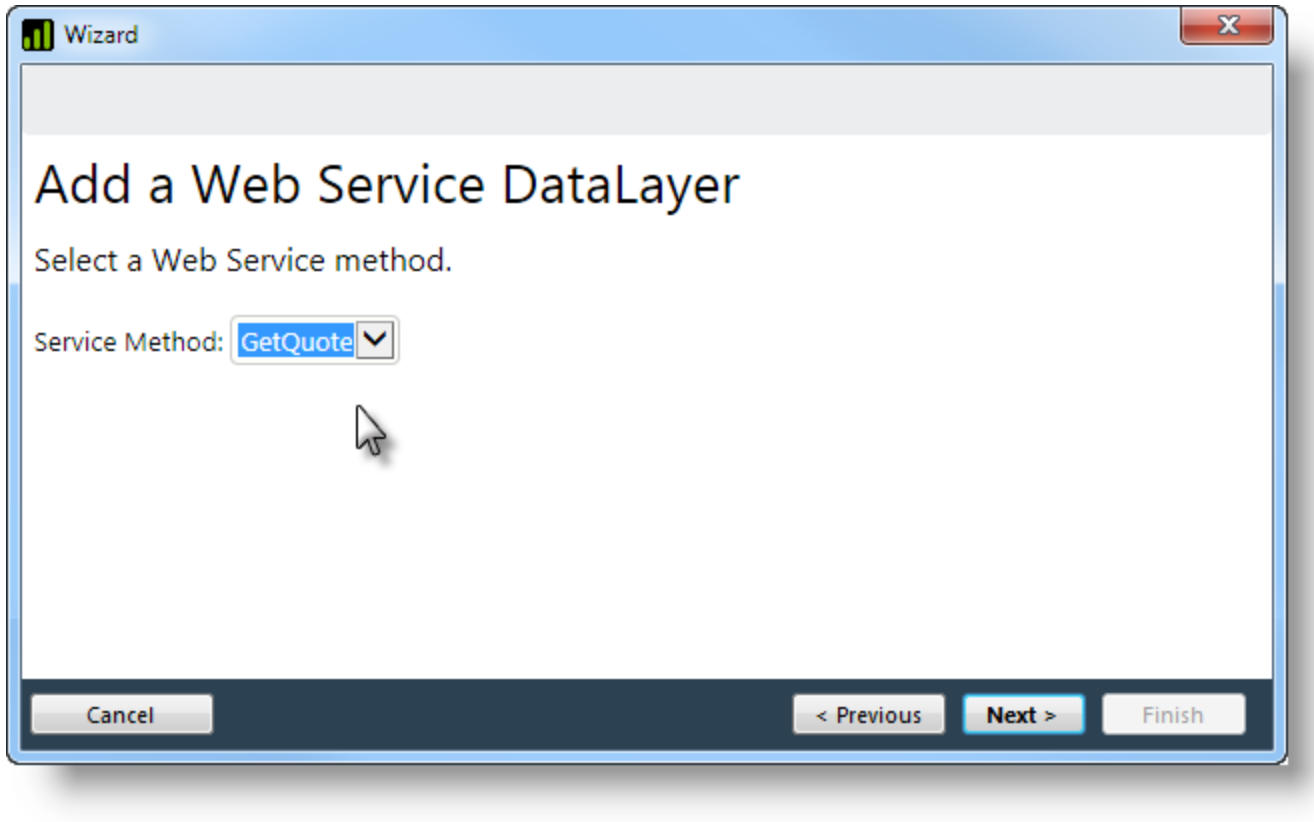
Logi Studio includes a wizard that can assist you in configuring DataLayer.Web Service. The wizard assumes that you have already added an appropriate database **Connection** element in the _Settings definition and configured it.



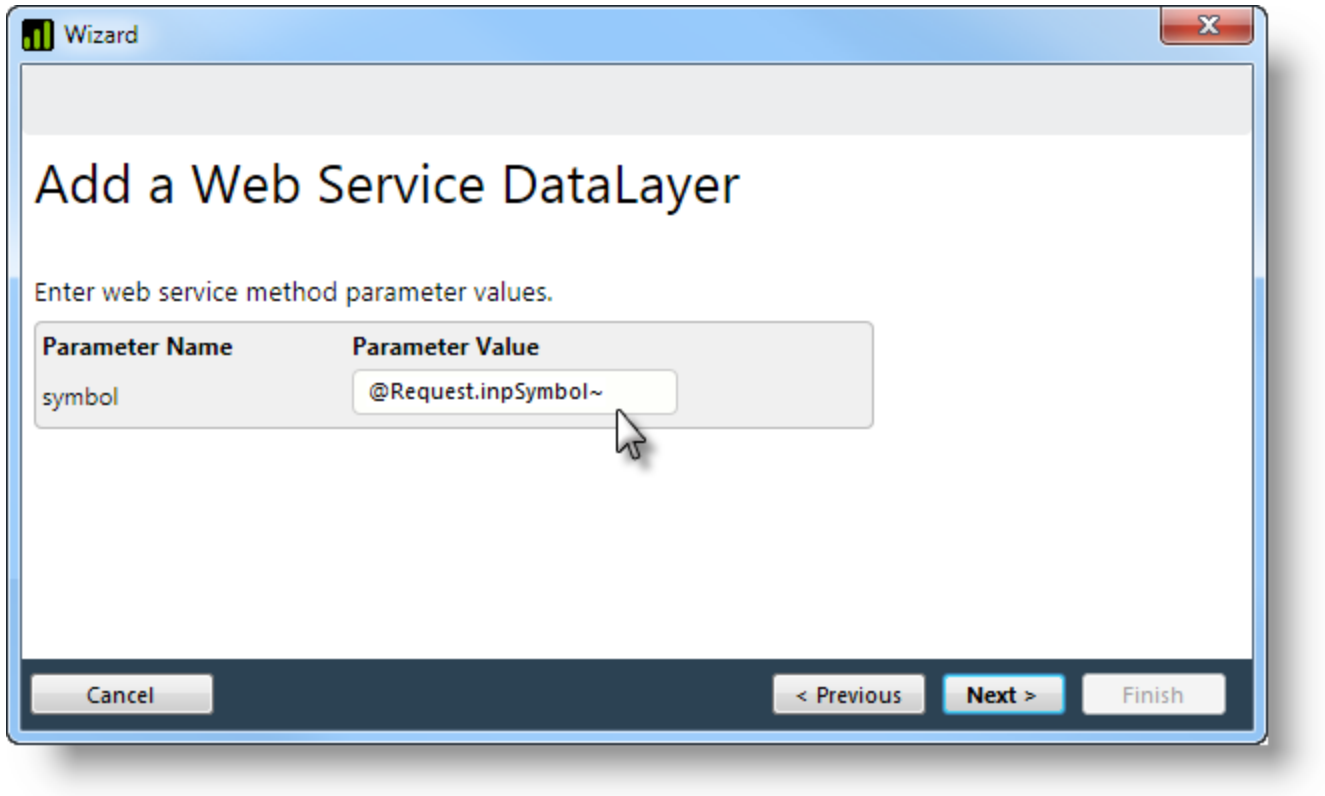
As shown above, the wizard can be started by selecting and then right-clicking the parent element under which you want to add the datalayer, and using the context menus to select "Add a Web Service DataLayer". The wizard will open; use it as follows:



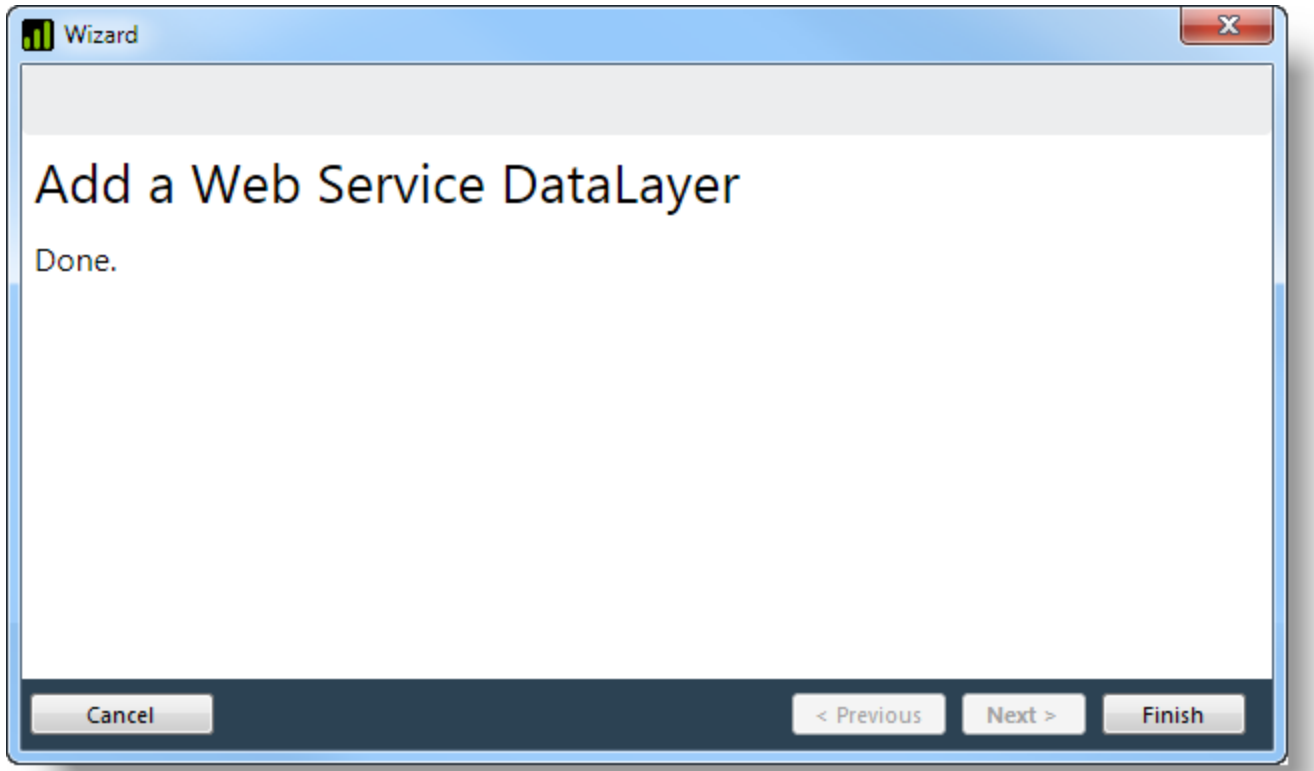
1. Select the ID of the Connection element from the drop-down list of available connections. Click **Next** to continue.



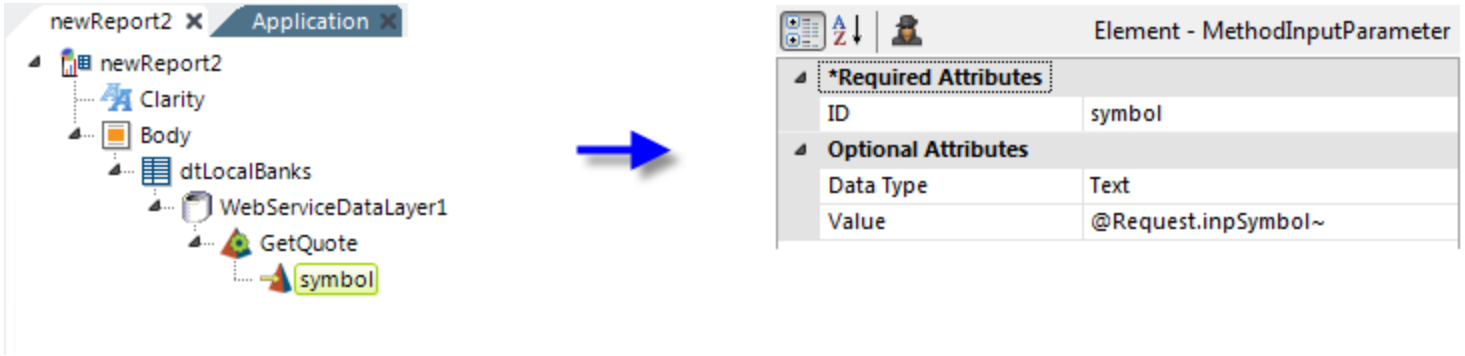
2. The wizard will query the web service to determine its available web service methods. Select the desired method from the drop-down list. If an error occurs here, then the wizard was not able to connect to the web service and you should examine your Connection element attributes to ensure that they are correct. Click **Next** to continue.



3. If the web method service requires parameters, the box shown above will be displayed, with each parameter listed. Enter a literal value or a token, as necessary, for each parameter and click **Next**.



4. Click **Finish** to close the wizard.



5. The wizard has inserted the datalayer and configured its attributes, as shown above, and has inserted and configured any required **Web Service Method** and **Method Input Parameter** elements. You will need to manually provide the input parameter values, either directly or using tokens.

DataLayer.Web Service - Multi-Step Web Service Access

Some web services require **multiple** interaction steps before they will return a result. For example, you may first need to tell the web service your security credentials, then tell it what data category you need, then tell it what format to put the results in, and then request the actual data. A single **DataLayer.Web Service** element is not designed for this kind of multi-step "conversation" with a web service. So how can this be accomplished?

One approach is to use *multiple* DataLayer.Web Service elements, one for each part of the conversation. In your report, just add multiple DataLayer.Web Service elements, one for each part of the conversation, and they will execute one after another.

In the example shown above, multiple DataLayer.Web Service elements are used, with each calling a different web service method, to complete the multi-step conversation required by the web service. The last one sends the query and the data is retrieved and available using standard @Data tokens.

Another approach is to write a **plug-in** to handle the multi-step conversation. When used with"DataLayer.Plugin" on page 294, the plug-in would issue the sequential web service method calls and place the resulting data into the datalayer for use in the report. DevNet members, after login, can see an example of this type of plug-in in action in our Web Service with Complex Header sample plug-in.

For more information, our Web Service Sample Application demonstrates the use of DataLayer.Web Service.

DataLayer.XML

The **DataLayer.XML** element gives developers the ability to retrieve data from an **XML data file** or REST-style web service.

The following topics introduce the DataLayer.XML element:

- DataLayer.XML Attributes
- Working with DataLayer.XML File
- Using DataLayer.XML with Input Elements
- Using DataLayer.XMLwith REST Web Services
- Using Studio's DataLayer Wizard

About DataLayer.XML

This element was renamed; formerly it was known asDataLayer.XML *File.*

The element retrieves data from a static XML file stored locally, or remotely (using a URL), or from a REST-style web service. Static XML files are very useful in Logi applications as small data sets of Input control options or configuration data that is local but external to the application and easily modified.

 If you're trying to communicate with a web service that requires the **TLS 1.1** or 1.2 protocol, you will need to use Logi Info v12.2-SP4 or later (earlier versions only support TLS 1.0). In addition, Info Java applications must use Oracle JDK or OpenJDK 8+ to make the protocol work.

Logi Info also includes "DataLayer.REST" on page 111 and "DataLayer.JSON" on page 98 for use with REST-style web services and "DataLayer.Web Service" on page 204 for use with SOAP-style web services.

DataLayer.XML - Attributes

The **DataLayer.XML** element has the following attributes:

Attribute	Description
XML File / URL	(Required) Specifies the identifier of the XML data file. You may enter either a file system location or the URL for a location on the web. By default, the server will look in the _SupportFiles folder and, if the file is located there, then only the file name and extension are needed in this attribute. If it's located elsewhere in the file system, a fully-qualified path and file name is required, such as: `D:\Projects\Accounting\Data\MyData.xml`If a URL is used to retrieve a file from a web server, it must be fully-formed, such as: `http://myserver-/mydata.xml`
Connection ID	Specifies the element ID of a Connection element defined in the _Settings definition. If a value is specified here, then XML File attribute value isappended to the Connection element's Url Host attribute value.
Http Method	Specifies the verb to be sent with a REST request. Options include: *DELETE*, *GET*, *POST*, and *PUT,* or can be a custom verb.The default value is *GET*.
ID	Specifies an optional element ID. Recommended for easier identification when debugging.
Maximum Rows	Specifies the maximum number of data rows to be retrieved.
Remove Namespace	Specifies whether the namespace and schema information that some XML data sources will add to the retrieved data is removed. The default value is *False*.

Attribute	Description
XPath	Specifies a standard XPath string that will be used to select a set of matching nodes. All of the matching nodes are then used to generate the resulting datalayer. Tokens may be used in this attribute value.

DataLayer.XML - Working with DataLayer.XML

Data in an XML file is required to be in a specific format for use within Logi reporting products: data rows must be child elements of the root element, and data columns must be attributes of the data rows. For example, here's a typical XML data file:

```
<CarInfo>
<Car Brand="Alfa Romeo" Model="174 Hatchback" Code="1"/>
<Car Brand="Aston Martin" Model="Spider Open" Code="5"/>
<Car Brand="Audi" Model="Spider Open" Code="6"/>
<Car Brand="Bentley" Model="Spider Open" Code="7"/>
<Car Brand="BMW" Model="3 Series Convertible" Code="8"/>
<Car Brand="BMW" Model="3 Series Coupe" Code="9"/>
 <Car Brand="BMW" Model="5 Series Saloon" Code="10"/>
<Car Brand="Cadillac" Model="Spider Open" Code="14"/>
</CarInfo>
```

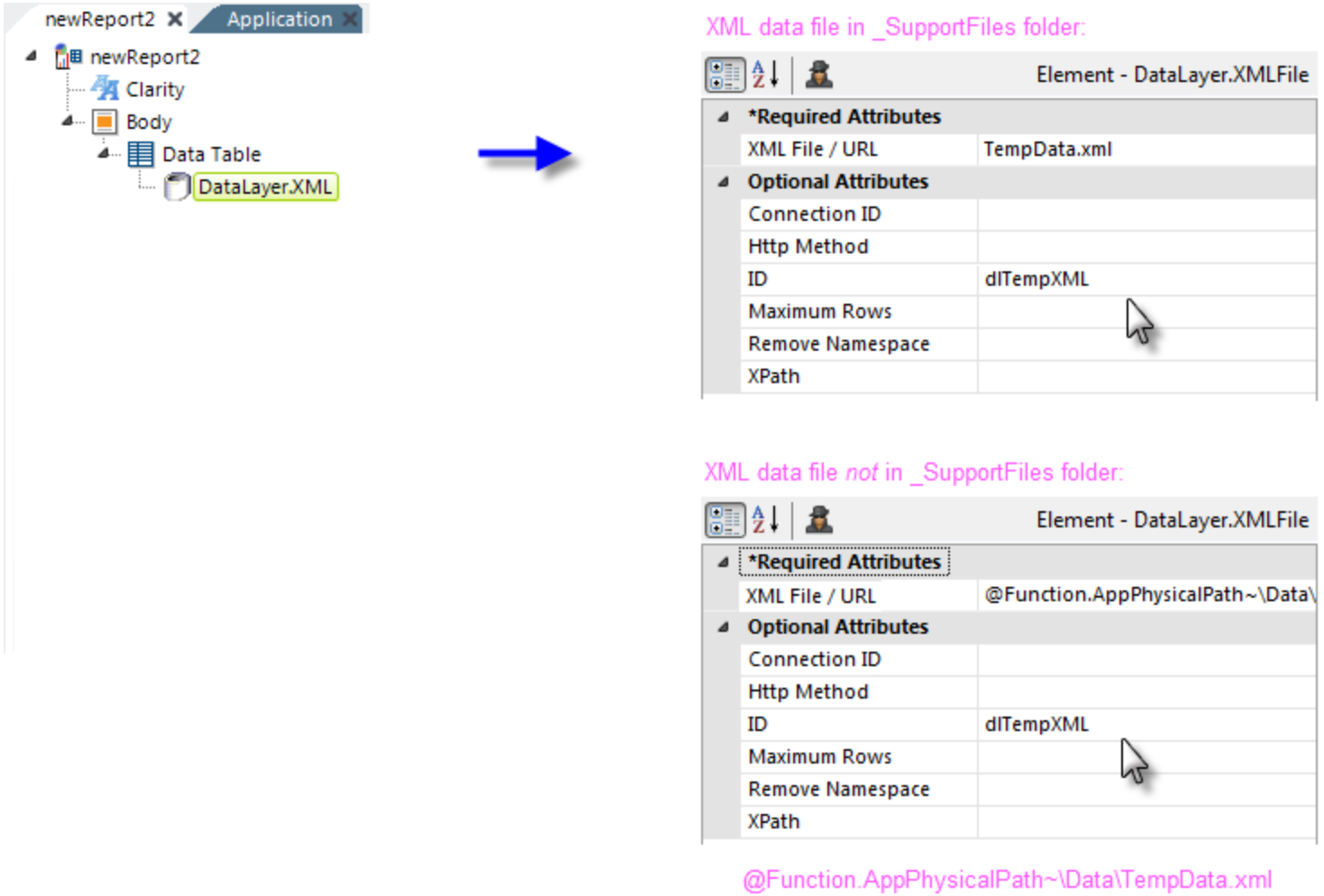
The example above illustrates the required formatting. When received into the datalayer, the data from this file will consist of eight data rows, each with Brand, Model, and Code columns.

The datalayer may be used with one or more child **XslTransform** elements, which use XSL files to transform the XML data *before* it's loaded into the datalayer. This allows the adjustment of XML data if it has a schema that cannot be understood by the Logi Server engine. For example, if the input XML comprises a data set with *two* tables, a transform can remove a table, since the datalayer expects only one.

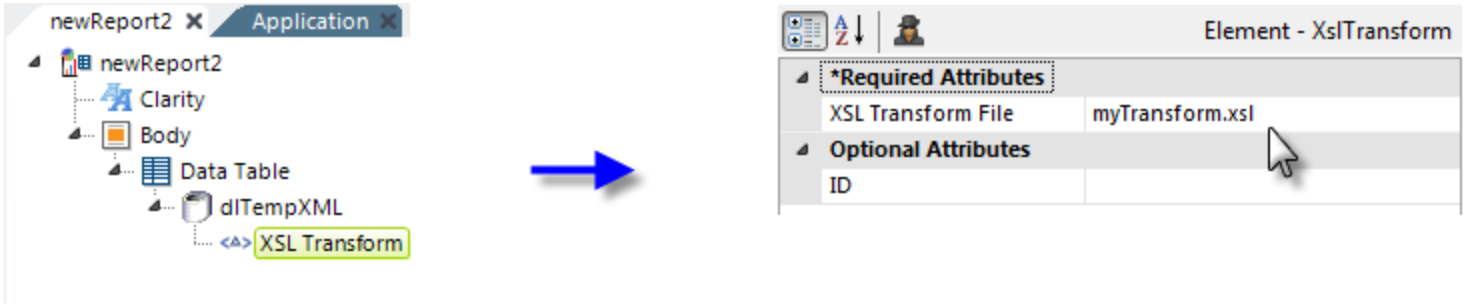
Data may be returned in a hierarchical format, containing multiple levels of parent-child relationships. This structure can create difficulties for further processing and analysis. *The Flattener* child element can be used to process hierarchical data into the standard datalayer tabular set of rows and columns.

XML data files make great temporary repositories for data retrieved from databases, possibly saving the overhead of another database query. For example, it may be faster or more efficient to *export* data retrieved from a database in one part of a Logi application out to a temporary XML file, then later, in another part of the application, *read it back in* with DataLayer.XML.

The following example shows DataLayer.XML in use:



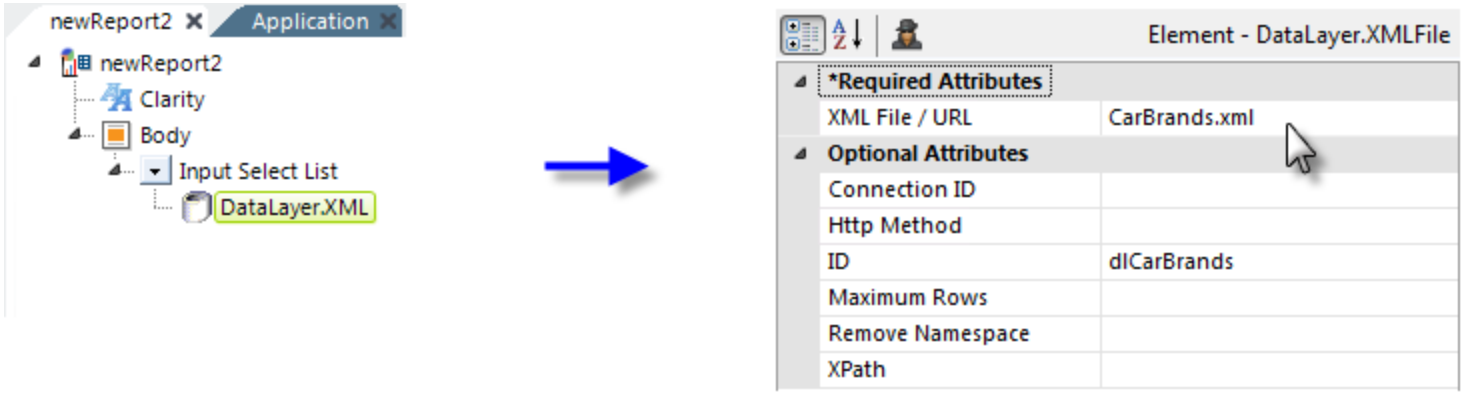
As shown above, a **DataLayer.XML** element has been added beneath a Data Table. Its **XML File/URL** attribute has been set to a value that can vary depending on the location of the actual file. There is **no query** or **search** feature; everything in the file will be pulled into the datalayer (where it can then be sorted, filtered, etc. using appropriate elements).



As shown above, one or more **XslTransform** elements can be added beneath the datalayer for each XSL transform to be applied to the XML data *before* it is retrieved into the datalayer. The same rules apply for the XSL file location and name as for the XML file (i.e. in the example above, the XSL file is in the _SupportFiles folder).

DataLayer.XML - Using DataLayer.XML with Input Elements

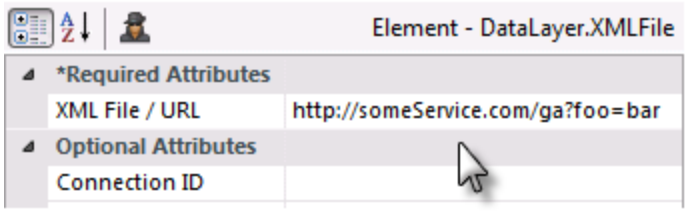
One convenient use of XML data files is as a source of options for menus and list-type Input elements. An XML data file can be created quickly and is easy to edit and, unlike static data hard-coded into the application, an XML data file can be edited externally. Here's an example of a common use of XML data and DataLayer.XML for this purpose:



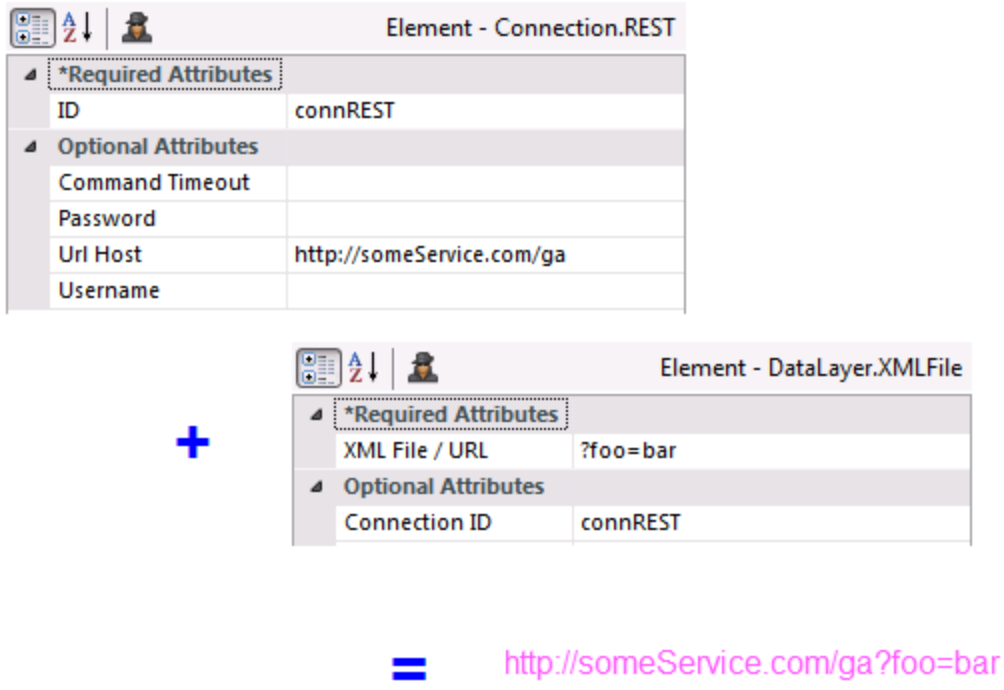
A **DataLayer.XML** has been added beneath an **Input Select List** element and its attributes set to use the sample XML file, shown in "DataLayer.XML - Working with DataLayer.XML" on page 222. In the Input Select List's attributes, "Brand" is identified as the **Caption Column** and "Code" as the **Value Column**.

DataLayer.XML - Using DataLayer.XML with REST Web Services

DataLayer.XML is able to retrieve data from REST-style web services and the simplest approach is to configure its **XML File/URL** attribute with a complete URL. If the web service requires nothing else, the data will be retrieved using the default HTTP GET method.



For more flexibility, you can instead use a **Connection.REST** element to define the firstpart of the URL:



Using a value in the datalayer's **Connection ID** attribute causes its XML File/URL value to be appended to the Connection element's **URL Host** attribute value, as shown above. This can be useful if your application uses multiple datalayers to connect to the same web service. If you prefer, you can leave the XML File/URL value blank and add a **Link Parameters** element beneath the datalayer. Its name/value pairs will then be appended to the Connection ID's URL Host value, preceded by a "?".

Multi-Step Processing

You may encounter a security scenario where a two-step process is required, such as when a web service expects a "login" API call with credentials (as opposed to supplying credentials in a Connection.REST element) and returns an access ID, which must then be part of any other API calls. You can use this approach to handle this scenario:

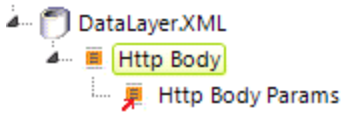
1. Use Local Data with DataLayer.XML to authenticate and retrieve the web service's access ID. If the results are returned as JSON data, you'll need to use a Flattener element to make them available in tokens.
2. Use the resulting @Local token for the access ID with other DataLayer.XML elements to retrieve the desired data.

Some web services may require you to login and get an access ID, using a login URL, and then use that access ID in the Request Header of any subsequent queries, which go to to a different URL. In that case, you can use this approach:

1. Add *two* Connection.REST elements in your _Settings definition, one for the API login and one for API queries, each with their appropriate URL.
2. Add a Request Header child element beneath the "query" connection element.
3. Use Local Data with DataLayer.XML and the "login" connection element to authenticate and retrieve the web service's access ID. If the results are returned as JSON data, you'll need to use a Flattener element to make them available in tokens.
4. Then use Set Session Variables to assign the resulting @Local token for the access ID to a session variable, for example "AccessID".
5. Use the @Session.AccessID~ token to configure the "query" connection element's child Request Header element.
6. All other DataLayer.XML elements that make API calls will use the "query" connection element.

Using Different HTTP Methods and Request Body Data

If the web service requires you to use a different HTTP method, such as *POST* instead of *GET*, you can select it from the option list in the datalayer's **Http Method** attribute. Some web services may want you to use a custom method that's not in the option list, like *PATCH*, and in that case you can just type it right into the attribute value.



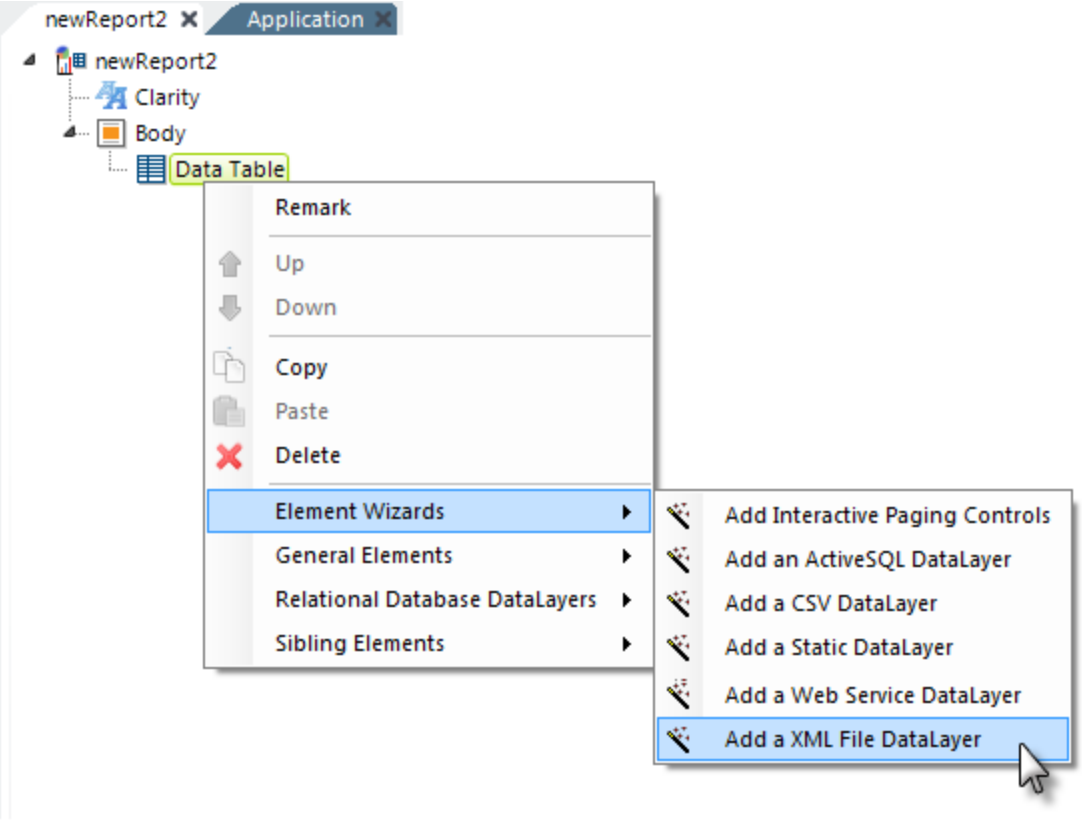
The **Http Body** element, shown above, has two attributes:

- **Content Type,** which sets the content type in the request header and defaults to *application/xml* for this datalayer.
- **Http Body Content**, which is the request body data, entered as an XML or JSON document or as a URL-encoded set of name/value pairs.

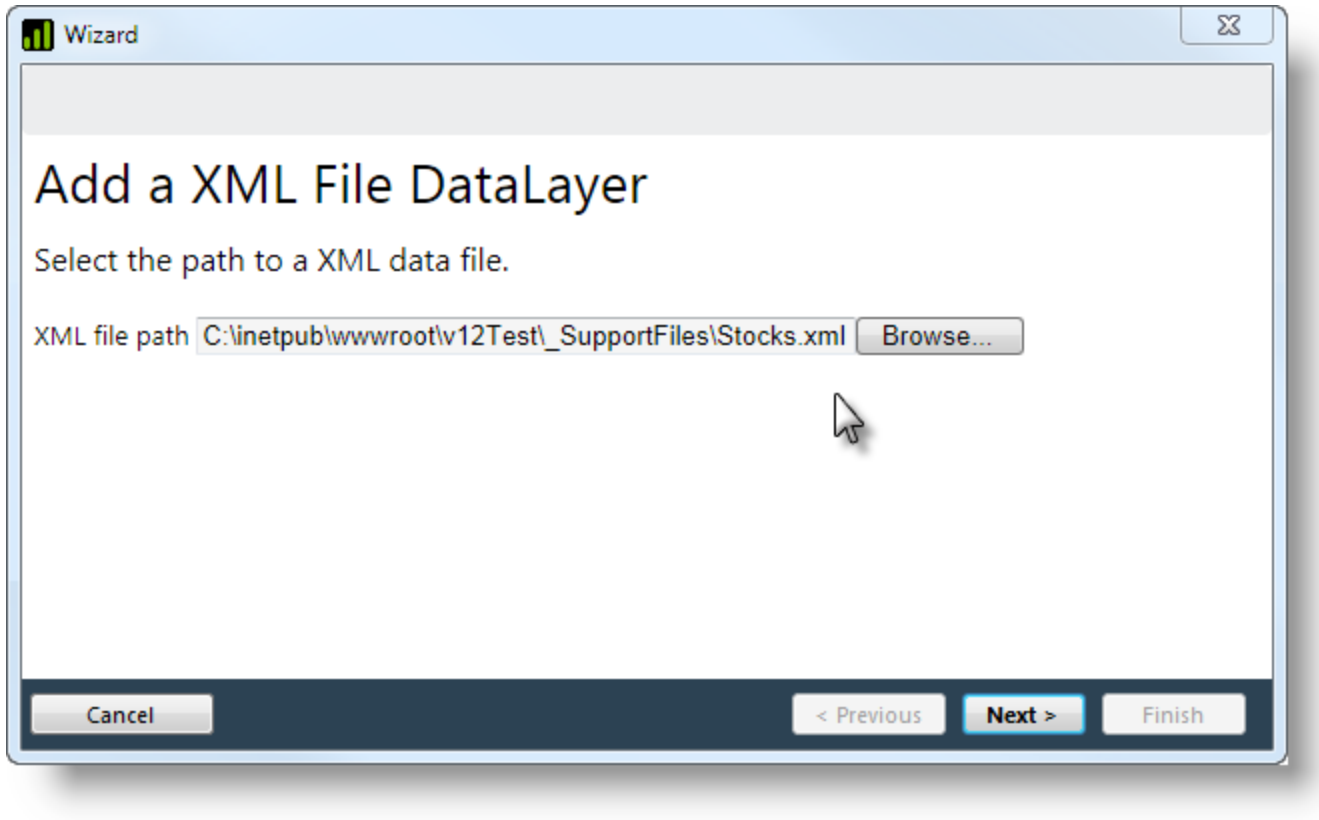
If you want to encode name/value pairs in the request body data to avoid possible issues with invalid characters, set the Http Body element's Content Type attribute to *application/x-www-form-urlencoded* and use the **Http Body Params** element to define the pairs. Tokens may be used in Http Body Param values.

DataLayer.XML - Using Studio's DataLayer Wizard

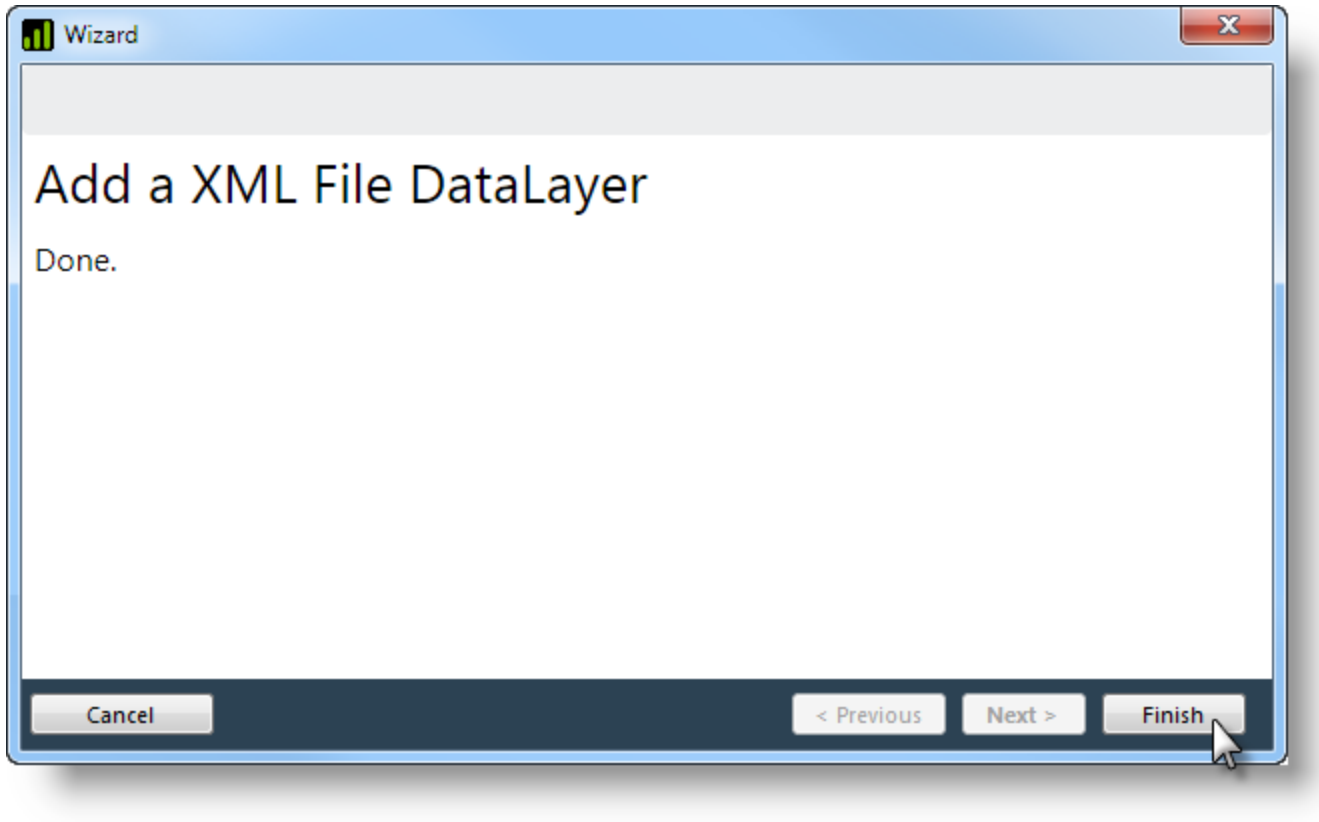
Logi Studio includes a wizard that can assist you in configuring DataLayer.XML.



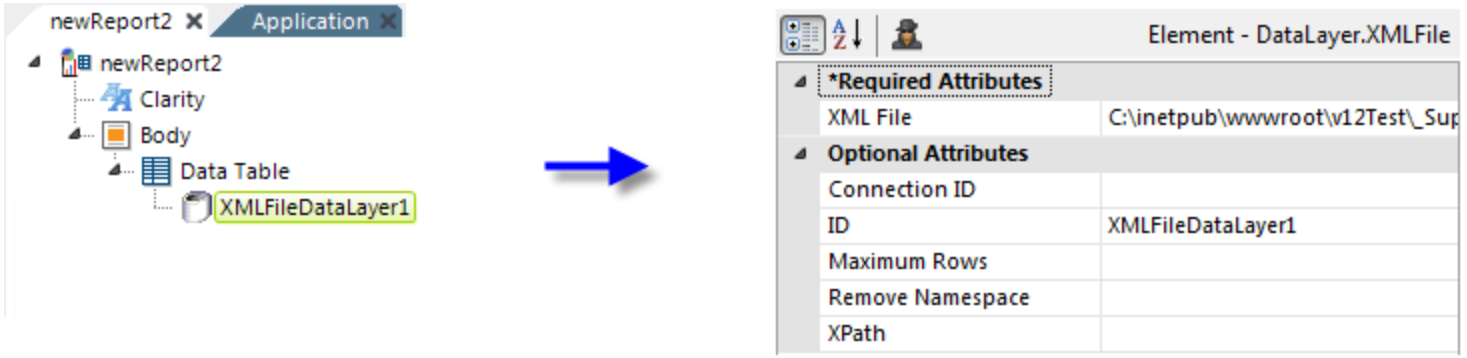
As shown above, the wizard can be started by selecting and then right-clicking the parent element under which you want to add the datalayer, and using the context menus to select "Add a XML File DataLayer". The wizard will open; use it as follows:



1. Browse to and select the filename of the XML file to be used. Click **Next** to continue.



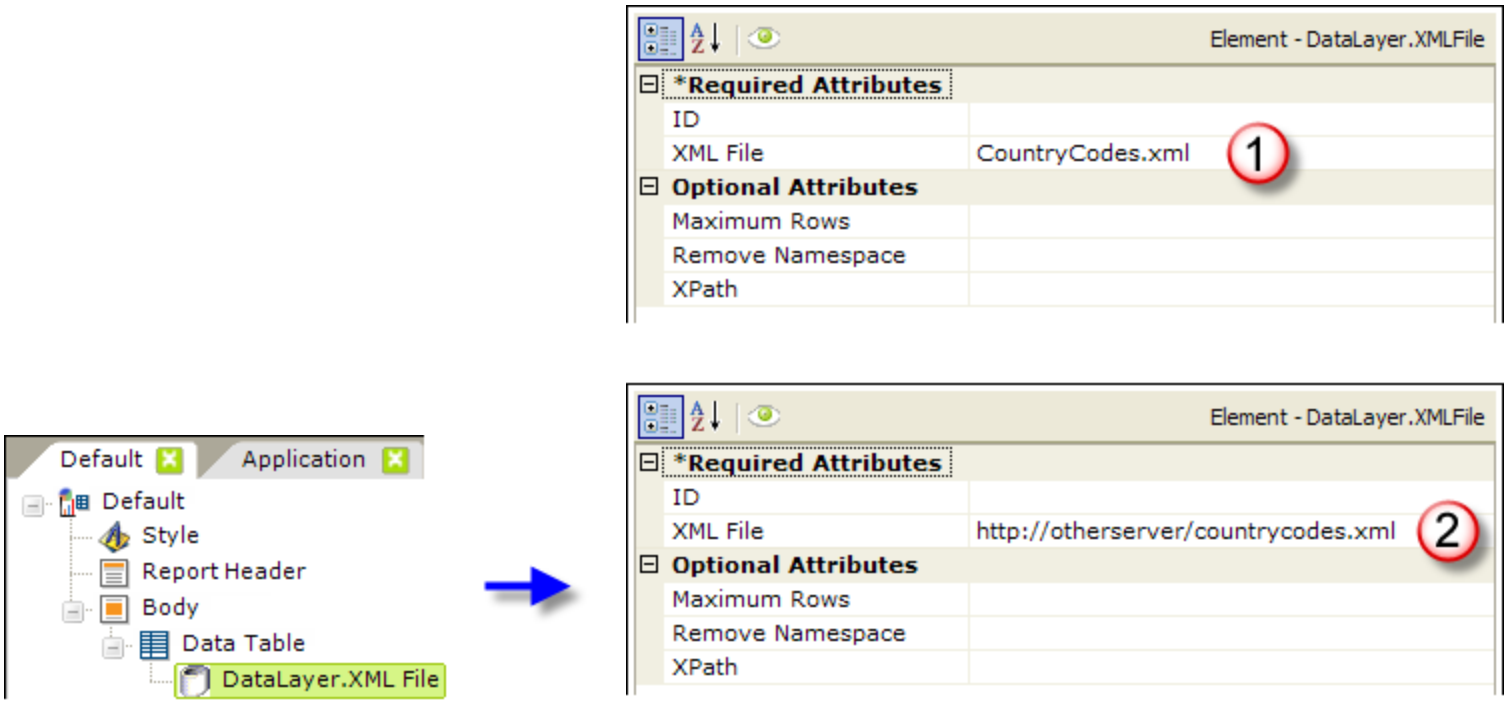
2. Click **Finish** to insert the datalayer.



3. The wizard will insert the datalayer and configure its attributes, as shown above. Other optional attributes may have to be configured manually.

DataLayer.XML Can Use URL, Too

The **DataLayer.XML File** element is very useful for getting data from an XML file into tables or input select lists. However, instead of specifying a filename, you can specify a **URL** instead and retrieve data from sources other than the local file system. Here's how it works:



Usually, when using a DataLayer.XML File element, for the **XML File** attribute you provide (1) the name of a file in the file system. If you enter just the filename, the application will look for it first in the _SupportFiles folder. However, you can instead provide (2) a URL to a file on another server. This could be useful in centralizing look-up information or getting data from another server.

DataLayer.ActiveSQL

The **DataLayer.ActiveSQL** element is a special type of datalayer designed for use with the Analysis Grid and Analysis Chart super-elements, and the Data Table.

The following topics introduce the DataLayer.ActiveSQL element:

- DataLayer.ActiveSQL Attributes
- Retrieving Data with DataLayer.ActiveSQL
- Using Studio's DataLayer Wizard

About DataLayer.ActiveSQL

The primary model for data retrieval in Logi applications involves retrieving *all* of the requested data from a datasource and caching it, either in memory or to disk, for use by elements that will display the data. This generally works well and provides good performance across a broad spectrum of use cases.

However, there are situations when a different approach is required: when SQL queries that return a large amount of data take a long time to process before the first page of data is returned and shown. In these situations, retrieval of large data sets is usually only the first step in an analysis process, because users are not likely to want to page through thousands of pages of results. Instead they're going to manipulate and reduce the data in order to identify trends or opportunities.

DataLayer.ActiveSQL is a special type of datalayer designed for just such a situation. It differs from conventional datalayers in that it does *not* retrieve all of the rows in the initial request and, in response to runtime manipulations of the interface by users, it dynamically modifies and resends its initial SQL query. This helps the parent element work with much larger data sets and still provide good performance. It also increases the number and frequency of SQL queries the database server must handle (which, generally, does not prove to be a burden on the database server) but dramatically improves performance for the user.

DataLayer.ActiveSQL understands different SQL dialects and automatically adjusts for different database servers.

Data Tables can also take advantage of DataLayer.ActiveSQL. The datalayer generates SQL queries that limit the number of rows returned, helping with paging and sorting. With Interactive Paging in use, the query requests a number of pages of records, instead of all of the data, improving performance. As the user moves beyond the first set of pages, the database is queried again to get the next set of pages.

Usage Restrictions

There are some important restrictions to be observed when using this datalayer:

- DataLayer.ActiveSQL is only available for use with the Analysis Grid, Analysis Chart, Data Table, Sharing List, and AutoComplete elements.
- DataLayer.ActiveSQL *onlyworks* with these database servers and others that use the same SQL syntax:
    - DB2
    - Microsoft SQL Server 2005+
    - MySQL (except v5.5)
    - Oracle
    - PostgreSQL
    - Progress OpenEdge
    - Redshift (Amazon)
    - Vertica (HP)

    It will also work with databases accessible through ODBC and JDBC connections, as long as the database supports the SQL syntax of one of the databases listed above. When you configure Connection.ODBC or Connection.JDBC elements for this purpose, you must specify the appropriate SQL syntax.

ActiveSQL support for 1010data sources is available.

Note, when creating queries, that database servers may support data object names that are *case-sensitive.*

- Do not include *comments* in SQL queries used with this datalayer.

- If the report definition includes interactive features allowing the user to affect the data, such as a super-element or UI elements, SQL queries used with this datalayer *should not* include **ORDER BY** clauses. Use a SQL Sort element instead for ordering the data.
- SQL queries that include a form of the **JOIN** clause should individually list all of the desired field names in the SELECT clause. Queries that don't include a JOIN clause can use the wild card asterisk (*), if desired.
- When using aggregations, the database server controls whether or not Null values will be included in the calculations. Typically, databases are configured to *exclude* Null values from aggregations, except for COUNT(*) operations.
- DataLayer.ActiveSQL will not work with Stored Procedures, nor with Common Table Expressions.

DataLayer.ActiveSQL - Attributes

The DataLayer.ActiveSQL element has the following attributes:

Attribute	Description
Source	(Required) Specifies the initial SQL query to be executed, for example SELECT * FROM *MyTable*. This query will be modified dynamically by the datalayer as its parent element's user interface is manipulated. If other elements allow the user to dynamically affect the data, queries should *not* include an **ORDER BY** clause. Use a SQL Sort element instead. Also, SQL queries that include a form of the **JOIN** clause should individually list all of the desired field names in the SELECT clause. Queries that don't include a JOIN clause can use the asterisk (*), if desired. **Tokens** can be used, for example, in WHERE clauses, to create dynamic queries. The Browse button for this attribute can be used to launch the **Query Builder** wizard.
ActiveSql Buffer Size	Specifies the maximum number of rows to buffer locally. Customizing this value can help performance when the data includes a large number of columns. Default value: *5,000 rows*
Connection ID	Specifies the ID of a **Connection.Oracle**, **Connection.MySQL**, **Connection.SqlServer** (to MS SQL Server 2005+), **Connection.PostgreSQL, Connection.DB2, Connection.Redshift, or Connection.Vertica** element defined in the _Settings definition. Other connection types *will not work*. If left blank, the top-most Connection element defined will be used by default.
Handle Quotes Inside	Specifies how to handle any single-quotes found in token values used to form part of the SQL statement in the Source attribute. When *True*, it ensures SQL syntax validity by **doubling** any single-quotes found.

Attribute	Description
Tokens	For example, imagine the SQL statement "SELECT * FROM Customers WHERE CompanyName LIKE '%@Request.Name~%' ". If the value of the Request token is "Trail's Head", the embedded single-quote could be problematic. With the HandlesQuotesInTokens attribute set to *True*, the SQL statement sent to the database server will become "SELECT * FROM Customers WHERE CompanyName LIKE '%Trail''s Head%' ". Default: *False*
ID	Specifies a unique element ID. Providing a value here is *highly recommended* for easier identification of data when debugging.
Maximum Rows	The maximum number of rows to retrieve from the data source, per request. Default: *5,000 rows* The default was changed to *200 rows* or, if pagination is being used, *10 pages*, whichever is greater.

DataLayer.ActiveSQL - Retrieving Data with DataLayer.ActiveSQL

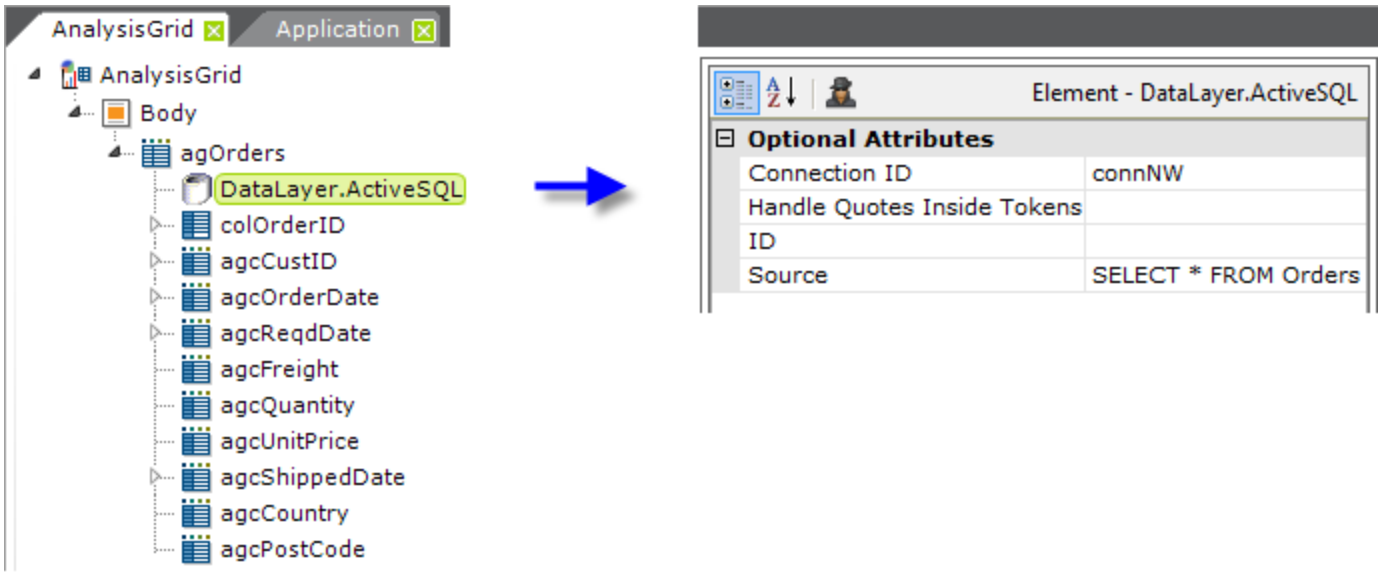
Like other datalayers, DataLayer.ActiveSQL has some special child elements (discussed below) that developers can use to shape the retrieved data. However, with the exception of **Formatted Column**, other child elements for linking datalayers and formatting or transforming the data are not available.

The data retrieved is available using @Data **tokens**, in the format @Data.*ColumnName*~. The spelling of the column name is **case-sensitive**. The data is only available within the **scope** of the **parent** element of the datalayer, not throughout the entire report definition.

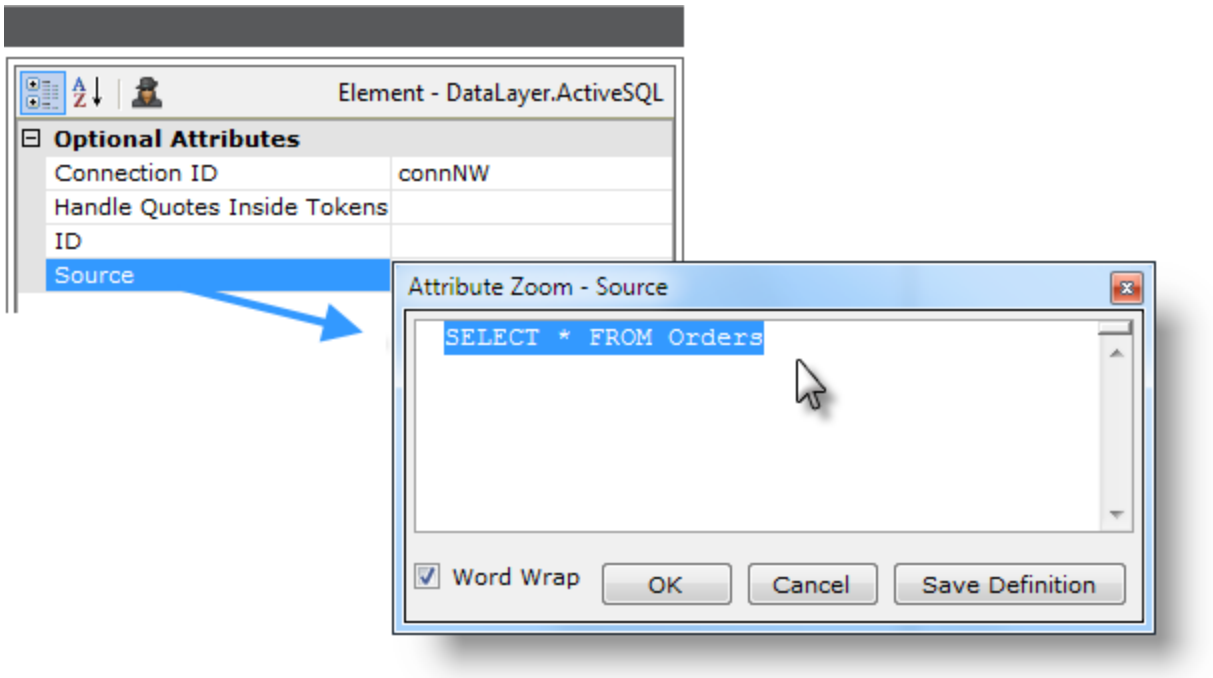
The data retrieved into the datalayer can be viewed by turning on the **Debugging Link** in your _Settings definition (General element) and using the resulting link at the bottom of your report page to view the **Application Trace** page. You'll be able to see the generated SQL query, and a link on the page will display the data it retrieved.

Using the Datalayer to Retrieve SQL Data

DataLayer.ActiveSQL runs a SQL query to retrieve data from a SQL datasource. The element's **Source** attribute value is the actual SQL statement, which must be composed using valid syntax to be run.



Use of the DataLayer.ActiveSQL element with an Analysis Grid is shown above.



For easier editing, double-click the Source attribute name and the attribute value can be entered in the **Attribute Zoom Window**, as shown above.

Special DataLayer.ActiveSQL Child Elements

As mentioned earlier, DataLayer.ActiveSQL has some unique child elements that can be used to shape the SQL query that's generated and the retrieved data.

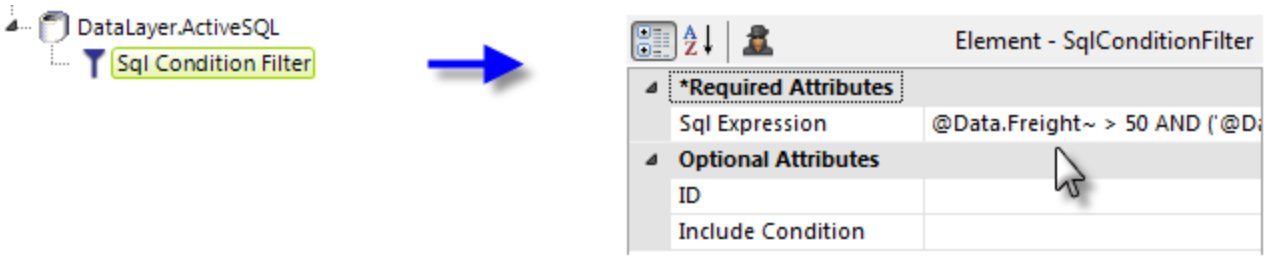
Element	Description
SQL Aggreg-	Adds a new column in the datalayer that represents an aggregate value of all rows of the datalayer. The value

Element	Description
ate Column	is populated in every row so that it can be used in further calculations, such as 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. Available aggregate functions include *Sum*, *Count*, *Average*, *Min*, or *Max*. The column's data type must be numeric for the Sum, Count, and Average functions.
SQL Cal-culated Column	Adds a new column in the datalayer created from other values in the same row, or from token values.
SQL Condition Filter	Applies a WHERE-clause like functionality to the datalayer. Its Condition attribute (an expression) is eval-uated for each row, and the row is removed when the result is *False*. The expression must written in the SQL query language of the data provider. Expressions usually contain tokens such as @Data to access specific val-ues of each data row. Expressions can include almost any combination of tokens available in Info. May be used in conjunction with SQL Compare Filters, discussed below.
SQL Group	Groups rows in the datalayer and allows grouped aggregate values to be created. Rows are grouped by val-ues in one or more datalayer columns. Grouped data can consist of just the first row of each group, or all grouped rows. Grouping elements can be nested.
SQL Para-meters	Specifies the parameters for the SQL command. Parameters are listed in the SQL command using question marks as placeholders. For example, this command requires two parameters: `SELECT * FROM Customers WHERE State=? AND City=?`

Element	Description
	Specified parameters are supplied to the command based on their *order*, and their ID is ignored.
SQL Sort	Sorts the rows of the datalayer. You may specify more than one column, using a comma-separated list of column names. Multiple Sort Sequences may also be entered in the same way.
SQL Time Period Column	Adds a new column in the datalayer that represents a time period, such as year, quarter, month, etc., derived by parsing the value in another date-time type column.
Generated Sql Plugin Call	Used to access and modify the SQL code generated by DataLayer.Active SQL, or the Active Query Builder element of an Analysis Grid. The plug-in is called right before the SQL is sent to the database and can append additional commands or modify existing commands.

Using SQL Compare Filters

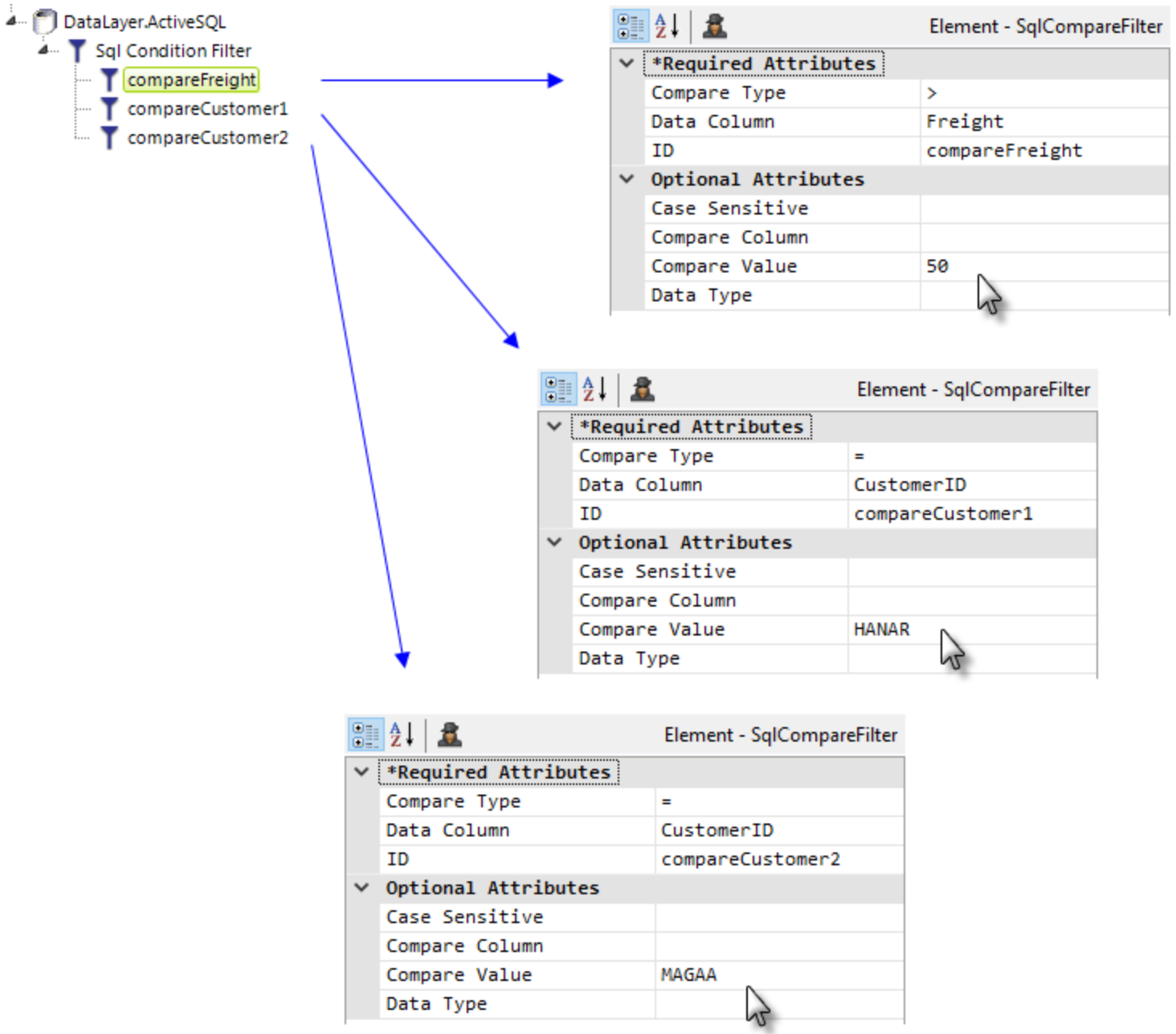
The **SQL Compare Filter**, which is used with a SQL Condition Filter element, is quite similar to *The Compare Filter* element. When used, it frequently provides *better performance* than the SQL Condition Filter element alone, because it's implemented as a native part of the Logi Data Engine and so does not need to execute script, as the SQL Condition Filter does.



Consider a typical arrangement like the one shown above, which just uses a SQL Condition Filter. The SQL Expression attribute value used is:

```
@Data.Freight~ > 50 AND ('@Data.CustomerID~' = 'HANAR' OR '@Data.CustomerID~' = 'MAGAA')
```

There are three comparisons that must evaluate to *True* in order for a row in the datalayer to be retained and each is run using scripting. Now let's see how we can obtain maximum performance by spreading those evaluations out, using the native power of SQL Compare Filter:


DataLayer.ActiveSQL
Sql Condition Filter
compareFreight
compareCustomer1
compareCustomer2
Element - SqlCompareFilter
*Required Attributes
Compare Type
>
Data Column
Freight
ID
compareFreight
Optional Attributes
Case Sensitive
Compare Column
Compare Value
50
Data Type
Element - SqlCompareFilter
*Required Attributes
Compare Type
=
Data Column
CustomerID
ID
compareCustomer1
Optional Attributes
Case Sensitive
Compare Column
Compare Value
HANAR
Data Type
Element - SqlCompareFilter
*Required Attributes
Compare Type
=
Data Column
CustomerID
ID
compareCustomer2
Optional Attributes
Case Sensitive
Compare Column
Compare Value
MAGAA
Data Type

In the example above, three SQL Compare Filter elements have been added beneath the SQL Condition Filter element. Each has a unique ID and its attributes are set to handle one of the three comparisons from our previous example and each of these "sub evaluations" will return a *True* or *False* result.

Finally, we change the SQL Condition Filter element's SQL Expression attribute to evaluate the results of the three SQL Compare Filter "sub evaluations", like this:

```
@Compare.compareFreight~ AND (@Compare.compareCustomer1~ OR @Compare.compareCustomer2~)
```

Note the use of a special token, @Compare, to represent the results of each "sub evaluation".

This approach not only provides better performance for large data sets, it may also be easier to use when there are many more comparisons and/or more complexity involved.

This table provides information about the SQL Compare Filter element's **Compare Type** options:

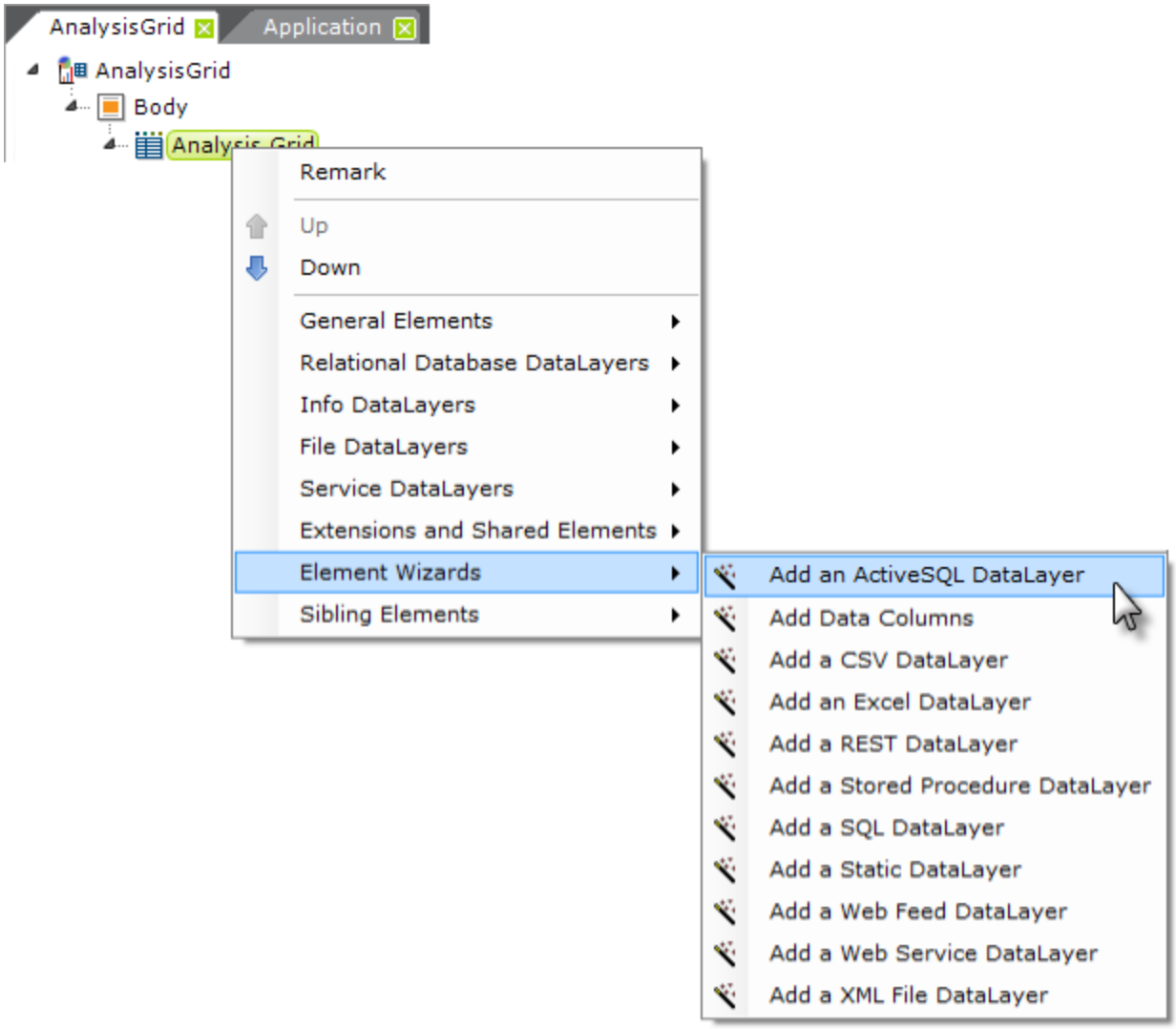
Operator	Description
=, <, >, <=, >=, < >	Basic standard comparison operations. String types evaluated are culture-sensitive.
InDay	For the row to be retained, the date value in the specified Data Column must be within the same day as the Compare Value.
InList	For the row to be retained, the value in 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

Operator	Description
	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.
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.

The **Compare Column** attribute allows you to compare two columns, rather than a column and a value. It and the Compare Value attribute are mutually exclusive and it supports all of the operators shown above *except* InList and NotInList.

DataLayer.ActiveSQL - Using Studio's DataLayer Wizard

Studio includes a wizard that can assist you in configuring DataLayer.ActiveSQL. The wizard assumes that you have already added an appropriate database **Connection** element in the _Settings definition and configured it.



As shown above, the wizard can be started by selecting and right-clicking the parent element under which you want to add the datalayer, and using the context menus to select "Add an ActiveSQL DataLayer". The wizard will open and ask you to select a connection from the list of those defined in the _Settings definition, and to provide a SQL query. It will then insert the configured datalayer element into your definition.

DataLayer.Bookmarks

The **DataLayer.Bookmarks** element is used specifically to retrieve saved bookmark information.

The following sections are covered in this topic:

- Attributes
- Working with DataLayer.Bookmarks

For more information, see *Bookmarks*. Bookmark implementation examples can be found in our Bookmarks Sample Application.

Attributes

The DataLayer.Bookmarks element has the following attributes:

Attribute	Description
ID	An optional element ID. Recommended for easier identification when debugging.
Bookmark Collection	(Required) Bookmarks are stored in XML data files, called a Bookmark Collection. These are located in a folder specified in the Bookmark Location attribute of the Settings definition's General element. This attribute sets the name of the collection; only the file name, without an extension, is entered here. Bookmarks can also be stored in a database. For more information, see the Storing Bookmark, Gallery, and SaveFiles in a Database section of *Bookmarks*. A value is *not* required here if the General element's Bookmark Collection Default has been specified in the _Settings definition.

Working with DataLayer.Bookmarks

This datalayer connects directly to the bookmark collection via the file system or, optionally in v12.5+, by querying a database. Unlike most datalayer elements, this one *does not* require aConnection element. Bookmark information is retrieved and cached as rows and columns, with one row per bookmark. Data retrieved into the datalayer is cached in XML format. The data retrieved with a datalayer is available using @Data tokens, in the format @Data.*ColumnName*~ and the spelling of the column name is case-sensitive. The data is only available within the scope of the parent element of the datalayer, not throughout the entire report definition. The DataLayer.Linked element can be used to make the data reusable in another datalayer outside this scope.

A "bookmark" consists of a record in an .xml file, similar to this (format may vary depending on Logi Info version):

```
<rdData>
    <dtBookmarks IsShared="True" BookmarkUserName="HCaulfield"
        BookmarkCollection="HCaulfieldCollection" FolderID="956ecb53-92a1-427e-9ed2-
        31a2a4507d5f" ExtraFile="goCollection_35d96c86-5d9b-4913-8cc1-
        2696bd579054.xml" BookmarkID="35d96c86-5d9b-4913-8cc1-2696bd579054"
        SaveTime="2014-10-08T15:29:02-04:00" Description="Shared Q3 Analysis"
        CustomColumn2="" CustomColumn1="" Name="" Report="InfoGo.goAnalysisGrid"/>
</rdData>
```

Bookmark data is retrieved into the datalayer in columns with the following names:

Column Name	Contains
BookmarkCollection	The name of the bookmark collection this bookmark belongs to.
BookmarkID	Aunique, system-assigned GUID value identifying the bookmark.
BookmarkUserName	The name of the user who created this bookmark.

Column Name	Contains
CustomColumn1	An optional value provided by the developer for customization purposes.
CustomColumn2	An optional value provided by the developer for customization purposes.
Description	An arbitrary text description given to the bookmark by the user.
ExtraFile	A GUID-based file namefor asuper-element bookmark.
FolderID	Aunique, system-assigned GUID value identifying the logical folder containing this bookmark.
IsShared	A flag indicating whether this bookmark is shared.
Name	An arbitrary name given to the bookmark.
Report	The name of the report associated with the bookmark.
SaveTime	The bookmark creation timestamp, in ISO 8601 format (e.g. "2014-06-23T09:32:11-04:00").

The data retrieved into the datalayer can be viewed during development by setting **General** element's **Debugger Style** attribute in the _Settings definition and using the resulting link at the bottom of the report page to view the Debugger Trace page. A link on the Trace page will display the retrieved data.

DataLayer.Cached

The **DataLayer.Cached** element retrieves data that has been cached to disk on the web server. It manages the lifetime and refresh rate of the cached data and uses other datalayers to do the data retrieval from a datasource when necessary. This allows a "snapshot" of the data to be used to generate reports, and reduces the number of retrievals that must be run against the data-source.

The following topics introduce the DataLayer.Cached element:

- DataLayer.Cached Attributes
- Working with DataLayer.Cached
- DataLayer.Cached Special Considerations

DataLayer.Cached - Attributes

The **DataLayer.Cached** element has the following attributes:

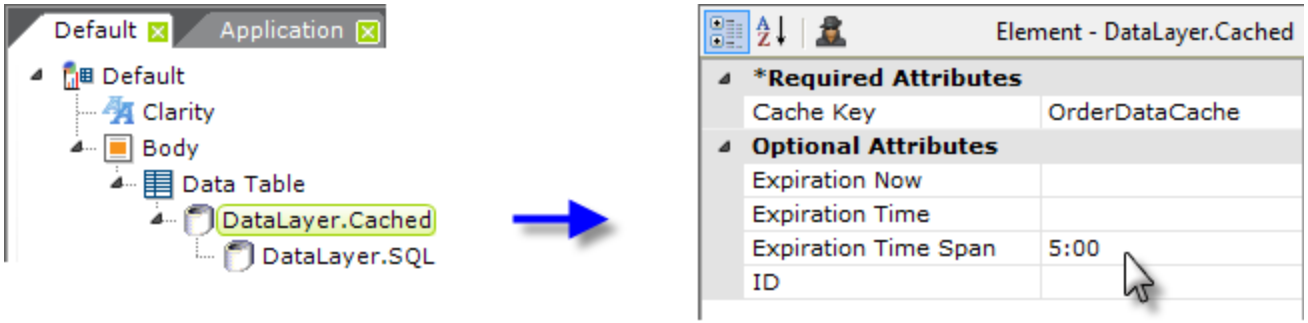
Attribute	Description
Cache Key	(Required) An arbitrary string value assigned by the developer. This is a unique identifier for the cached data, used to identify the cache when it's created and used with any subsequent requests for access to the cached data. It has application-wide scope, so the cache can be shared among all user sessions. The value can contain tokens in order to dynamically identify the cache to be used based on request variables or other values. Caches can be made user-specific by using the token @Session.ID~ in this attribute. Developers can also identify a cache with multiple identifiers or tokens, in a comma-separated list.
ID	An optional element ID. Recommended for easier identification when debugging.
Expiration Now	If set to *True*, causes the data cache to be immediately deleted and recreated. Primarily useful for development purposes. Default value: *False*
Expiration Time	The **time of day** at which the data cache expires, causing it to be deleted and recreated by the next user request. Uses the 24-hour clock in the format of *hh:mm* or *h:mm*. For example: 5:00 is 5:00 a.m. and 14:24 is 2:24 p.m. Each expiration-related attribute operates independently of other expiration-related attributes.
Expiration Time Span	Sets the **lifetime**, in hours and minutes, of the data cache. Once this amount of time has passed since the creation of the cache, it will be deleted and recreated by the next user request. For example: a value of 1:34 represents one hour and 34 minutes, which will cause a cache created at 2:00 p.m. to expire at 3:34 p.m. Values without a colon are treated as minutes. Each expiration-related attribute operates independently of other expir-

Attribute	Description
	ation-related attributes.

DataLayer.Cached - Working with DataLayer.Cached

Reports do not always need to use the absolutely most-current data. Data generated a few hours ago, yesterday, or last month is frequently what's required. In these circumstances, there's no point in burdening datasources by requesting this kind of data from them every single time a report is run and DataLayer.Cached offers the flexibility of an alternate approach.

DataLayer.Cached modifies the normal data retrieval activities of datalayers; when it's used, data is retrieved, cached, and made available for use in Logi reports for a specific time period, after which the data is refreshed (deleted and recreated). Its primary purpose is to reduce the number of data retrieval operations, such as complex SQL queries, that need to be run against the data-source.



In the example shown above, a **DataLayer.Cached** element has been added as a child element beneath a Data Table and its attributes set as shown. DataLayer.Cached requires a child datalayer element, which retrieves the data from the original data-source when necessary and can be any of the other datalayer elements available. The data lifecycle will then be as follows:

1. When the report is requested for the *first* time, the data will be retrieved by the DataLayer.SQL and cached in an XML data file in the applications' rdDataCache folder. The request will then be satisfied using this data.

2. The next time the report is requested by any user, it will be satisfied using the cached data in the XML data file; the SQL query will *not* be executed. This will remain the case for any subsequent requests until the cache expires (five hours after it was created, based on the Expiration Time Span attribute value above of *5:00*).
3. If the cache has expired, the next time the report is requested by any user, the XML data file will be deleted and the cycle will begin again with Step #1.

You can add child elements beneath DataLayer.Cached and/or its child datalayers to modify the data, including:

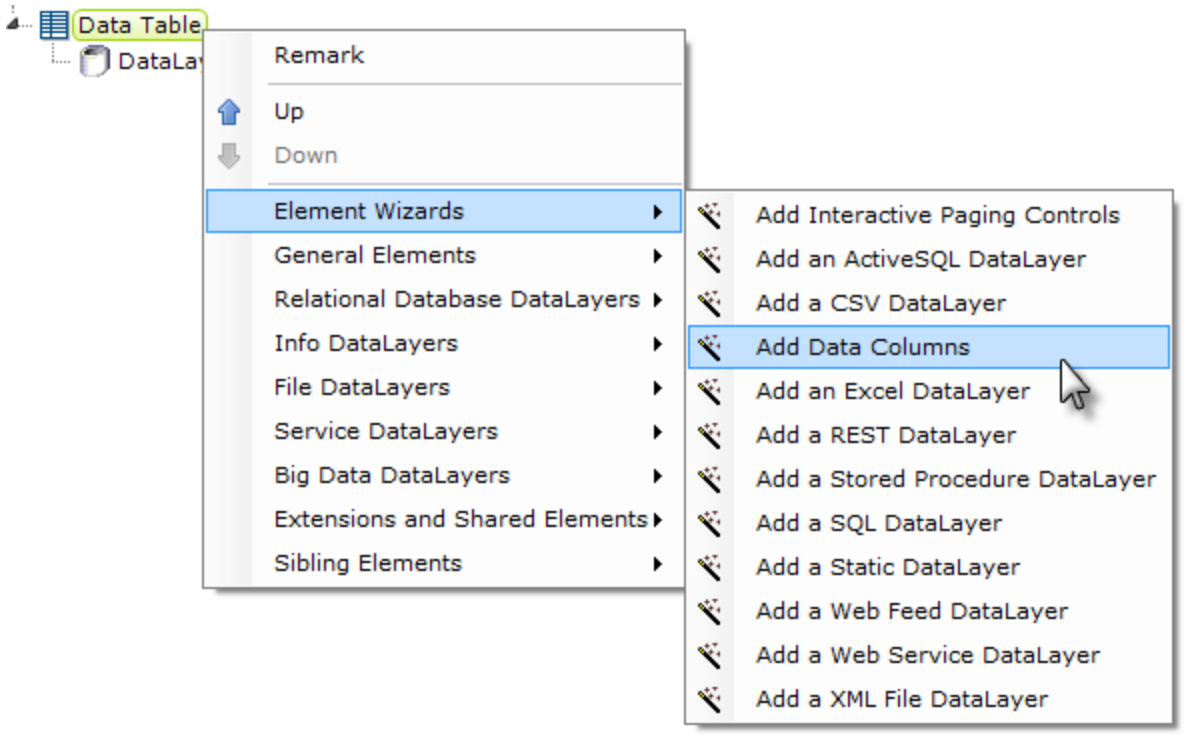
- **Filtering**: Sort, group, or restrict the data
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled data values
- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report definitions

Data retrieved into the datalayer is cached in **XML** format, in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

The data in the datalayer is available using **@Datatokens**, in the format @Data.*ColumnName*~. The spelling of the column name is **case-sensitive**. The data is only available within the **scope** of the **parent** element of the datalayer, not throughout the entire report definition. The **DataLayer.Linked** element can be used to make the data reusable in another datalayer outside this scope.

The *Auto Columns* element can be used to quickly display in your report all the data in a datalayer.

The data retrieved into the datalayer can be viewed by turning on the **Debugging Link** in your _settings definition (**General** element) and using the resulting link at the bottom of your report page to view the **Debugger Trace Report** page. There will be clear indications when the cache is being built and when data from it is being used. A link on the Trace page will display the actual cached data.



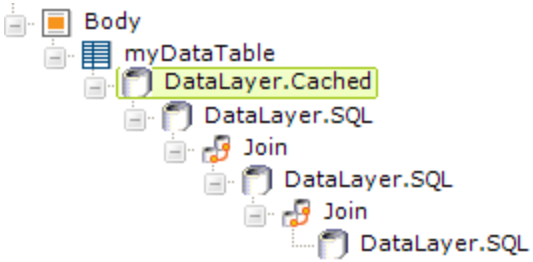
Studio includes a **wizard** that will add elements for the **columns** in the datalayer to your definition. You can select the desired columns before the operation begins. With the datalayer's parent **Data Table** or similar element selected in the Definition Editor, click the wizard link, shown above, in the Element Toolbox.

DataLayer.Cached - DataLayer.Cached Special Considerations

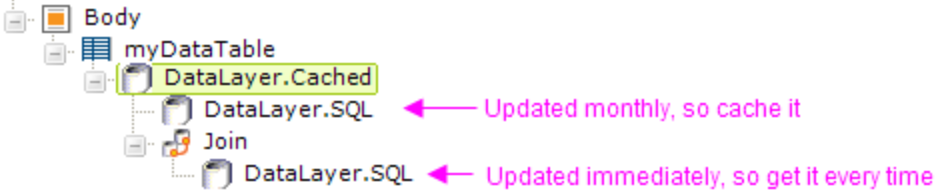
The following should be considered when using this datalayer:

Joins

Developers may choose to retrieve data from multiple sources and use **Join** elements to combine it. How does DataLayer.Cached work in this scenario?



As shown in the example above, all of the datalayers and joins can be placed beneath one DataLayer.Cached element and the data resulting from *all* of the joins will be cached. This is fine if the currency of all of the data sources is similar. However, what if some of the data is updated more frequently than the rest?



One approach when data has mixed currency, as shown above, is to use DataLayer.Cached to cache "long-term" data that *does not* change frequently and then join that data with another datalayer that retrieves "short-term" data that *does* change frequently.

If you have a large number of joins, then this approach depends upon your ability to relate the data appropriately (no short- and long-term datalayers interleaved).

What about caching at different levels of the join? Sorry, it's not possible to use a DataLayer.Cached element beneath a Join element.

Time to Service Requests Can Vary

Typically, systems that cache data "snapshots", such as data marts and mini data warehouses, use independent processes to monitor the cache's expiration time and refresh it. However, no such process exists for DataLayer.Cached; cache expiration is tested with every request and the cache is refreshed when expiration is detected. This may result in some requests (those that cause the cache to be refreshed) taking longer than others, as the child datalayer that actually retrieves the data will then run.

Scope Issues

This datalayer can be used in multiple report definitions to access the same cached data (by using the same Cache Key value). It's very important, in such circumstances, that all instances of DataLayer.Cached, and its child elements, which access the same cache be *configuredidentically*. They must have the exact same attribute settings and the exact same child datalayer, otherwise the cached data will be different and/or be set to expire differently, depending on which report definition's request causes it to be refreshed. This is a good reason to consider implementing this datalayer as a Shared Element, see *Shared Elements*.

Not for Use with Subdatalayers

At this time, DataLayer.Cached should *not* be used with reports that include **Subdata Layer** elements; an error will result.

DataLayer.Data Services

The **DataLayer.Data Services** element is a special type of datalayer designed for use with Logi Platform Services.

The following topics introduces the DataLayer.Data Services element:

- Logi Services
- DataLayer.Data Usage Details
- DataLayer.Data Services Attributes
- Working with DataLayer.Data Services
- Selecting a Dataview
- Special Data Services Child Elements
- Using DsCompare Filters

About DataLayer.Data Services


Many of the data-related elements discussed here are available in Logi Info v12.5 but have been deprecated in later Info versions; consult the Release Notes for specific details.

The **DataLayer.Data Services** element uses Logi Platform Services for data retrieval and is only available in Logi Info if the Discovery Module v3.0 or Logi Platform Serviceshas been installed. It executes a Dataview, processes the resulting data, and presents it in the XML format common to Logi Info datalayers.

DataLayer.Data Services - Logi Services

Introduced to Logi Info in v12.5, Logi Services technology is used to create, manage, and query "Dataviews". A Dataview is a "virtual view" of data that has been drawn from one or more datasources, enriched as needed (by formatting, grouping, aggregating, etc.), and made available as tabular data set, with numbered rows and named columns. Dataviews are created using tools like the Dataview Authoring tool in Logi Studio. The Dataviews used with Logi Info can retrieve data from a variety of databases; Dataviews used with our DataHub 3.0+ product can also retrieve, cache, and blend data from online applications such as Facebook, Google Analytics and Marketo. For more information about this technology and Dataviews, see *Dataviews*.

DataLayer.Data Services - DataLayer.Data Services Usage Details

Here are some important details about using this datalayer:

- DataLayer.Data Services uses the special **Connection.Logi Application Service** element, which connects it to the built-in web service installed with the Discovery Module 3.0, *not* directly to a datasource. The Dataviews executed by the datalayer specify the required datasource connection details themselves. Connection configuration details are available in Connecting to the Logi Application Service.
- DataLayer.Data Services provides data for numerous parent elements, including Local Data, Data Table, certain Input, and all Series elements. The data is available using @Local or @Data tokens, as usual.

- DataLayer.Data Services has optional, specialized "Ds" child elements which can be used to introduce additions to a Dataview at execution. These include custom filtering, grouping, and sorting, and creation of calculated and aggregation columns. See "DataLayer.Data Services - Attributes" on the next page for details. A small set of standard child elements, e.g. Crosstab Filter and Time Period Column, which act on the returned data in the standard fashion are also available for use with this datalayer.

DataLayer.Data Services - Attributes

The DataLayer.Data Services element has the following attributes:

Attribute	Description
Dataview ID	(Required) Specifies the ID of a Dataview definition, stored at the Logi Application Server, to be used to provide data for the datalayer. Once a Logi Application Service ID has been specified in the next attribute, the browse button in this attribute's value can be used to browse for available Dataviews.
Logi Application Service ID	(Required) Specifies the element ID of a **Connection.Logi Application Services** element in the _Settings definition.
ID	Specifies a unique element ID. Providing a value here is *highly recommended* for easier identification of data when debugging.
Maximum Rows	The maximum number of rows to retrieve from the datasource, per request. Default: *200 rows* or, if pagination is being used, *10 pages*, whichever is greater.

DataLayer.Data Services - Working with DataLayer.Data Service

The data-related elements discussed here are available in Logi Info v12.5 but have been deprecated in later Info versions; consult the Release Notes for specific details.

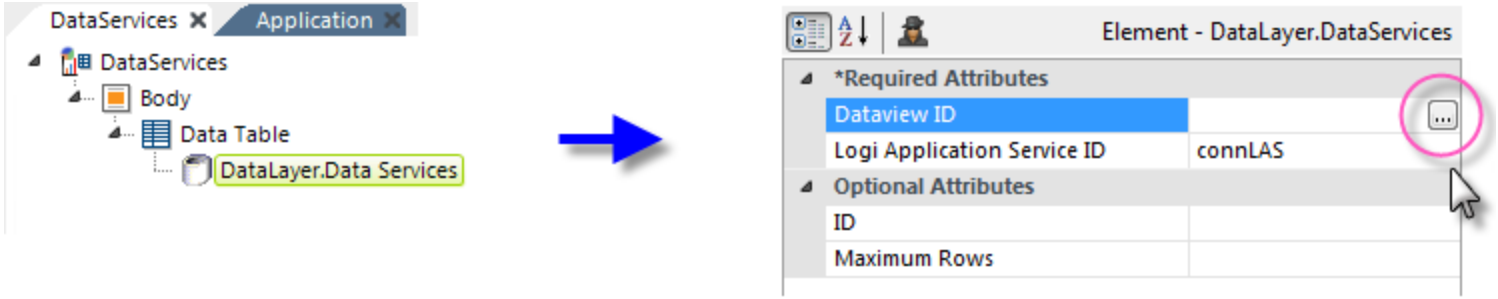
Like other datalayers, DataLayer.Data Services has a few standard child elements that can be used to shape the retrieved data. In addition, it has some special "Ds" child elements (discussed below) that developers can use to augment the Dataview when it's executed. However, other child elements for linking datalayers and formatting or transforming the data are not available.

The data retrieved is available using @Data tokens, in the format @Data.*ColumnName*~. The spelling of the column name is *case-sensitive*. The data is only available within the scope of its parent element, not throughout the entire report definition.

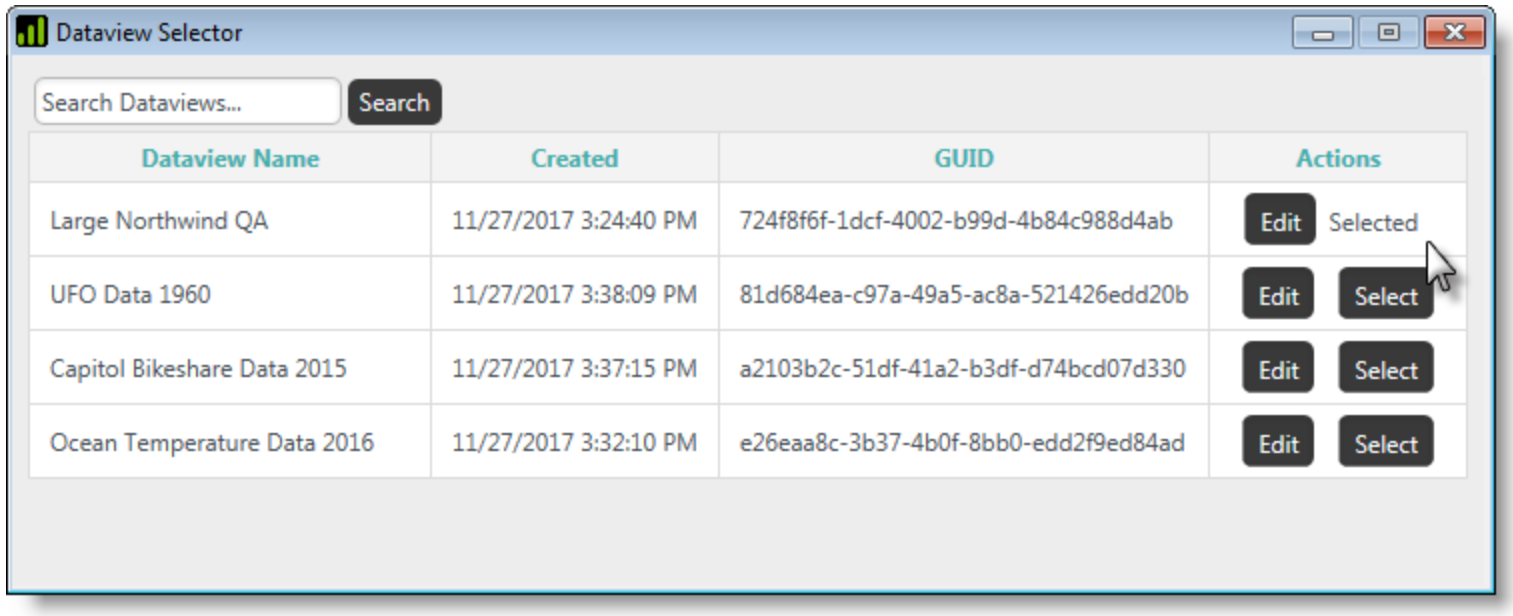
The data retrieved into the datalayer can be viewed by turning on debugging in Logi Studio and using the resulting link at the bottom of your report page to view the Debugger Trace page, see *Debug Reports*. You'll be able to see the XML data retrieved.

DataLayer.Data Services - Selecting a Dataview

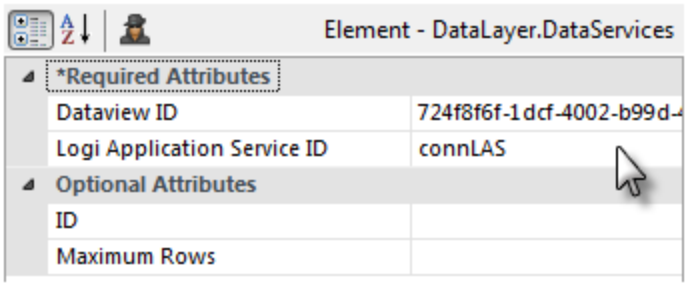
The following assumes that you've already added and configured a Connection.Logi Application Services element in _Settings and have used Logi Studio's Dataview Authoring tool to create at least one Dataview.



As shown above, add a DataLayer.Data Services element beneath the table or visualization of your choice and configure the **Logi Application Service ID** with the ID of your Connection element. Click the browse button at the end of the **Dataview ID** attribute.



The Dataview Selector dialog box will appear in Studio, displaying a searchable list of available Dataviews. Click the **Select** button of the desired Dataview.



The dialog box will close and the GUID of the selected Dataview will be automatically copied into the datalayer's Dataview ID attribute, as shown above. To select a different Dataview, just repeat the process.

DataLayer.Data Services - Special Data Services Child Elements

The data-related elements discussed here are available in Logi Info v12.5 but have been deprecated in later Info versions; consult the Release Notes for specific details.

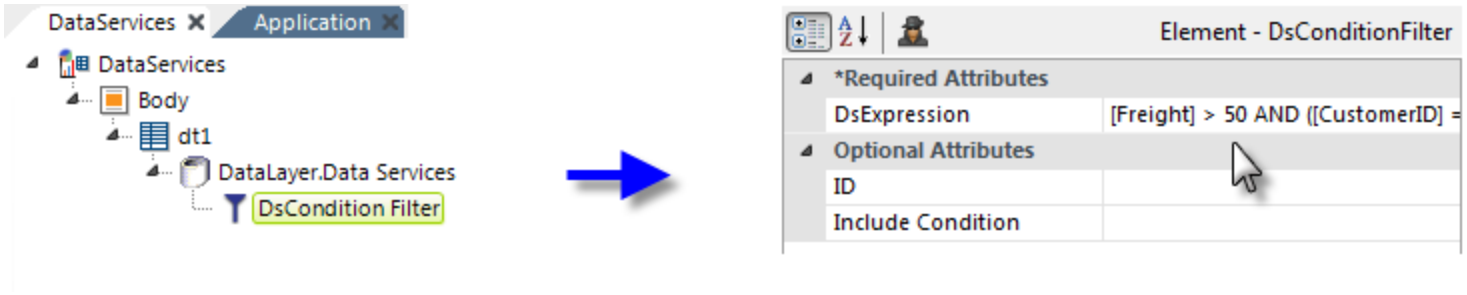
DataLayer.Data Services has some unique child elements that can be used to shape the data. Unlike similar child elements for regular datalayers, which modify the *data* in the datalayer, these elements modify the *Dataview* just before its execution.

Element	Description
The DsAggregate Column	Adds a new column to the Dataview containing an aggregate value of all rows of the datalayer. The value is populated in every row so that it can be used in further calculations, such as Summary Rows, Link Params, etc. Available aggregate functions include *Average*, *Count*, *Distinct, Max, Min*, and *Sum.* The column's data type must be numeric for the Average, Count, Distinct, and Sum functions.
The DsCalculated Column	Adds a new column to the Dataview, created from other values in the same row or from token values, containing the results of an expression. Its **DsExpression** attribute value contains a Data Services-style expression which returns the result. In these expressions, the value of a data column is indicated by enclosing the column name in square brackets, not with an @Data token. Example expression: `[Quantity] * [UnitPrice] - .05`See *Dsexpression Reference* for guidelines regarding expressions.
The DsCondition Filter	Adds a row filtering operation to the Dataview, removing rows whose **DsExpression** expression evaluates to *False*. Its DsExpression attribute value must be a Data Services-style expression which evaluates to *True* or *False*. In these expressions, the value of a data column is indicated by enclosing the column name in square brackets, not with an @Data token. Example expression: `[Quantity] > @Request.Threshold~`See *Dsexpression Reference* for guidelines regarding expressions. This element may be used in conjunction with **DsCompare Filter** elements, as discussed in "DataLayer.Data Services - Using DsCompare Filters" on

Element	Description
	page 274.
DsGroup	Adds a grouping operation to the Dataview, grouping rows based on the values of one or more columns. The names of the columns to be used for grouping are entered in the **Group Column** attribute, in a comma-separated list. Grouped row values can be aggregated by adding a **DsAggregate Column** element as a child of DsGroup. When grouping is applied, only values from the columns named in the Group Column and, if aggregating, the Aggregate Column attributes are returned to the datalayer.
DsSort	Adds a sorting operation to the Dataview, sorting rows based on the values of one or more columns in the **Sort Column** attribute. You may specify more than one column, using a comma-separated list of column names. Multiple **Sort Sequence** attribute values may also be entered in the same way, corresponding to the named columns.

DataLayer.Data Services - Using DsCompare Filters

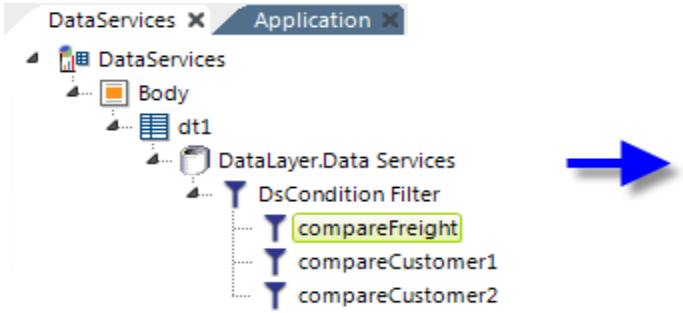
The **DsCompare Filter**, which is used with a DsCondition Filter element, is quite similar to the standard *The Compare Filter* element. It may be easier to organize a complex filter using this element.



Consider a typical arrangement like the one shown above, which just uses a DsCondition Filter. The DsExpression attribute value used is:

```
[Freight] > 50 AND ([CustomerID] == 'HANAR' OR [CustomerID] == 'MAGAA')
```

There are three comparisons that must evaluate to *True* in order for a row in the data to be retained. Now let's see how we can might improve performance by spreading those evaluations out, using DsCompare Filters:



Element - DsCompareFilter

*Required Attributes	
Compare Type	>
Data Column	Freight
ID	compareFreight
Optional Attributes	
Compare Value	50
Data Type	Number
Include Condition	

Element - DsCompareFilter

*Required Attributes	
Compare Type	==
Data Column	CustomerID
ID	compareCustomer1
Optional Attributes	
Compare Value	HANAR
Data Type	Text
Include Condition	

Element - DsCompareFilter

*Required Attributes	
Compare Type	==
Data Column	CustomerID
ID	compareCustomer2
Optional Attributes	
Compare Value	MAGAA
Data Type	Text
Include Condition	

In the example above, three DsCompare Filter elements have been added beneath the DsCondition Filter element. Each has a unique ID and its attributes are set to handle one of the three comparisons from our previous example and each of these "sub evaluations" will return a *True* or *False* result. Finally, we change the DsCondition Filter element's DsExpression attribute to evaluate the results of the three DsCompare Filter "sub evaluations", like this:

```
@Compare.compareFreight~ AND (@Compare.compareCustomer1~ OR @Compare.compareCustomer2~)
```

Note the use of a special token, @Compare, to represent the results of each "sub evaluation". This approach not only provides better performance for large data sets, it may also be easier to use when there are many more comparisons and/or more complexity involved. This table provides information about the DsCompare Filter element's **Compare Type** options:

Operator	Description
=, <>, <, >, <=, >=	Basic standard comparison operators. String types evaluated are culture-sensitive.
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.
StartsWith	For the row to be retained, the text value in the specified Data Column *must* start with the text in the Compare Value.

DataLayer.Definition List

The **DataLayer.Definition List** element provides developers with programmatic information about the definitions used in their application. The data can then be used to create report menus, report management systems, and more.

The following topics introduce the DataLayer.Definition List element:

- DataLayer.Definition List Attributes
- Working with DataLayer.Definition List
- Example Usage: With an Input Select List

For similar functionality for any file (not just definitions) see our "DataLayer.Directory" on page 78 element.

DataLayer.Definition List - Attributes

The DataLayer.Definition List element has the following attributes:

Attribute	Description
ID	Specifies an optional element ID. We recommend you specify onefor easier identification when debugging.
Definition List Folder	Enter a file system folder name here to restrict the results returned to definitions within a single folder. The folder named is assumed to be within the _Definitions folder of the application's project folder. Examples: *_Processes, _Reports, _Templates, _Widgets*Default: Returns data for all definitions

DataLayer.Definition List - Working with DataLayer.Definition List

The datalayer reads the definition data for the current project and caches it as rows and columns, with one row per definition file. Data retrieved into the datalayer is cached in the usual XML format.

The data retrieved with a datalayer is available using @Data tokens, in the format @Data.*ColumnName*~. The spelling of the column name is *case-sensitive*. The data is only available within the scope of the parent element of the datalayer, not throughout the entire report definition. The DataLayer.Linked element can be used to make the data reusable in another datalayer outside this scope.

Column returned into the datalayer and available through the @Data token are:

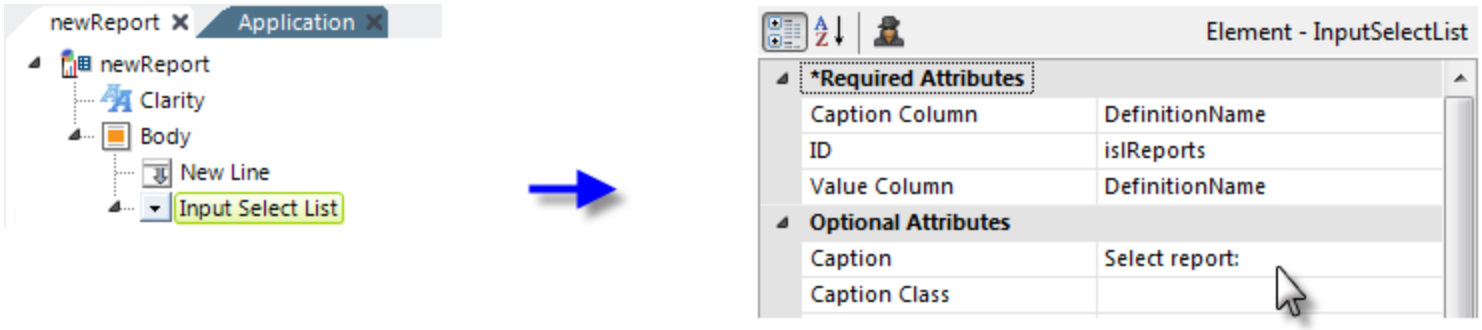
Column Name	Description
DefinitionName	The definition name, as it appears in the Application panel, e.g. *Default*
DefinitionType	The definition type, e.g. *Report*, *Process*, *Template*, or *Widget*
Caption	The Caption attribute value (for Report definitions).
EngineVersion	The build number of the version of the Logi Server Engine used, e.g. *12.0.30*
Filename	The fully-qualified filename of the definition file, e.g. *c:\inetpub\wwwroot\TestApp_Definitions_Reports\CrosstabSample.lgx*
Folder*n*	For easier organization, Logi definition files can be named using period-separated prefixes, such as `Accounting.ProfitLoss.lgx`. The prefixes are referred to as folders, though they don't exist in the file system as such. This column returns the *n*th prefix or folder name for each definition. Examples: For `Account-`

Column Name	Description
	`ing.Profits.lgx`, the Folder1 column returns "Accounting" For `New.Accounting.Profits.lgx`, the Folder2 column returns "Accounting" *Due to the optional nature of these prefixes, the AutoColumns element may not detect all instances of this column in the datalayer.*
ID	The Report ID from the report definition's root node, ID attribute.
SavedAt	The timestamp indicating when the definition was last saved, e.g. *3/8/2008 2:25 PM*
SavedBy	User name of the person who last saved the definition.

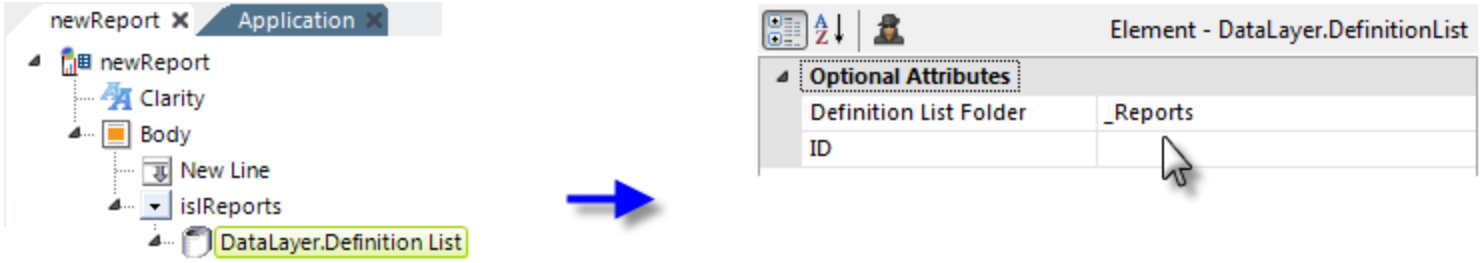
If the **Alternative Definition Folder** attribute is set in the _Settings definition's **Path** element, the datalayer also includes definition objects from that folder. The data retrieved into the datalayer can be viewed by turning on the Debugging Link in the _Settings definition (General element) and using the resulting icon at the bottom of the report page to view the Debugger Trace Page. A link on the Trace page will display the retrieved data.

DataLayer.Definition List - Example Usage: With an Input Select List

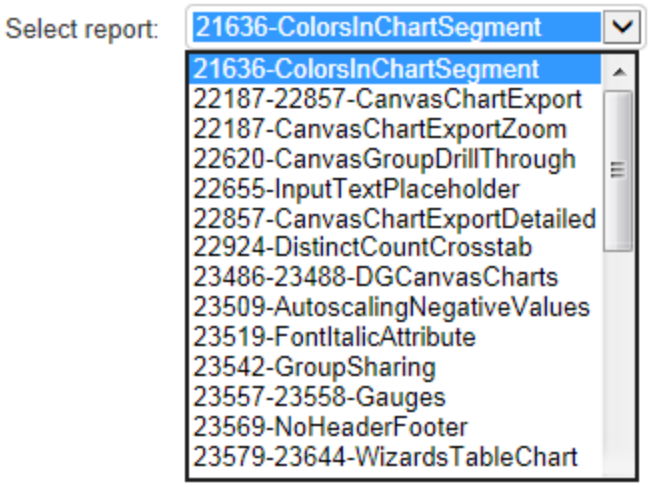
In the following example, you'll see how to use DataLayer.Definition List with an **Input Select List** element, so that users can select report definitions to see at runtime.



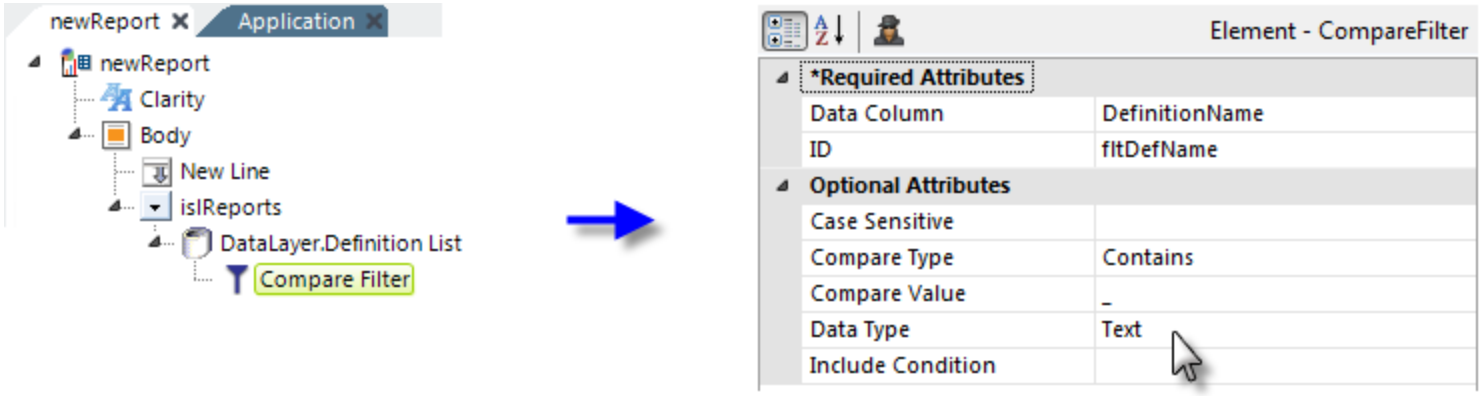
Start by adding an **Input Select List** element to your report definition and setting its attributes as shown above. Referring to the table of returned columns from "DataLayer.Definition List - Working with DataLayer.Definition List" on page 279, use "DefinitionName" for the Caption Column and Value Column attribute values.



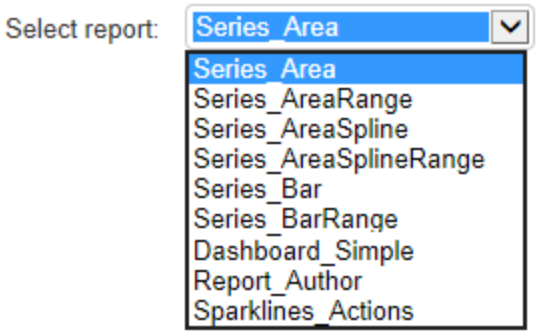
Next, add a **DataLayer.Definition List** element beneath the input element, as shown above. You only want to present users with a choice of *report* definitions, so set the Definition List Folder attribute to *_Reports*. Without this, they'd also see Process, Widget, and all other definitions.



When you preview the report, you see that your list is populated, but there's a problem: the list includes many report definitions in subfolders that you don't want users to see. To fix this, filter the datalayer:



As shown above, add a **Compare Filter** beneath the datalayer. Set its attributes to keep only definitions that have an underscore in their names.



Now when we preview the definition again, we see that the list includes the desired set of definitions.

DataLayer.MDX

The **DataLayer.MDX** element allows specification of an OLAP MDX query command that returns a cell set and populates an OLAP Table or OLAP Grid element.

The following topics introduce the DataLayer.MDX element:

- DataLayer.MDX Attributes
- Entering an MDX Query
- Using MDX Query Elements

For additional information, see *Logi XOLAP*. This element is not available for Logi Java applications.

About DataLayer.MDX

The **DataLayer.MDX** element is used as a child of either an OLAP Table or an OLAP Grid element.

When used with an OLAP Table only, the text of anMDX query can be entered directly into the DataLayer.MDX element's **MDX Source** attribute, or generated usingthe special **MDX Query Builder** tool, which can beaccessed by clicking the browse button at the end of the MDX Source attribute. This tool allows developers to createacorrectly-formatted MDX query using a graphical interface, similar to the SQL Query Builder found in DataLayer.SQL.

An alternative approach, when DataLayer.MDX is used with an OLAP Table or OLAP Grid, is to skip entering the query text directly and instead add an **MDX Query** child element. You can use it and its child elements tospecify the dimensions, measures, and filters needed and the Logi Engine will generate a correct MDX query at runtime based on them.

Theyappear on the top and left axes. The order in which dimensions are defined for an axis (the element order) determines the order in which they are displayed. For example, if a dimension showing "customer city" is provided first, followed by a dimension showing "customer marital status", the user will be able to drill down to a particular city and then drill down on marital status to see measures for each marital status in that city.

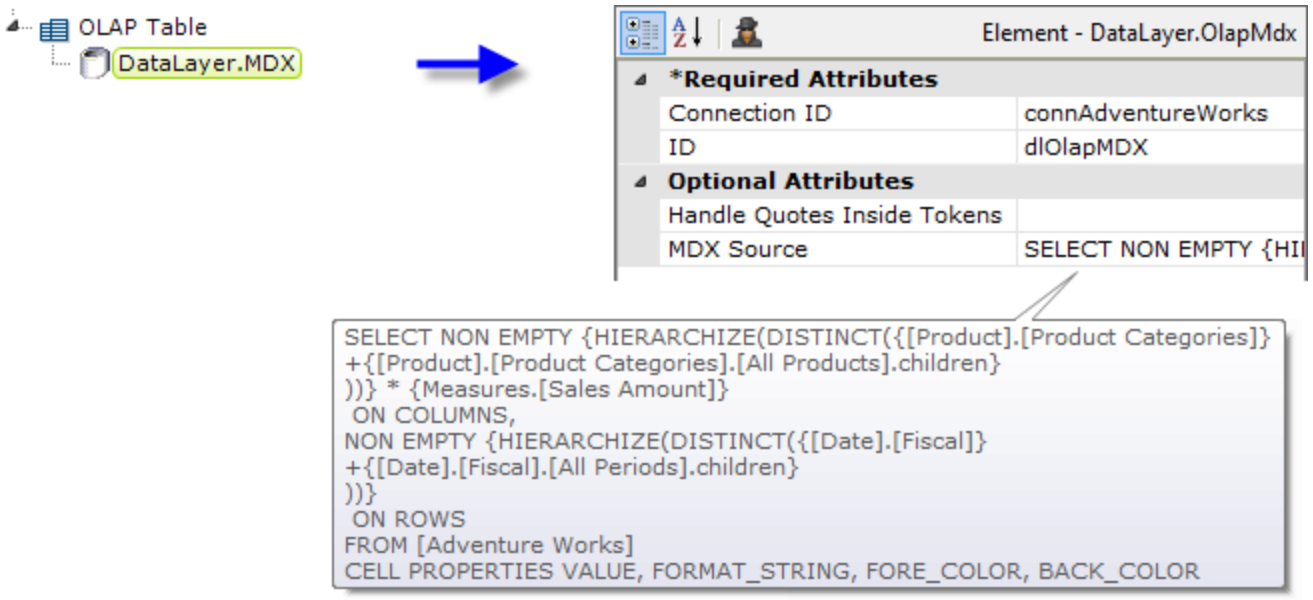
DataLayer.MDX - Attributes

The **DataLayer.MDX** element has the following attributes:

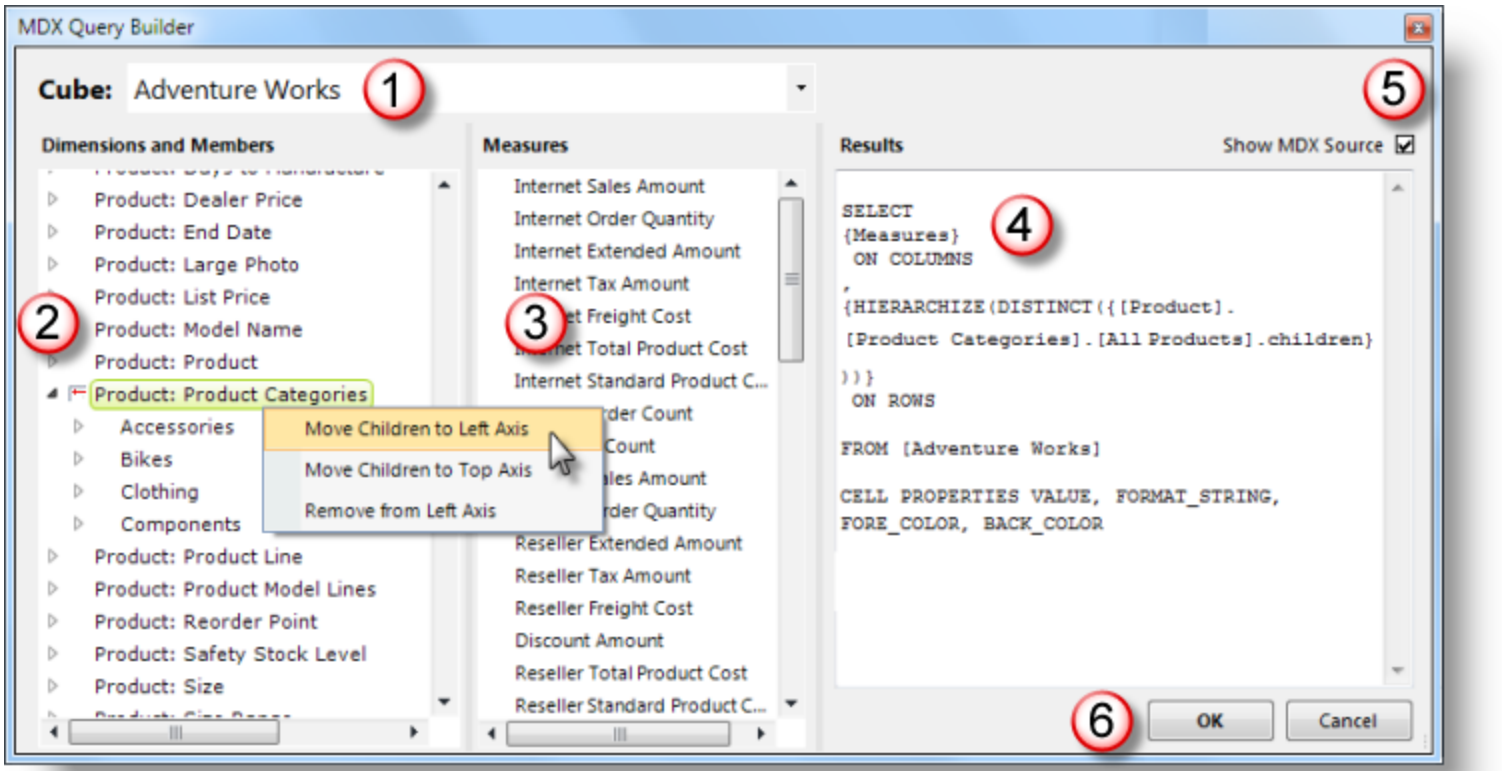
Attribute	Description
Connection ID	(Required) Specifies a connection to a data source, which has been defined in the _Settings definition.
ID	(Required) A unique element name.
Handle Quotes Inside Tokens	Enables special token processing, allowing use of token values that contain single quotes. When set to *True*, the single quotes in the values of any tokens in the MDX Source attribute will be "doubled" so that they work within aSQL statement. The default for value is *False*.
MDX Source	Specifies an optional developer-coded MDX command to be sent to the OLAP server. Only applicable when used with an OLAP Table parent element; OLAP Gridsmust use the **MDX Query** element instead. The Browse button for this attribute can be used to launch the **MDX Query Builder** tool.

DataLayer.MDX - Entering an MDX Query

When working with an **OLAP Table** element, you can enter an MDX query directly into the DataLayer.MDX element's **MDX Source** attribute. This query is executed at runtime against the datasource associated with the element's Connection ID attribute.



The example above shows a query in the MDX Source attribute value. As you can see, this type of query is relatively complex and prone to typos during entry. In order to help you formulate a correct query, instead of typing in the query you can use the **MDX Query Builder** tool, which is accessed by clicking the Browse button at the end of the MDX Source attribute value.



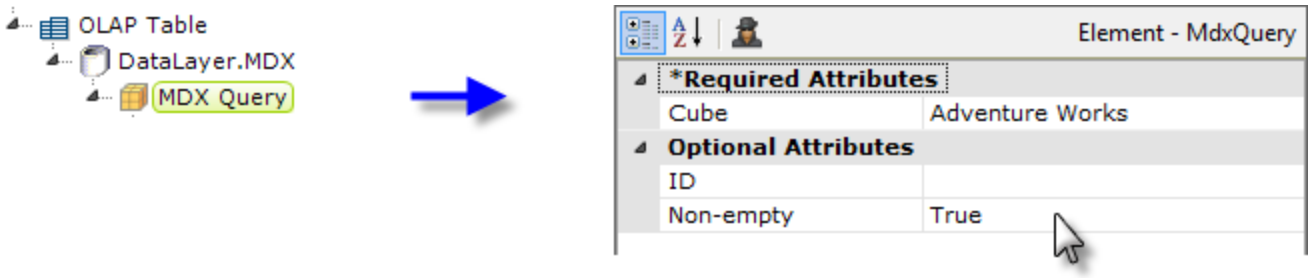
Based on the selected (1) Cube, the MDX Query Builder presents lists of (2) Dimensions and Members, **and** (3) Measures. These can be moved to an axis or added and removed by selecting and right-clicking them, as shown above. The (4) results of these actions can be seen in the Results panel, in tabular or (5) source code form. Clicking (6) **OK** will cause the query constructed in the Results panel to be copied into the datalayer's **MDX Source** attribute, if the MDX Query Builder was called by clicking the attribute's browse button. Otherwise, the query text can be selected, copied, and pasted elsewhere as needed.

When a Query Has Been Entered Manually

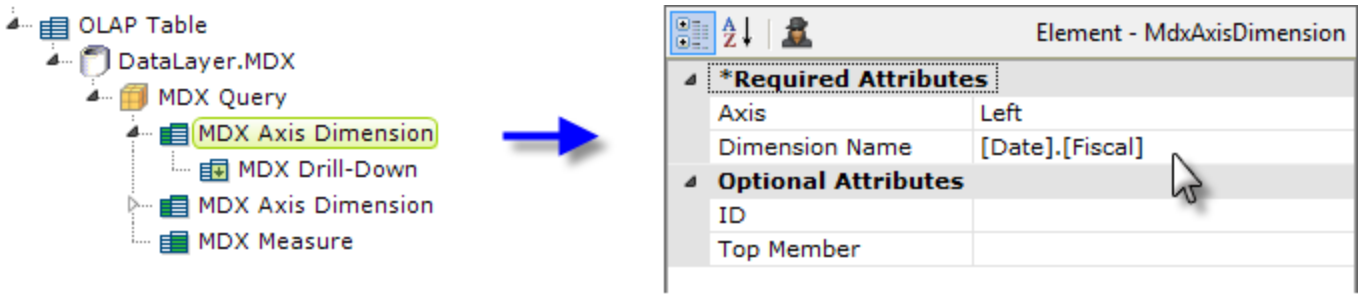
If you *have already entered a query manually* in the MDX Source attribute and you then open it in the MDX Query Builder, the tool will try to "reverse-engineer" the query. It succeeds at this most of the time but, if the query includes **tokens** or is created using a non-standard SQL language, the MDX Query Builder may fail and display an error message. This is especially likely when Logi tokens have been used outside of a SELECT statement.

DataLayer.MDX - Using MDX Query Elements

When working with an OLAP Table or OLAP Grid element, you can have the Logi Engine create the MDX query for you by using DataLayer.MDX and the **MDX Query** element.



The example above shows the datalayer with its MDX Query child element. The name of the target data cube must be provided, as shown.



Beneath the MDX Query element, you add child elements to define the dimensions, measures, and filters needed, as shown above. When the application runs, the Logi Engine will use these elements to create the appropriate query.

The available child elements, and their child elements, include:

Element	Description
MDX Axis Dimension	Specifies adimension that will be shown in the OLAP Gridor OLAP Table, across the top or left side. You may have any number of MDX Axis Dimension elements for either axis. Generally, the OLAP Table requires these elements, becausethe user has no way to add them at runtime. However, since the OLAP Grid allows user selection of dimensions, MDX Axis Dimensions need only be specified for "default" dimensions. There must be at least one dimension defined for the left axis to see any results. You may specify a hierarchy in the Dimension Name attribute, e.g. `[MyDimension].[MyHierarchy]`.
Child: MDX Drill-Down	Specifies a single member of the dimension to be drilled/expanded. You may add any number of MDX Drill-Down elements. Their Value attribute must be a member of the parent element's dimension. Separate each level with the period character, and when a member has spaces in the name, it must be enclosed in brackets. Begin the Value with the dimension name, e.g. `[Customers].[All Customers]`. When there are multiple dimensions on a single axis, the members of each axis must be separated by the asterisk character. For example: `[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.
Child: MDX Drill-Up	Used tocollapse a dimension member that has been drilled-down. This may be used to undo drill-downs caused by settingMDX Drill-Down to *AllMembers*.
Child: MDX Find	When present, automatically drills down and filters to show all members that contain the specified Value. *Available only when using the OLAP Grid element.*

Element	Description
Child: MDX Member Property	Specifies a single member property that will be displayed with the appropriate member values. Each member property is listed in its own column to the right of the Dimension Members. The column only appears when there is at least one member that has the property. Add an MDX Member Property element for each property to be listed. At runtime, users can change the properties when the OLAP Grid element's Pick Member Properties attribute is *True*. *Available only when using the OLAP Grid element.*
MDX Calculated Measure	Adds an additional measure based on an MDX formula. MDX formulae basically follow intrinsic functionsyntax 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 Dimension	Filters or slices values out of the OLAP cell set. Each dimension to be filtered should have an MDX Filter Dimension element. *Dimensions cannot be used in both this elementand MDX Axis Dimension elements at the same time*.
Child: MDX Filter Value	Specifies a single member to be included in, or excluded from, the cell calculations. You may have any number of MDX Filter Values for each dimension. The Value attribute must be a member of the MDX Filter Dimension element's dimension. Separate each level with the period character. When a member has spaces in the name, it must be enclosed in brackets. Example: `[USA].[CA]`
MDX Measure	Specifies a measure that will be shown in the OLAP Grid or OLAP Table cells. You may have any number of MDX Measure elements. If none of these elements are included, the cube's default measures will be shown. Cells are formatted according to the format specified in the cube definition. Localization is applied to these

Element	Description
	formats according to the browsers language setting.

The data retrieved into the datalayer can be viewed by turning on the **Debugging Link** in the _Settings definition (General element) and using the resulting link at the bottom of the report page to view the **Application Trace** page. A link on the Trace page will display the retrieved data.

DataLayer.Plugin

The **DataLayer.Plugin** element gives developers the ability to retrieve data from datasources not otherwise supported with standard connection and datalayer elements, using **custom-written** code. The code, known as a plug-in, is either a Windows .dll or a Java .javafile. The returned data is XML in a rowset format; @Data tokens can be used to retrieve the data for each column.

- Attributes
- Working with DataLayer.Plugin

Attributes

The DataLayer.Plugin element has the following attributes:

Attribute	Description
Assembly Name	(Required) The complete filename of the plug-in assembly, ending in ".dll" (.NET) or .jar (Java). No path information is required if the assembly file has been placed in a folder named _Plugins under the application folder. Otherwise, a fully-qualified path and filename for the assembly file may be provided. If the dll is not in the _Plugins folder, you have the ability to call plugins within the application directory by setting the path using `@Function.AppPhysicalPath~/` . Other tokens, such as @Request, @Sessions, and @Constant can also be used to call plugins within the application directory by setting the path.
Class Type Name	(Required for .NET; ignored in Java) Specifies the plug-in class that contains the method to be called. The value consists of the dll's root namespace, ".", and the class name. For example: `LogiXML.SamplePlugin.Plugin`
ID	An optional element ID. Recommended for easier identification when debugging.

Attribute	Description
Method	(Required) The name of the plug-in Method to be called.
Pass Data As	Specifies how data is returnedfrom the plug-in to the datalayer. When it's set to *XmlDocument* (the default value), the data will be passed to the datalayer as an XML document object in memory and added to the "ReturnedData" property of rdServerObjects. To reduce the amount of memory used when manipulating large datasets (100,000+ rows), this attribute can be set to *FileName*, and the datalayer will be passed the name of a cache file used by the plug-in, rather than the data itself. The plug-in can then use an XML stream writer to cache the data and the datalayer will use a stream reader to retrieve it. The filename includes the fully-qualified path and is placed in the rdServerObjects object's "ReturnedDataFile" properties.

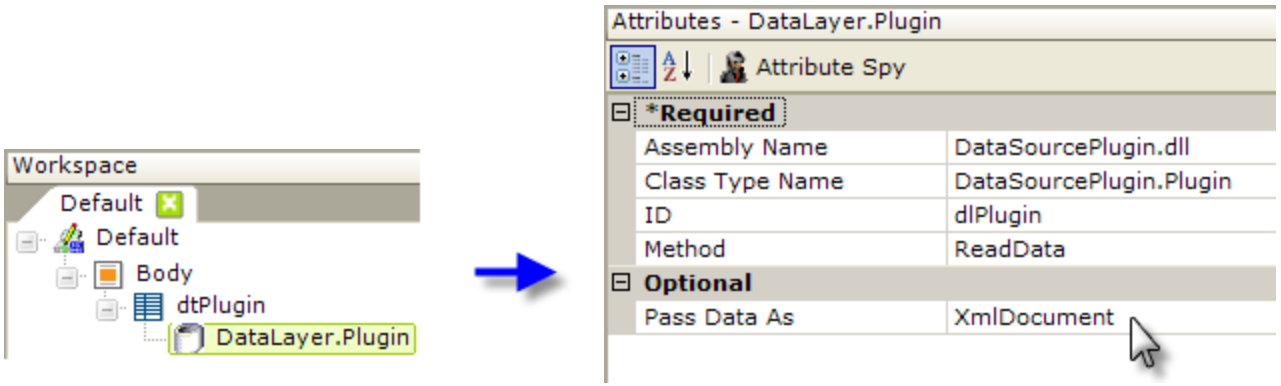
Working with DataLayer.Plugin

Instead of working with a datasource connection or with a file system file, as other datalayers do, DataLayer.Plugin retrieves its data using a custom-written plug-in. This allows access to datasources that are not already supported in Logi applications via the standard suite of connections and datalayers. Plug-ins written for this purpose should return an XML document object, or the name of a file containing the results as serialized string data.

Examples of this type of datasource include things like multi-pass web services (where your datalayer may need to make multiple trips back to the web service before your results can be generated) or a custom .NET or Java data object that interacts with another app or process.

Do not confuse this element with the **DataLayer Plugin Call** element. That element is designed to apply the operations of a plug-in on data *already retrieved* into a datalayer; this element actually uses a plug-in to *retrieve* the data.

The following example illustrates how to use the **DataLayer.Plugin** element:



Add a **DataLayer.Plugin** element to the definition, as shown above, and configure its attributes per the attribute table in the previous section. Remember that the **Class Type Name** attribute is *required* for .NET plug-ins, but is *ignored* for Java plug-ins.

Because the actual behavior of the plug-in is determined by developers outside of Logi Analytics, the **Add data columns** wizard found in Studio may not work if the plug-in does not return schema information. For information about writing a plug-in, see *Logi Plug-ins*.

DataLayer.XOLAP Query

The **DataLayer.XOLAP Query** element is used to predefine a data view based on a Logi XOLAP Cube. It creates the view that will be displayed by a Dimension Grid and can also be used with the OLAP Grid and OLAP Table elements.

For additional information, see *Logi XOLAP*.

About DataLayer.XOLAP Query

The DataLayer.XOLAP Query element behaves a little differently than the usual datalayer in that it does not connect to a data-source and retrieve data. Instead, it works with a Logi XOLAP cube that has already been populated with data (using other datalayers), creating a view of the data for use with other elements.

DataLayer.XOLAP Query is simply a parent element for elements that specify the dimensions that appear on the top and left axes of other elements such as the Dimensions Grid. The order in which dimensions are defined for an axis determines the order in which they are displayed. For example, if a dimension showing "customer city" is provided first, followed by a dimension showing "customer marital status", the user will be able to drill down to a particular city and then drill down on marital status to see measures for each marital status in that city.

The datalayer can also specify which measures will be displayed in the report and can include member properties and filters.

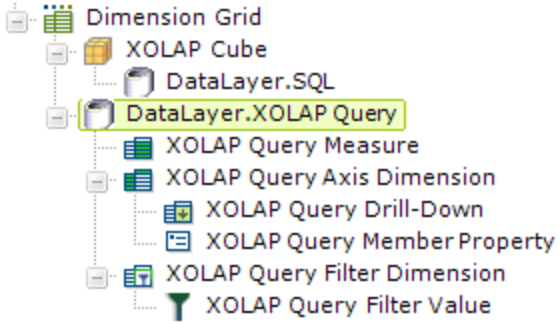
Attributes

The DataLayer.XOLAP Query element has no attributes other than an element **ID**.

Working with DataLayer.XOLAP Query

As mentioned earlier, this element does not retrieve data as most datalayers do. Logi XOLAP operations involve two data-related steps: first, the creation of a virtual data cube (which uses one of the other datalayer types), and second, reporting against that data cube, optionally using DataLayer.XOLAP Query.

If DataLayer.XOLAP Query is not used as part of the second step, then reporting takes place against *all* of the data in the cube. When it is used, then it defines a view of the data.



The example above shows a datalayer that retrieves data for the XOLAP cube and DataLayer.XOLAP Query, which is the parent element for a number of elements that can be used to define the data view. The possible child elements are:

Element	Description
XOLAP Query Measure	Defines which measures will be shown in the cells of a Dimension Grid. If no measure is specified, the first one defined in the XOLAP Cube is used. Measures are formatted according to the XOLAP Cube element's Format

Element	Description
	attribute.
XOLAP Query Axis Dimension	Defines a dimension of the XOLAP Cube that will appear in the report on the left or top axis.
XOLAP Query Drill-Down	Defines a level of a XOLAP Query dimension that will be expanded to show its child values. Each dimension shows the "All" level by default. A drill-down element causes a level of child values to be added to the report. The parent level of a drill-down can be included if desired but is not necessary. For example, a drill-down with a value of [Customers].[All Customers].[USA].[California] would cause the report to show all cities in California, whether there is a drill-down for states in USA or not. To define a drill-down on a second or lower dimension, separate each position with an asterisk, for example: [Product].[All Products]*[Customers].[All Customers].[USA]
XOLAP Query Member Property	This child element is added beneath an axis dimension to make a member property defined in the XOLAP Cube appear in the Dimension Grid when the corresponding level of that dimension is visible.
XOLAP Query Filter Dimension	Specifies a filter that will be applied to the default view of the Dimension Grid. In the grid's user interface, the filter's pop-up listing of checked and unchecked members only recognizes a Filter Type attribute value of *Exclude*.
XOLAP Query Filter Value	Specifies a single member to be included in, or excluded from, the cell calculations. Any number of these filters may be provided for each dimension. The element's Value attribute value must include members of the XOLAP Query Filter Dimension element's dimension and each level is separated with the period character. If a mem-

Element	Description
	ber has spaces in its name, it must be enclosed in square brackets: Example: [USA].[CA]

The data retrieved into the datalayer can be viewed by turning on the **Debugging Link** in the _Settings definition (General element) and using the resulting link at the bottom of the report page to view the **Application Trace** page. A link on the Trace page will display the retrieved data.

DataLayer.Excel

Developers often need to read data from Microsoft Excel spreadsheets and Logi Info includes a datalayer element just for this purpose, **DataLayer.Excel**. This topic discusses this datalayer.

Attributes

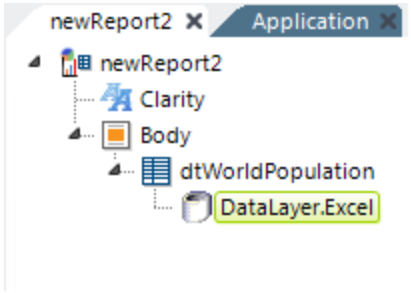
The **DataLayer.Excel** element has the following attributes:

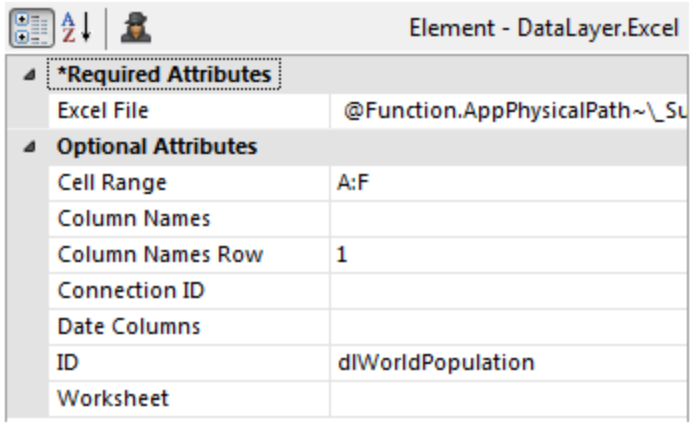
Attribute	Description
Excel File	(Required) Specifies the fully-qualified file system **path** and **filename** for the spreadsheet file. The token @Function.AppPhysicalPath~ can be used to get the application folder path. Both .xls and .xlsx formats are supported. Example: `@Function.AppPhysicalPath~\SampleData\Contacts.xls`A fully-qualified URL to an Excel file on the Internet can also be specified. Example: `http://www.example.com/data/records.xls`
Cell Range	Specifies a **range of cells** from the worksheet to be read. Multiple ranges, separated by commas, can be specified and will be joined together to include all matching columns and rows. Examples: `A1:H10` will include rows 1 through 10 of columns A through H. `C:F,H` will include all data rows from columns C, D, E, F, and H. `C5:E10,H` will include rows 5 through 10 of columns C, D, E, and H. `10:30` will include rows 10 through 30 of all data columns.

Attribute	Description
Column Names	If left blank and if the **Column Names Row** attribute has no value, columns retrieved into the datalayer will be assigned default names in the form: Column1, Column2, Column3, etc. If, however, an optional comma-separated list of **custom** datalayer **column names** is specified here, they will override the default names. Or, if the **Column Names Row** attribute specifies a row number, valid text names from the **specified data row** will automatically be assigned as the datalayer column names, overriding the default names.
Column Names Row	Specifies the **row number** which contains names for the datalayer columns. This row will not be included in the data, but will be used as column names when the data is valid (names should start with a letter and include only letters or numbers).
Connection ID	Specifies a connection to a data source that's defined in the _Settings definition.
Date Columns	Specifies a comma-separated list of column names of those columns that should be formatted as **DateTime** data. The names entered here should match those specified using earlier attributes for the **datalayercolumnnames**, e.g. these may be the default names: Column1, Column5, etc. or the custom names specified in the Column Names attribute, or the names read from the row specified in the Column Names Row attribute.
ID	Specifies an unique element ID. Recommended for easier identification when debugging.
Worksheet	Specifies the worksheet in the Excel file from which to read data. The default value is the *first worksheet*.

Working with DataLayer.Excel

In most respects, **DataLayer.Excel** functions exactly as other datalayer elements do and its data can be filtered and conditioned using appropriate elements. One major difference, however, is that *there is no need to use a Connection element in the _Settings definition with this element.* The DataLayer.Excel element directly accesses the file system to read .XLS and .XLSX files.





@Function.AppPhysicalPath~_SupportFiles\WorldPopulation.xlsx

As shown above, a **DataLayer.Excel** element is added as a child element to a Data Table. Its attributes are set so that it accesses the desired Excel file, reads the right worksheet and columns, and also handles column names as desired. The datalayer reads and caches the data from the Excel file. You can add child elements beneath the datalayer to affect its contents, including:

- **Filtering**: Sort, group, or restrict the data
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled data values

- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report definitions

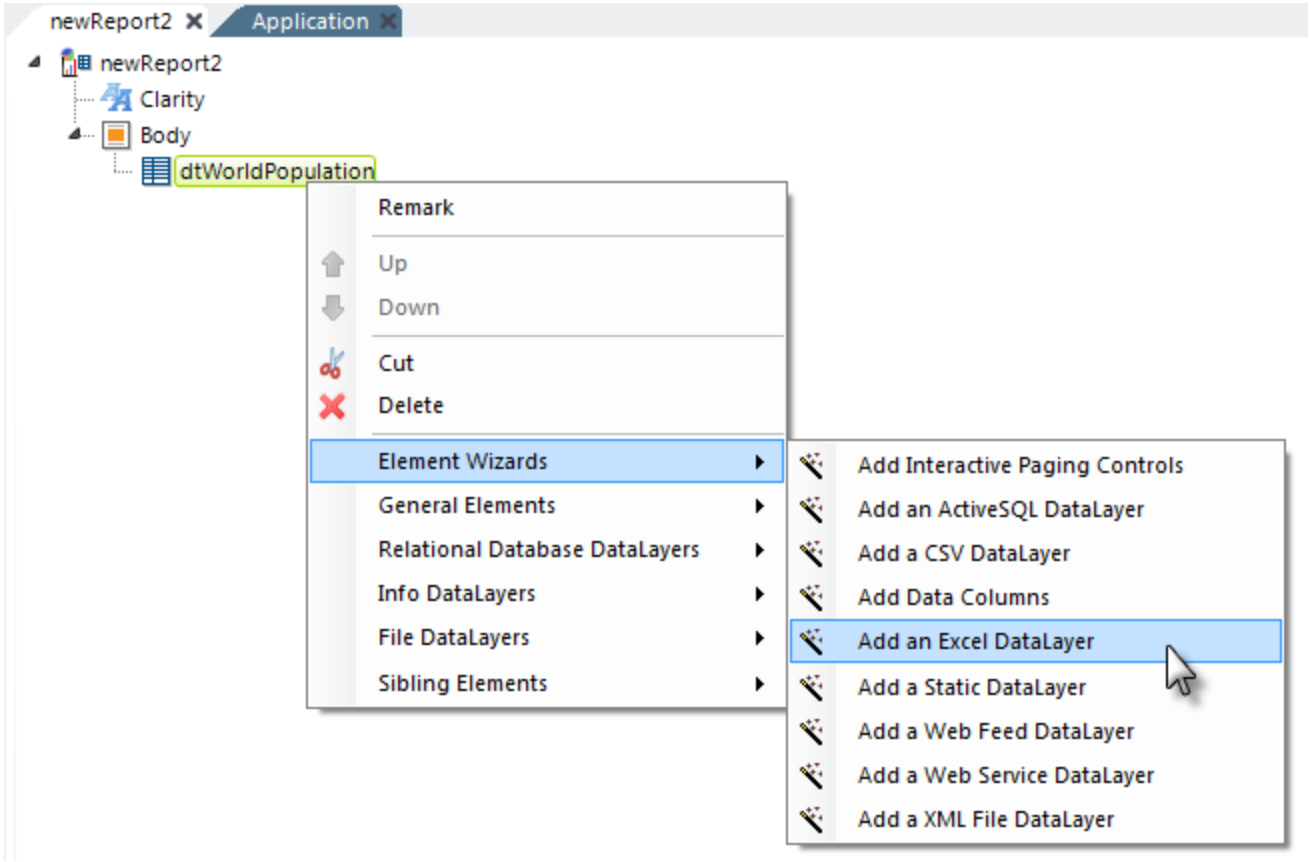
 If you have the target spreadsheet open in Excel, the datalayer may not be able to read it.

Data retrieved into the datalayer is cached in XML format, in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

The data read with the datalayer is available using @Data tokens, in the format @Data.*ColumnName*~. The spelling of the column name is **case-sensitive**. The data is only available within the **scope** of the parent element of the datalayer, not throughout the entire report definition. The **DataLayer.Linked** element can be used to make the data reusable in another datalayer outside this scope.

The *Auto Columns* element can be used to quickly display in your report all the data in a datalayer.

The data retrieved into the datalayer can be viewed by turning on the **Debugging Link** in your _Settings definition (**General** element) and using the resulting link at the bottom of your report page to view the **Debugger Trace** page. A link on the Trace page will display the retrieved data.



Studio includes a **wizard** that will add an Excel datalayer to your definition, letting you select columns and then configuring and inserting the element for you. You can launch the wizard using the right-click context menus shown above.

DataLayer.Google App

The Google App Engine lets companies run their web applications "in the cloud", on Google's application platform. App Engine applications are easy to build, easy to maintain, and easy to scale as traffic and data storage needs grow. The Logi **DataLayer.Google App** datalayer enables connection to the datastore of a published application hosted on the Google App Engine.

The following topics discuss this datalayer:

- DataLayer.Google App Attributes
- Adding Logi Python File to Your Google Application
- Working with DataLayer.Google App
- DataLayer.Google App Example

DataLayer.Google App - Attributes

The DataLayer.Google App element has the following attributes:

Attribute	Description
Google App Entity	(Required): The name of the data entity (also called "kind" or "model") desired. This value is case-sensitive.
Google App Module	(Required): The name of the Python source code in the Google App which contains the class defining the data entity specified in Google App Entity.
Google App URL	(Required): The web address of the Google App. Example: `http://mydemoapp.appspot.com`

Google App Filter	A filter that can be used to restrict the data that's returned. Examples:
	`State = 'Virginia'`
	`Level = 2 AND Country = 'France'`
	`Color IN ('red', 'blue')`

Google App ID Field	The name of a data model property that is unique for each entity. This field is used to build a complete set of data from multiple queries when the datastore is too large for a single query.
Google App Limit	The maximum number of entities to request per query, up to 1000. Reducing this limit can help if a query times out. Default and maximum value: *1000*.

Attribute	Description
ID	An optional, unique element ID. Recommended for easier identification when debugging.

DataLayer.Google App - Adding Logi Python File to Your Google Application

In order to enable your Google App Engine application to provide the appropriate response to the Logi datalayer element, you must include a **special Python file** in your Google application. Open this file and save it, using the file name "logi.py", to the root folder of your Google application. Your Logi application will call functions in this file, which run on Google, in order to retrieve data (the Logi Engine itself does not run Python scripts).

DataLayer.Google App - Working with DataLayer.Google App

Detailed information about the Google App Engine referenced here can be found at the Google App Engine web site.

Unlike most datalayers, **DataLayer.Google App**, does not require a separate connection with the Google web service. Therefore, there is no related Connection element that must be used.

The data retrieved with a datalayer is available using @Data **tokens**, in the format @Data.*ColumnName*~. The spelling of the column name is **case-sensitive**. The data is only available within the **scope** of the **parent** element of the datalayer, not throughout the entire report definition. The DataLayer.Linked element can be used to make the data reusable in another datalayer outside this scope.

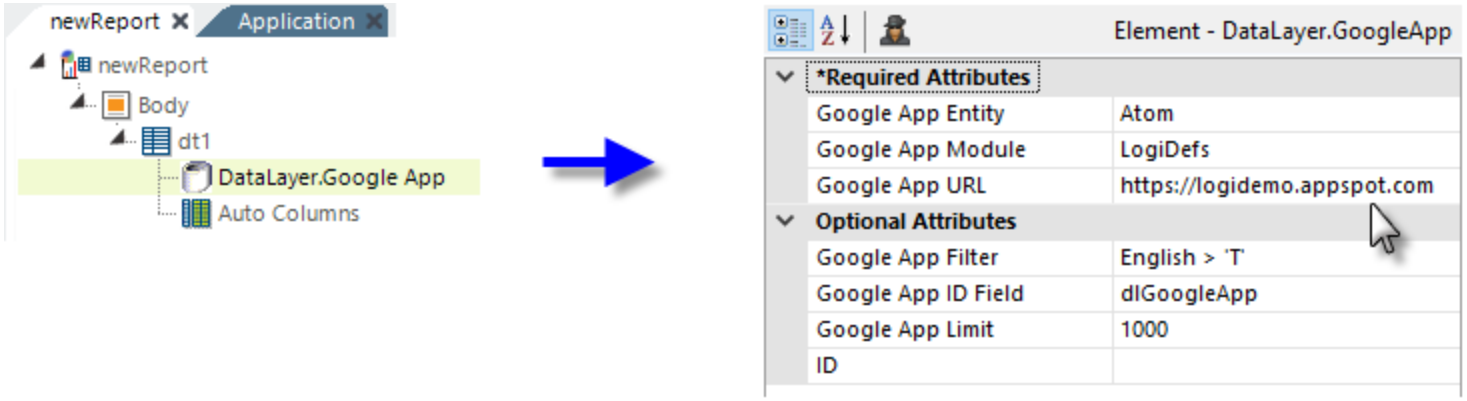
The *Auto Columns* element can be used to quickly see which "column names" are being returned by the web service.

You can also view the data retrieved into the datalayer by turning on the **Debugging Link** in their _settings definition (General element) and using the resulting link at the bottom of the report page to view the **Application Trace** page. A link on the Trace page will display the retrieved data.

 *Take appropriate security precautions and use the account facilities available in the Google App Engine service to ensure that your data is secured.*

DataLayer.Google App - DataLayer.Google App Example

The following example demonstrates how the **DataLayer.Google App** element can be used.



The example definition shown above includes a **Data Table** and a **DataLayer.Google App** element, configured with appropriate attribute settings.

key	atomicnum	Spanish	Turkish	symb	English	French	Italiano
agtsb2dpeG1sZGVtb3ILC	73	Tantalio	Tantal	Ta	Tantalum	Tantale	Tantalio
agtsb2dpeG1sZGVtb3ILC	43	Tecnecio	Tekhnetyum	Tc	Technetium	Technétium	Tecnezio
agtsb2dpeG1sZGVtb3ILC	52	Telurio	Tellür	Te	Tellurium	Tellure	Tellurio
agtsb2dpeG1sZGVtb3ILC	65	Terbio	Terbiyum	Tb	Terbium	Terbium	Terbio
agtsb2dpeG1sZGVtb3ILC	81	Talio	Talyum	Tl	Thallium	Thallium	Tallio
agtsb2dpeG1sZGVtb3ILC	90	Torio	Toryum	Th	Thorium	Thorium	Torio
agtsb2dpeG1sZGVtb3ILC	69	Tulio	Tulyum	Tm	Thulium	Thulium	Tulio
agtsb2dpeG1sZGVtb3ILC	50	Estaño	Kalay	Sn	Tin	Étain	Stagno
agtsb2dpeG1sZGVtb3ILC	22	Titanio	Titanyum	Ti	Titanium	Titane	Titanio
agtsb2dpeG1sZGVtb3ILC	74	Wolframio	Tungsten	W	Tungsten	Tungstène	Tungsteno
agtsb2dpeG1sZGVtb3ILC	112	Ununbio	Ununbiyum	Uub	Ununbium	Ununbium	Ununbio
agtsb2dpeG1sZGVtb3ILC	116	Ununhexio	Ununheksiyum	Uuh	Ununhexium	Ununhexium	Ununhexio
agtsb2dpeG1sZGVtb3ILC	118	Ununoctio	Ununoktiyum	Uuo	Ununoctium	Ununoctium	Ununoctio
agtsb2dpeG1sZGVtb3ILC	115	Ununpentio	Ununpentiyum	Uup	Ununpentium	Ununpentium	Ununpentio
agtsb2dpeG1sZGVtb3ILC	114	Ununquadio	Ununkuadyum	Uuq	Ununquadium	Ununquadium	Ununquadio
agtsb2dpeG1sZGVtb3ILC	117	Ununseptio	Ununseptiyum	Uus	Ununseptium	Ununseptium	Ununseptio
agtsb2dpeG1sZGVtb3ILC	113	Ununtrio	Ununtriyum	Uut	Ununtrium	Ununtrium	Ununtrio
agtsb2dpeG1sZGVtb3ILC	92	Uranio	Uranyum	U	Uranium	Uranium	Uranio
agtsb2dpeG1sZGVtb3ILC	23	Vanadio	Vanadyum	V	Vanadium	Vanadium	Vanadio
agtsb2dpeG1sZGVtb3ILC	54	Xenón	Ksenon	Xe	Xenon	Xénon	Xeno
agtsb2dpeG1sZGVtb3ILC	70	Iterbio	Itterbiyum	Yb	Ytterbium	Ytterbium	Itterbio
agtsb2dpeG1sZGVtb3ILC	39	Itrio	Itriyum	Y	Yttrium	Yttrium	Ittrio
agtsb2dpeG1sZGVtb3ILC	30	Zinc	Çinko	Zn	Zinc	Zinc	Zinco
agtsb2dpeG1sZGVtb3ILC	40	Circonio	Zirkon	Zr	Zirconium	Zirconium	Zirconio

The screenshot of the resulting **Logi application table**, above, shows the data from the worksheet being used in the sample application.

DataLayer.Google Spreadsheet

Google Docs is a free, web-based word processor, spreadsheet, and presentation application offered as a web service by Google. The **DataLayer.Google Spreadsheet** element allows developers to connect to Google Docs and retrieve data from spreadsheets stored there; the data can then be used in Logi applications.

The following topics introduce the DataLayer.Google Spreadsheet element:

DataLayer.Google Spreadsheet - Attributes

The **DataLayer.Google Spreadsheet** element has the following attributes:

Attribute	Description
ConnectionID	(Required) Specifies the element ID of a **Connection.GoogleDocs** element, in the _Settings definition. Valid Google Account credentials must be provided in the connection element.
Google SpreadsheetName	(Required) Specifies the case-insensitive name of the desired **spreadsheet** document stored on Google Docs. If the value "List spreadsheets" is entered here, a list of the spreadsheet documents associated with the Google Account will be returned. The names of the columns returned to the datalayer will be *Name*, *Author*, and *Updated* (a timestamp). You may want to temporarily use an AutoColumns element to display the list of spreadsheets.
GoogleWorksheetName	(Required) Specifies the case-insensitive name of the desired **worksheet** within the spreadsheet document stored on Google Docs. If the name of the spreadsheet specified earlier is valid, and the value "List worksheets" is entered here, a list of the worksheets associated with the spreadsheet will be returned. The names of the columns returned to the datalayer will be *Name*, *Updated* (a timestamp), and *Rows* (the row count). You may want to temporarily use an AutoColumns element to display the list of workheets.
ID	Specifies an optional element ID. Recommended for easier identification when debugging.
GoogleDateColumns	Specifies a comma-delimited list of the worksheet columns, if any, that should be formatted as *dates*.

Attribute	Description
GoogleNumericColumns	Specifies a comma-delimited list of the worksheet columns, if any, that should be formatted as *numbers*.
Google Spread-sheetCulture	Specifies the culture value for the spreadsheet document, provided here to assist in interpreting international data formats. Default: *invariant (en US)*

DataLayer.Google Spreadsheet - Getting a Google API Key

In order to use the DataLayer.Google Spreadsheet in your application, you'll need a Google API key for the *Sheets API*.

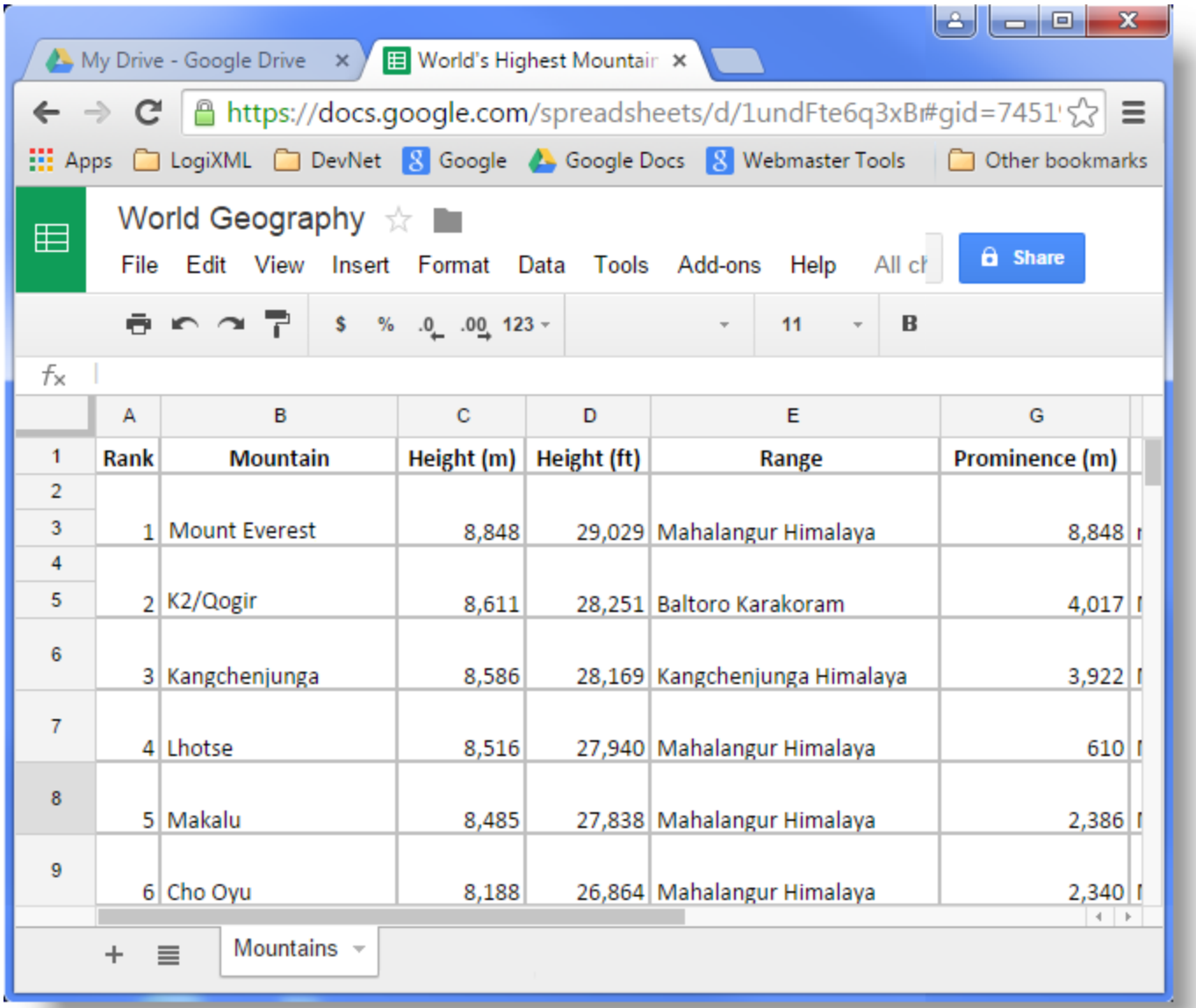
 In July 2018, Google changed its licensing scheme, ending the free API access previously offered at some usage levels. This specifically affected *geocoding* in Logi applications. Information about how to get a Google Sheets API key is available in *Google Connections*. Please read that topic and take the necessary steps before proceeding to use data from Google Spreadsheets in your Logi application.

DataLayer.Google Spreadsheet - Working with DataLayer.Google Spreadsheet

The datalayer uses a special connection element, **Connection.GoogleDocs**, to establish the connection with the Google web service. In order to work with this datalayer, worksheets must be formatted to have non-numeric column headers in the first row. These headers (after spaces and punctuation, except hyphens and periods, are removed and the name is converted to lowercase) become the column names of the data retrieved into the datalayer. Worksheet data is retrieved from the web service and cached as *rows* and *columns*, one data row per worksheet row. Row 1, as discussed in the previous paragraph, is not read as data. Data retrieved into the datalayer is cached in XML format. The data retrieved with a datalayer is available using @Data tokens, in the format @Data.*ColumnName*~. The spelling of the column name is *case-sensitive*. The data is only available within the scope of the parent element of the datalayer, not throughout the entire report definition. The DataLayer.Linked element can be used to make the data reusable in another datalayer outside this scope. Logi Info developers can make use of the **AutoColumns** element to quickly see which "column names" are being returned by the web service. We do not recommend the use of AutoColumns for production deployments. Developers can also view the data retrieved into the datalayer by turning on the **Debugging Link** in their _Settings definition (General element) and using the resulting link at the bottom of the report page to view the Debugger Trace page. A link on the Trace page will display the retrieved data.

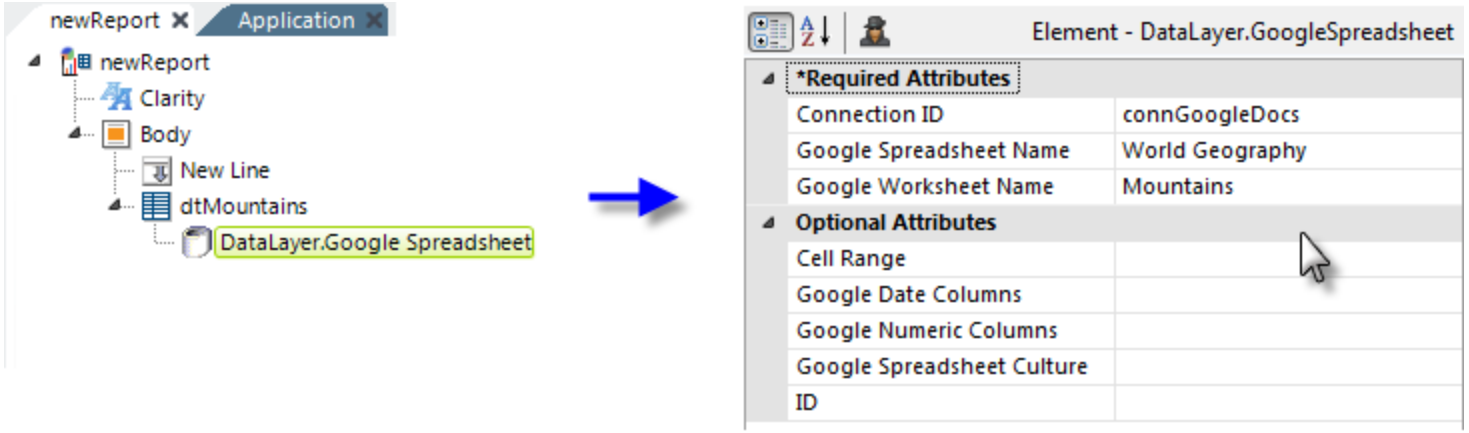
DataLayer.Google Spreadsheet - DataLayer.Google Spreadsheet Example

The following example demonstrates how the DataLayer.Google Spreadsheet element can be used.

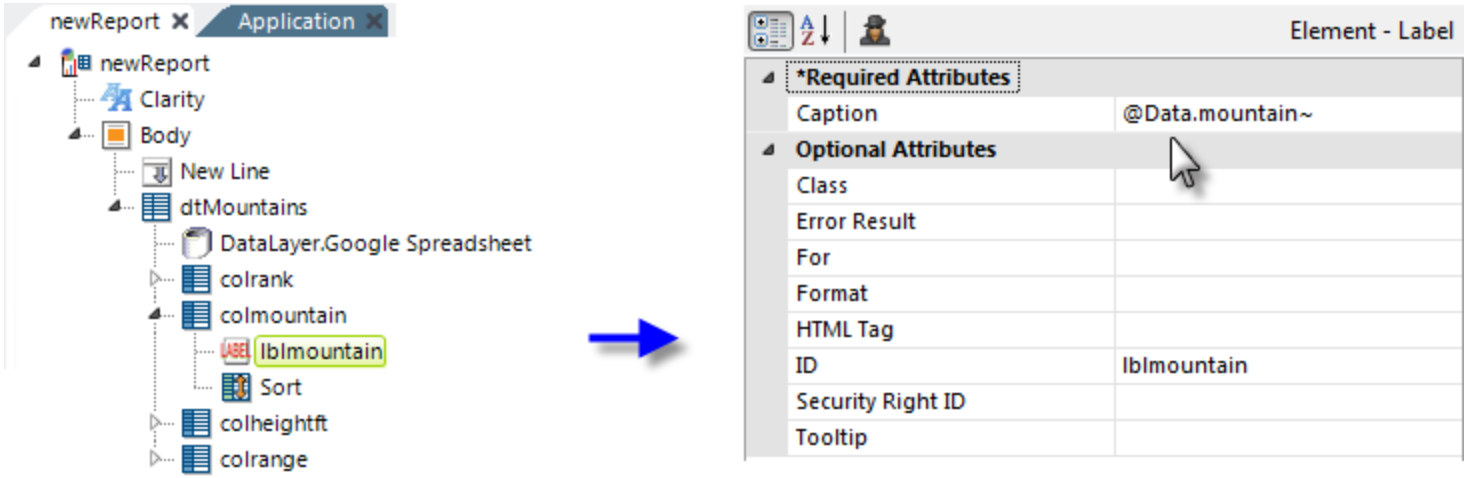


Rank	Mountain	Height (m)	Height (ft)	Range	Prominence (m)
1	Mount Everest	8,848	29,029	Mahalangur Himalaya	8,848
2	K2/Qogir	8,611	28,251	Baltoro Karakoram	4,017
3	Kangchenjunga	8,586	28,169	Kangchenjunga Himalaya	3,922
4	Lhotse	8,516	27,940	Mahalangur Himalaya	610
5	Makalu	8,485	27,838	Mahalangur Himalaya	2,386
6	Cho Oyu	8,188	26,864	Mahalangur Himalaya	2,340

The screenshot above shows the first few rows of the "Mountains" worksheet in the "World Geography" spreadsheet, stored on Google Docs in a Logi Analytics demo account.



The example definition shown above includes a **Data Table** and a **DataLayer.Google Spreadsheet** element. The datalayer's attributes are set as shown.



As shown above, **Label** elements within each **Data Table Column** use tokens to display the data retrieved into the datalayer. Note the correspondence between the worksheet **column header text** and the **column name** used with the @Data token in the definition (and that the text has been made all lower-case).

rank	mountain	heightft	range
1	Mount Everest	29,029	Mahalangur Himalaya
2	K2/Qogir	28,251	Baltoro Karakoram
3	Kangchenjunga	28,169	Kangchenjunga Himalaya
4	Lhotse	27,940	Mahalangur Himalaya
5	Makalu	27,838	Mahalangur Himalaya
6	Cho Oyu	26,864	Mahalangur Himalaya
7	Dhaulagiri I	26,795	Dhaulagiri Himalaya
8	Manaslu	26,781	Manaslu Himalaya
9	Nanga Parbat	26,660	Nanga Parbat Himalaya

The screenshot of the resulting Logi application Data Table, above, shows the data from the worksheet being used in the sample application.

DataLayer.Mongo Find

MongoDB is a cross-platform, document-oriented database system, classified as a "NoSQL" database. Rather than using a traditional table-based, relational DB structure, it uses a collection of JSON-like documents with variable schemas. The **DataLayer.Mongo Find** element retrieves data from this system, using the MongoDB Find API. This topic discusses this datalayer.

 General information about MongoDB can be found at the official website.

Attributes

The DataLayer.Mongo Find element has the following attributes:

Attribute	Description
Connection ID	(Required) Specifies the ID of a Connection.MongoDB element in the application's _Settings definition. *This element is discussed in more detail in the next section.*
Mongo Collection	(Required) Specifies the name of the document collection to open.
ID	An optional element ID. Recommended for easier identification when debugging.
Maximum Rows	Specifies a the maximum number of data rows to retrieve. If left blank, then all records that satisfy the request will be retrieved.

Attribute	Description

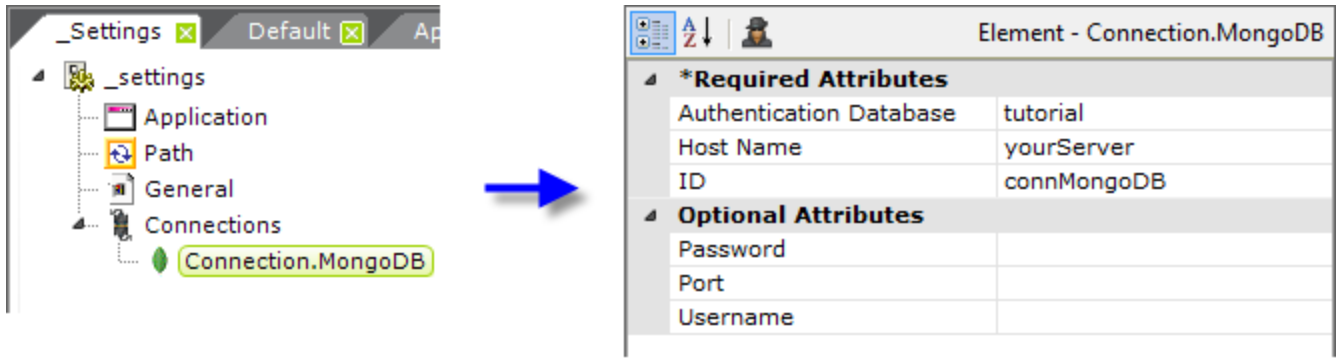
Mongo Fields Document	Specifies a Mongo "fields" document to identify the values returned from a query. For example:
	`{`
	`name : 1,`
	`phone_office : 1`
	`}`To vary the document content based on user or other input, use tokens, such as @Request, inside the document.

Mongo Query Document	Specifies a Mongo "find" document to identify one or more documents. Examples:
	Find all documents with "type" equal to "snacks".
	`{ type: "snacks" }`
	
	Find all documents with "type" equal to "food" or "snacks"
	`{ type: { $in: [ 'food', 'snacks' ] } }`
	To vary the document content based on user or other input, use tokens, such as @Request, inside the document.

Mongo Sort Document	Specifies the sort order for the results of a DataLayer.Mongo Find query. For example:
	
	`{`
	`country_name : 1,`
	`city_population : -1`
	`}`To vary the document content based on user or other input, use tokens, such as @Request, inside the document.

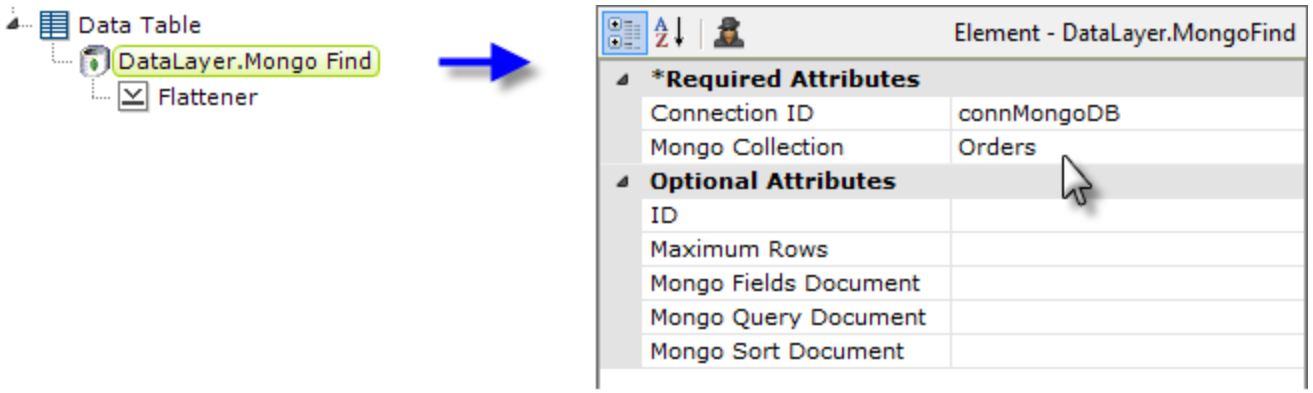
Working with DataLayer.Mongo Find

DataLayer.Mongo Find retrieves data from the MongoDB system. Its attributes can be used to form a "query" so that specific data is returned. It also supports a cursor, which allows results sets to be greater than 16MB in size. In many respects, DataLayer.Mongo Find functions exactly as other datalayer elements do and, once retrieved, its data can be filtered and conditioned using appropriate child elements. Like other datalayers, it works with a specific Connection element, **Connection.MongoDB**, to connect to the datasource.



Note that the Authentication Database name is case-sensitive!

As shown above, the **Connection.MongoDB** element is added to the **_Settings** definition, beneath the Connections element. Attributes are set appropriately for the MongoDB datasource. A **Mongo Params** element is available for use beneath Connection.MongoDB to supply additional options when connecting to the MongoDB database. Examples of additional parameters include "connectTimeoutMS" and "maxPoolSize". Refer to the MongoDB connection options documentation for a the complete list of options.



Next, in the report definition, a **DataLayer.Mongo Find** element is added as a child element of a Data Table or another data container element. Its attributes are set so that it accesses the desired MongoDB connection and data collection.

MongoDB results are often hierarchical and may consist of multiple levels of parent-child relationships. In order to work with Logi elements, the data often must be "tabularized" and this can be done by using *The Flattener* element.

The datalayer **reads** and **caches** the data from the datasource. You can add standard **child elements** beneath the datalayer to affect its contents, including:

- **Filtering**: Sort, group, or restrict the data
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled data values
- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report definitions

Data retrieved into the datalayer is cached in **XML** format, in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

The data read with the datalayer is available using **@Datatokens**, in the format @Data.*ColumnName*~. The spelling of the column name is **case-sensitive**. The data is only available within the **scope** of the **parent** element of the datalayer, not throughout the entire report definition. The **DataLayer.Linked** element can be used to make the data reusable in another datalayer outside this scope.

The data retrieved into the datalayer can be viewed by turning on the **Debugging Link** in your _Settings definition (**General** element) and using the resulting link at the bottom of your report page to view the **Debugger Trace** page. A link on the Debug page will display the retrieved data.

DataLayer.Mongo Map Reduce

MongoDB is a cross-platform, document-oriented database system, classified as a "NoSQL" database. Rather than using a traditional table-based, relational DB structure, it uses a collection of JSON-like documents with variable schemas. The **DataLayer.Mongo Map Reduce** element runs a Mongo Map Reduce command designed to return one or more documents.

General information about MongoDB can be found at the official website.

Attributes

The DataLayer.Mongo Map Reduce element has the following attributes:

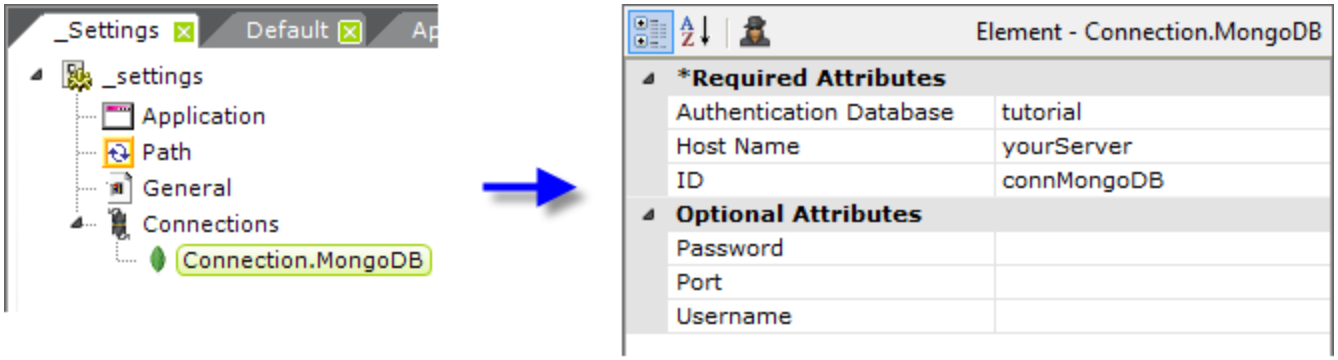
Attribute	Description
Connection ID	(Required) Specifies the ID of a Connection.MongoDB element in the application's _Settings definition. *This element is discussed in more detail in the next section.*
ID	(Required) A unique element ID.
Map Function	(Required) In MongoDB, map-reduce operations use custom JavaScript functions to *map*, or associate, values to a key. This attribute specifies that function. For example: `function() { emit(this.cust_id, this.price); };`
Mongo Col-	(Required) Specifies the name of the document collection to open. The collection name is case-sensitive!

Attribute	Description
lection	
Reduce Function	(Required) If the Map Function produces a key with multiple values mapped to it, the custom JavaScript function specified here *reduces*, or aggregates, the values for the key to a single object. For example: `function (keyCustId, valuesPrices) { return Array.sum(valuesPrices); }`
Finalize Function	The output of the Reduce Function may pass through a *finalize* function. specified here, to further condense or process the results of the aggregation. For example: `function (key, reducedVal) { reducedVal.avg = reducedVal.qty/reducedVal.count; return reducedVal; }`
Maximum Rows	Specifies a the maximum number of data rows to retrieve. If left blank, then all records that satisfy the request will be retrieved (up to a maximum of 16MB).

Mongo Query Document	Specifies a Mongo "find" document to identify one or more documents. Examples:
	Find all documents with "type" equal to "snacks".
	`{ type: "snacks" }`
	
	Find all documents with "type" equal to "food" or "snacks"
	`{ type: { $in: [ 'food', 'snacks' ] } }`
	To vary the document content based on user or other input, use tokens, such as @Request, inside the document.

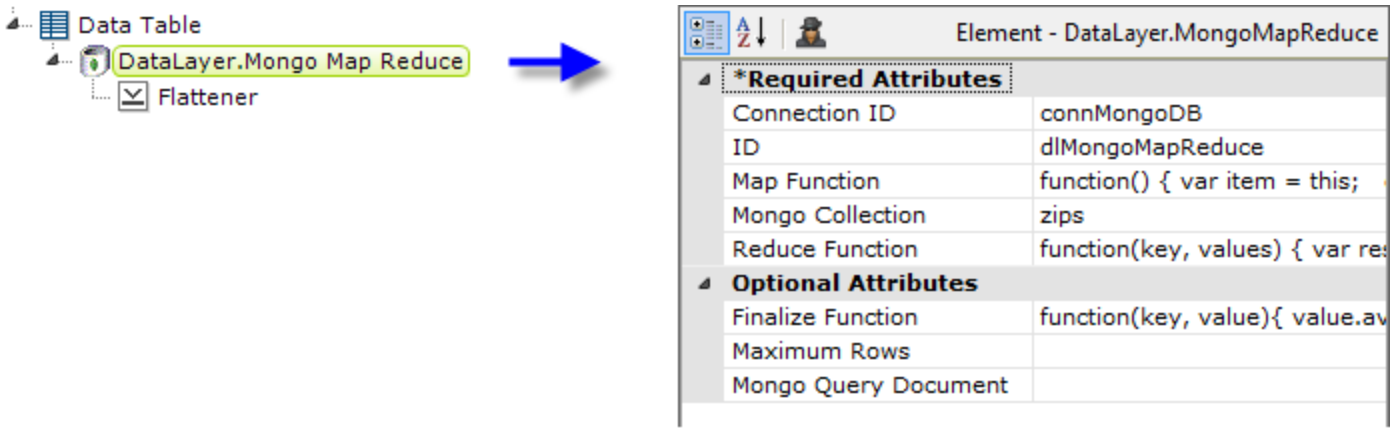
Working with DataLayer.Mongo Map Reduce

The Mongo map-reduce operation is used to condense large volumes of data into useful aggregated results, and this element incorporates that functionality into its data retrieval process. In many respects, DataLayer.Mongo Map Reduce functions exactly as other datalayer elements do and, once retrieved, its data can be filtered and conditioned using appropriate child elements. Like other datalayers, it works with a specific Connection element, **Connection.MongoDB**, to connect to the datasource.



Note that the Authentication Database name is case-sensitive!

As shown above, the **Connection.MongoDB** element is added to the **_Settings** definition, beneath the Connections element. Attributes are set appropriately for the MongoDB datasource. A **Mongo Params** element is available for use beneath Connection.MongoDB to supply additional options when connecting to the MongoDB database. Examples of additional parameters include "connectTimeoutMS" and "maxPoolSize". Refer to the MongoDB connection options documentation for a the complete list of options.



Next, in the report definition, a **DataLayer.Mongo Map Reduce** element is added as a child element of a Data Table or another data container element. Its attributes are set so that it accesses the desired MongoDB connection and data collection.

MongoDB results are often hierarchical and may consist of multiple levels of parent-child relationships. In order to work with Logi elements, the data often must be "tabularized" and this can be done by using *The Flattener* element.

The datalayer **reads** and **caches** the data from the datasource. You can add standard **child elements** beneath the datalayer to affect its contents, including:

- **Filtering**: Sort, group, or restrict the data
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled data values
- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report definitions

Data read into the datalayer is cached in XML format in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

The data read with the datalayer is available using **@Datatokens**, in the format @Data.*ColumnName*~. The spelling of the column name is **case-sensitive**. The data is only available within the **scope** of the **parent** element of the datalayer, not throughout the entire report definition. The **DataLayer.Linked** element can be used to make the data reusable in another datalayer outside this scope.

The data retrieved into the datalayer can be viewed by turning on the **Debugging Link** in your _Settings definition (**General** element) and using the resulting link at the bottom of your report page to view the **Debugger Trace** page. A link on the Debug page will display the retrieved data.

DataLayer.Mongo Run Command

MongoDB is a cross-platform, document-oriented database system, classified as a "NoSQL" database. Rather than using a traditional table-based, relational DB structure, it uses a collection of JSON-like documents with variable schemas. The **DataLayer.Mongo Run Command** element runs a Mongo command designed to return one or more documents.

- Attributes
- Working with DataLayer.Mongo Run Command

 General information about MongoDB can be found at the official website.

Attributes

The DataLayer.Mongo Run Command element has the following attributes:

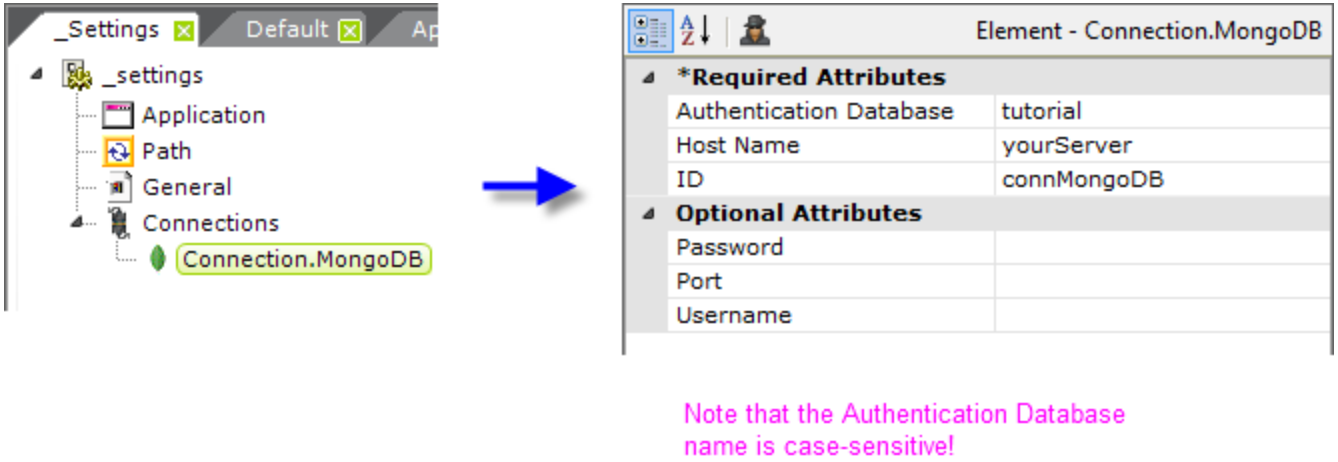
Attribute	Description
Connection ID	(Required) Specifies the ID of a Connection.MongoDB element in the application's _Settings definition. *This element is discussed in more detail in the next section.*
Mongo Run Command	(Required) Specifies a MongoDB command document. To vary the document content based on user or other input, use tokens, such as @Request, inside the document.
ID	A unique element ID.
Maximum Rows	Specifies a the maximum number of data rows to retrieve. If left blank, then all records that satisfy the request will be retrieved (up to a maximum of 16MB).

Attribute	Description
Preserve Hierarchy	Specifies whether or not to process the data to create traditional rows and columns. Set to *True* (the default value) to leave the data as it is, or *False* to process it.

Working with DataLayer.Mongo Run Command

DataLayer.Mongo Run Command runs a Mongo command designed to return one or more documents and is suitable for running aggregations with the Aggregation Pipeline, a simpler alternative to using map-reduce operations. Mongo Run Command has a result set size limit of 16MB.

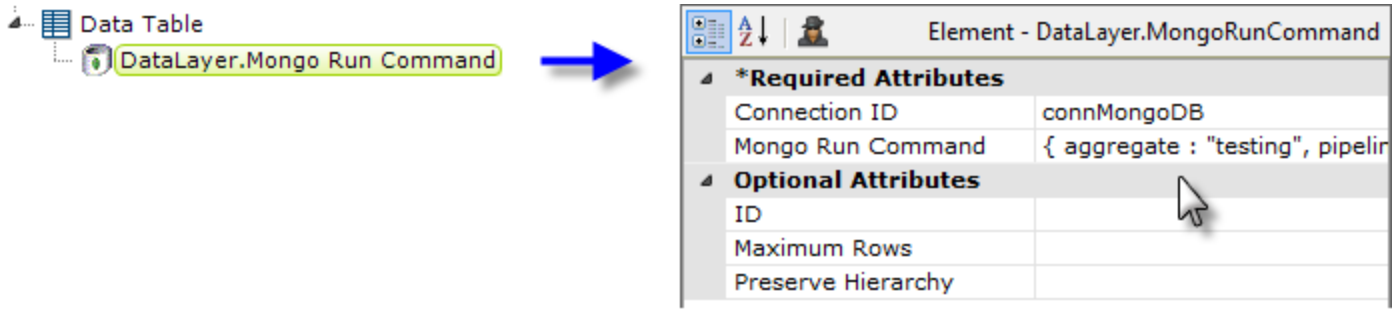
In many respects, DataLayer.Mongo Run Command functions exactly as other datalayer elements do and, once retrieved, its data can be filtered and conditioned using appropriate child elements. Like other datalayers, it works with a specific Connection element, **Connection.MongoDB**, to connect to the datasource.



Note that the Authentication Database name is case-sensitive!

As shown above, the **Connection.MongoDB** element is added to the **_Settings** definition, beneath the Connections element. Attributes are set appropriately for the MongoDB datasource.

A **Mongo Params** element is available for use beneath Connection.MongoDB to supply additional options when connecting to the MongoDB database. Examples of additional parameters include "connectTimeoutMS" and "maxPoolSize". Refer to the MongoDB connection options documentation for a the complete list of options.



Next, in the report definition, a **DataLayer.Mongo Run Command** element is added as a child element of a Data Table or another data container element. Its attributes are set so that it issues the desired MongoDB database command. This operation has a result size limit of 16MB.

MongoDB results are often hierarchical and may consist of multiple levels of parent-child relationships. In order to work with Logi elements, the data often must be "tabularized" and this can be done by using *The Flattener* element.

The datalayer **reads** and **caches** the data from the datasource. You can add standard **child elements** beneath the datalayer to affect its contents, including:

- **Filtering**: Sort, group, or restrict the data
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled data values

- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report definitions

Data retrieved into the datalayer is cached in **XML** format, in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

The data read with the datalayer is available using **@Datatokens**, in the format @Data.*ColumnName*~. The spelling of the column name is **case-sensitive**. The data is only available within the **scope** of the **parent** element of the datalayer, not throughout the entire report definition. The **DataLayer.Linked** element can be used to make the data reusable in another datalayer outside this scope.

The data retrieved into the datalayer can be viewed by turning on the **Debugging Link** in your _Settings definition (**General** element) and using the resulting link at the bottom of your report page to view the **Debugger Trace** page. A link on the Debugpage will display the retrieved data.

DataLayer.Twitter

The **DataLayer.Twitter** element searches the Twitter web service for "tweets" and messages based on the search criteria specified. There are multiple search criteria available for this search and the datalayer returns a dataset with the results.

- Attributes
- Working with DataLayer.Twitter

This element is only available in Logi Info .NET applications; it is *not available* in for Java applications.

Attributes

The DataLayer.Twitter element has the following attributes:

Attribute	Description
Connection ID	(Required) Specifies the ID of a Connection.Twitter connection element configured in the _Settings definition. If this value is left blank, the datalayer will use the first connection found in the _Settings definition. We recommend you enter an ID here in all cases.
Search Type	(Required) Specifies the type of search to be performed on the Twitter site. Options include: • User Statuses • Friend Statuses • User's Inbox • User's Sent Items

Attribute	Description
	• Keyword Search
ID	Specifies an optional, unique element ID. Recommended for easier identification when debugging.
Keyword	This attribute is only used when the Search Type attribute is set to "Keyword Search". The datalayer will then limit the tweets it retrieves to those including this keyword.
Maximum Rows	Specifies the maximum number of results rows to retrieve. Default: *unlimited*

Working with DataLayer.Twitter

This datalayer works with a **Connection.Twitter** datasource connection element, which must be added to the _Settings definition and properly configured.

The datalayer retrieves and caches the results returned from Twitter. You can add child elements beneath the datalayer to affect the results, including:

- **Filtering**: Sort, group, or restrict the result data
- **Joining**: Apply SQL JOINs to the data in the datalayer
- **Extending**: Add virtual columns to the datalayer that contain aggregated, calculated, or totaled result values
- **Securing**: Limit access to the data using Logi security
- **Linking**: Make the results reusable elsewhere in your report defintions

Data retrieved into the datalayer is cached in XML format, in memory and/or on the web server's file system. The latter is discussed in *The Logi Server Engine* and may be of interest to developers working with extremely large datasets or large numbers of concurrent users.

The data retrieved with a datalayer is available using @Data tokens, in the format @Data.*ColumnName*~. The spelling of the column name is case-sensitive. The data is only available within the scope of the parent element of the datalayer, not throughout the entire report definition. The **DataLayer.Linked** element can be used to make the data reusable in another datalayer outside this scope.

DateTime-type data retrieved into the datalayer is automatically reformatted to ISO 8601 format ("2009-06-23T09:32:11-04:00").

The *Auto Columns* element can be used to quickly display in your report all the data in a datalayer; this is particularly useful during development.

The valid "column names" retrieved and available through the @Data token vary depending on the type of search conducted. These are identified in the Twitter API documentation and may change with different API releases. The datalayer retrieves *all* of the columns made available by the API for each search type.

The data retrieved into the datalayer can be viewed by turning on the Debugging Link in your _Settings definition (General element) and using the resulting link at the bottom of your report page to view the Debugger Trace page. A link on the Trace page will display the retrieved data.

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.