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material may assist with understanding historical functionality, please be aware that the software described is no longer maintained at this version level and may contain outdated or inaccurate information.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

Using Logi Studio

Our primary development tool, **Logi Studio**, is used by developers to create Logi applications. It's a powerful development tool with many useful features and this topic provides specific instructions for using it.

The following topics provide more details about using Logi Studio:

About Logi Studio

Logi Studio is a stand-alone **Windows .NET application** that provides a comprehensive development environment for creating Logi Info web applications. It's capable of producing applications that run in .NET and Java environments. A typical scenario for Java developers is to use Studio on a Windows system to develop and test their applications, and then deploy them to their Linux/UNIX or Windows servers to be run under Java libraries.

Studio provides a number of features that make development *easy* and *fast*. It manages your project files and includes editors for every type of file that might be included in a web application, except images. Intelligent-completion features reduce the number of keystrokes necessary and potential typos made. You can develop and test your application right inside Studio.

This topic provides all the information necessary to become successful with Logi Studio. New users should read it carefully to get a full understanding of the tools Studio offers and how Logi applications are developed.

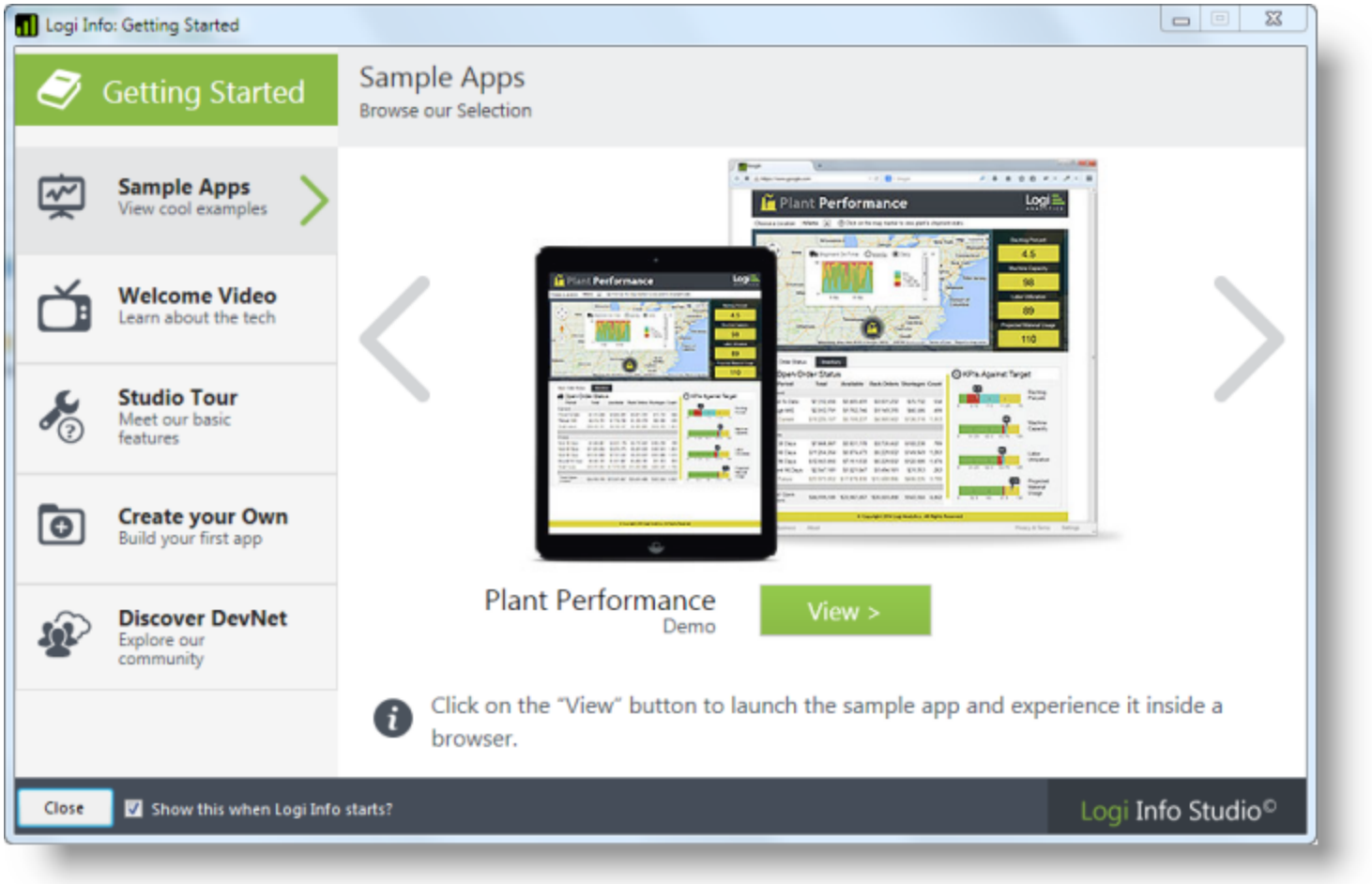
This version of Logi Studio includes these important changes:

- A *Filter Definitions* control allows you to filter definitions, in all categories at once, in the Application Panel.
- The `Consolas` font is now used in the Definition Source editor, the Support file editors, the Attributes panel, and the Attribute Zoom window.
- The Search feature provides additional filters, such as *Exclude Remarked Elements*, the ability to search within search results, a new results format, and more improvements.

Getting Started

This topic introduces the process of getting started and intro features in Logi Studio 12.

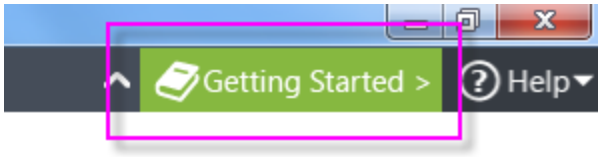
When Logi Studio starts, it presents a **Getting Started** dialog box:



A number of features are available to introduce you to Logi Info, including:

Feature	Description
Sample Apps	Sample applications built with Logi Info can be launched using the View button. They'll open in your default browser and provide some interesting examples of the possibilities available.
Welcome Video	This short video gives you an introduction to the technology behind a Logi Info application, and shows you how they're constructed. Key concepts and features are explained.
Studio Tour	An interactive image of Studio helps you discover its geography, terminology, and features. Hover your cursor over the numbers in the images for descriptive text.
Create Your Own	This step-by-step tutorial, in conjunction with an included tutorial application, guides you through creating your first reports, including a Data Table, a Bar Chart, an Analysis Grid, and a Dashboard.
Discover DevNet	Links are provided into useful topics and features in our Developer Network website.

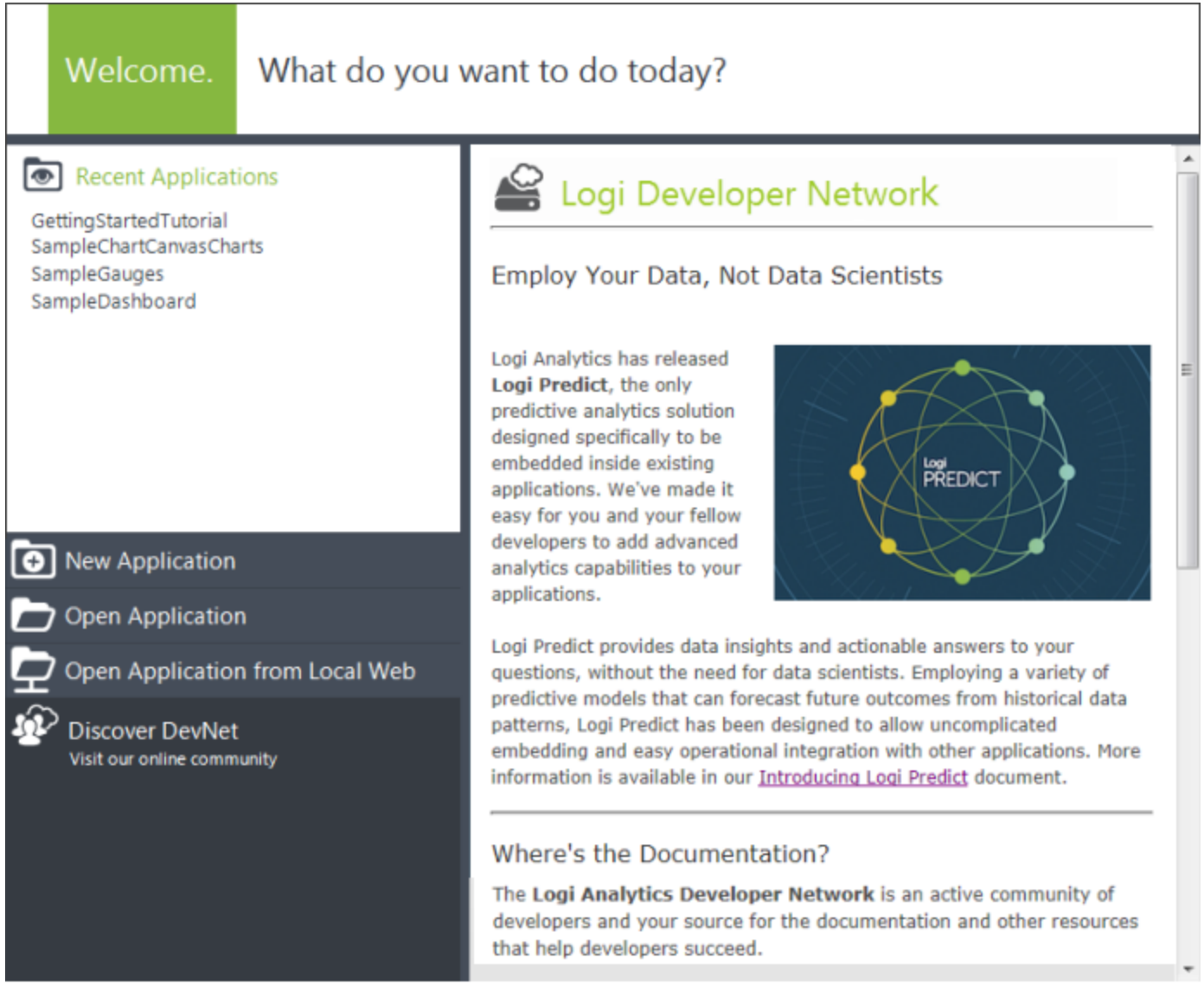
You can disable the dialog box, so that it no longer appears when Studio starts, by clearing the check box at the bottom.



You can always redisplay it by clicking the **Getting Started** button, shown above, at the top-right of the Studio's main menu. Its state can also be set directly in Tools → Options.

The Welcome Panel

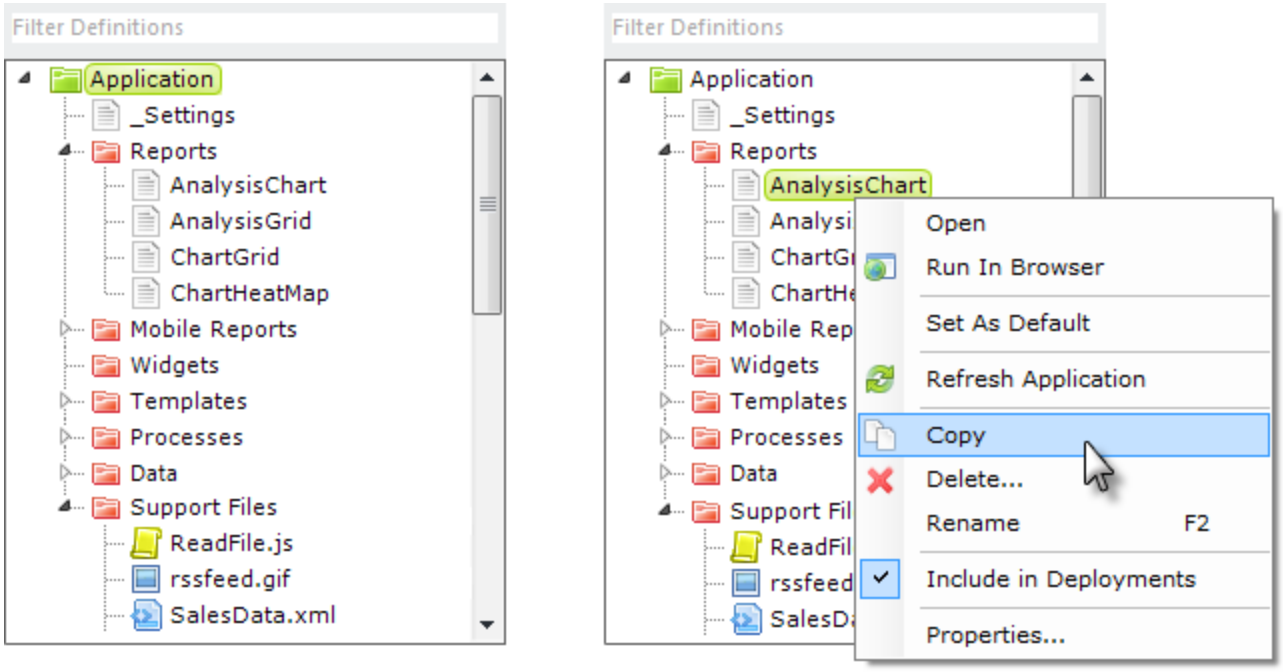
When the Getting Started dialog box is hidden or moved aside, you'll see the **Welcome panel** when you start Studio and whenever there's no application open.


Welcome.
What do you want to do today?
Recent Applications
GettingStartedTutorial
SampleChartCanvasCharts
SampleGauges
SampleDashboard
New Application
Open Application
Open Application from Local Web
Discover DevNet
Visit our online community
Logi Developer Network
Employ Your Data, Not Data Scientists
Logi Analytics has released Logi Predict, the only predictive analytics solution designed specifically to be embedded inside existing applications. We've made it easy for you and your fellow developers to add advanced analytics capabilities to your applications.
Logi PREDICT
Logi Predict provides data insights and actionable answers to your questions, without the need for data scientists. Employing a variety of predictive models that can forecast future outcomes from historical data patterns, Logi Predict has been designed to allow uncomplicated embedding and easy operational integration with other applications. More information is available in our Introducing Logi Predict document.
Where's the Documentation?
The Logi Analytics Developer Network is an active community of developers and your source for the documentation and other resources that help developers succeed.

The Welcome panel provides quick links to recently-opened applications, links that open applications using a variety of methods, and a link to DevNet, our online community. In addition, you'll see some useful content, from DevNet, displayed in a scrolling area.

Application Folders and Files

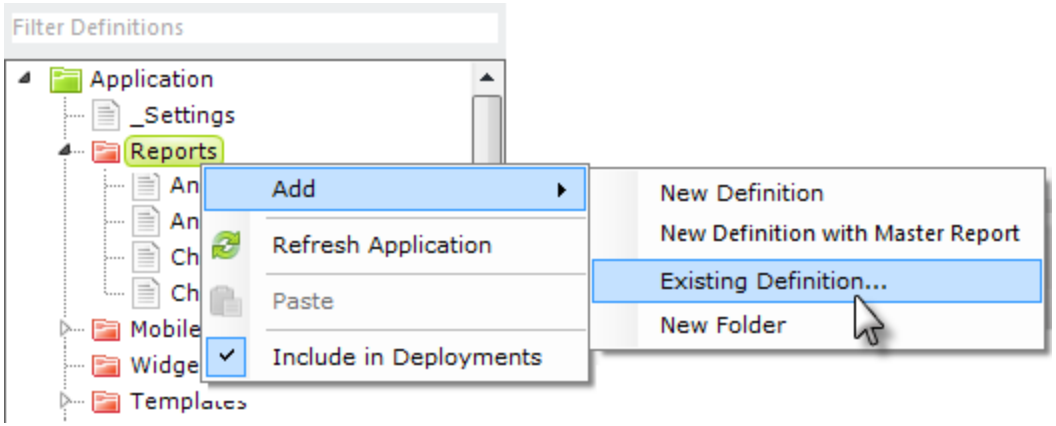
A Logi application is made up of a number of **files** that reside in an application folder group with a specific hierarchy.



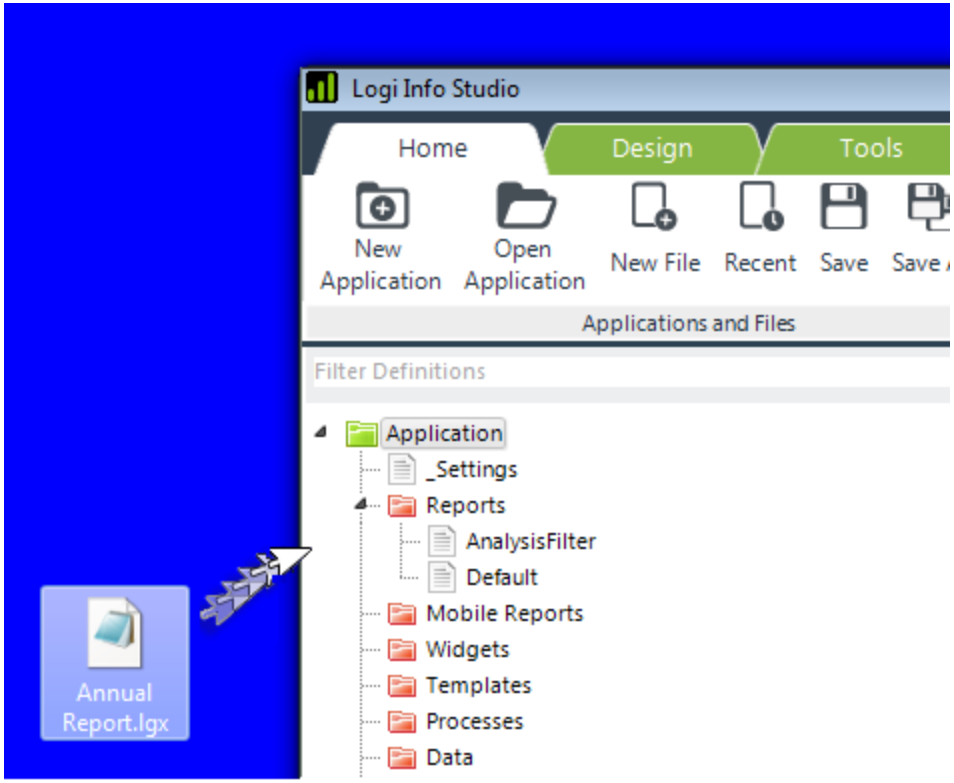
The physical folder structure that contains source files is displayed within Logi Studio in the **Application** panel, as shown above left. Standard folders are colored red and cannot be deleted. Files can be *right-clicked* and copied, renamed, and deleted, as shown above right.

Files that are deleted in Studio are placed in the Recycle Bin and can be restored, if desired.

Report definitions can also be designated as the *default* report for an application (this can useful to temporarily make a report the default during development, to avoid having to navigate to it through other reports when testing).

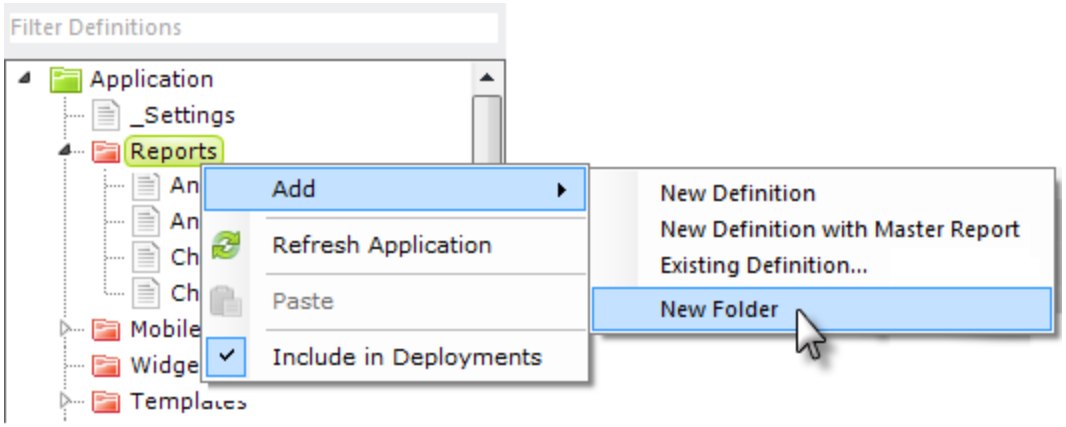


Files can be **added** to the application via the New File toolbar icon or, as shown above, by selecting and right-clicking the destination folder in the Application panel. Right-clicking the folder presents you with the option add a new file or an existing file to the application. If you select *Existing Definition*..., a **copy** of the file will be placed in your application; the original file is left untouched.

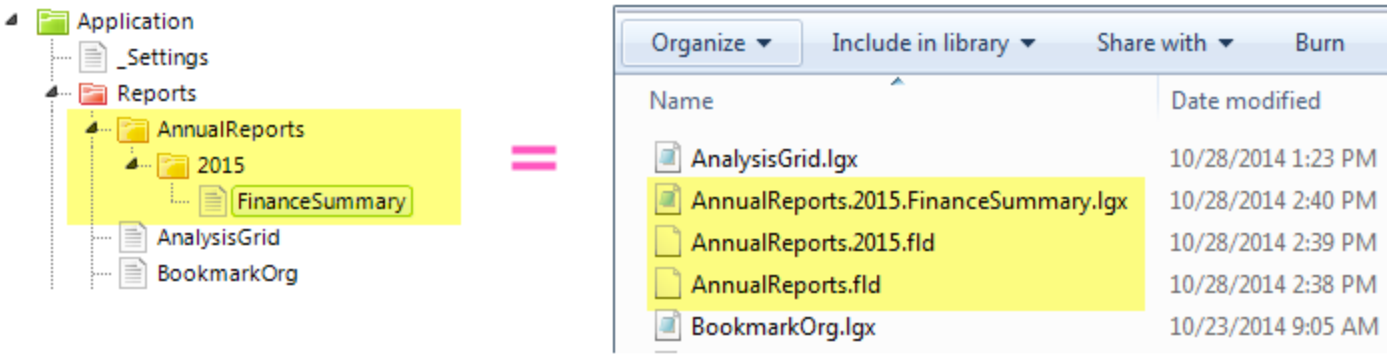


You can also **drag** files into, or out of, Studio from, or to, the operating system (from Windows Explorer or the Desktop, for example) or another Studio instance.

In Studio, files are dragged from or dropped onto the file/folder tree in the Application panel. When dropping files, if you don't explicitly drop them onto a folder in the tree, Studio will attempt to place them in the correct folders based on their file extensions. Files can be dragged and dropped within other folders in the file tree.



Studio also supports a hierarchy of "pseudo folders" for additional file organization purposes. Folders are created by right-clicking any folder, as shown above, and selecting **Add** → **New Folder**.

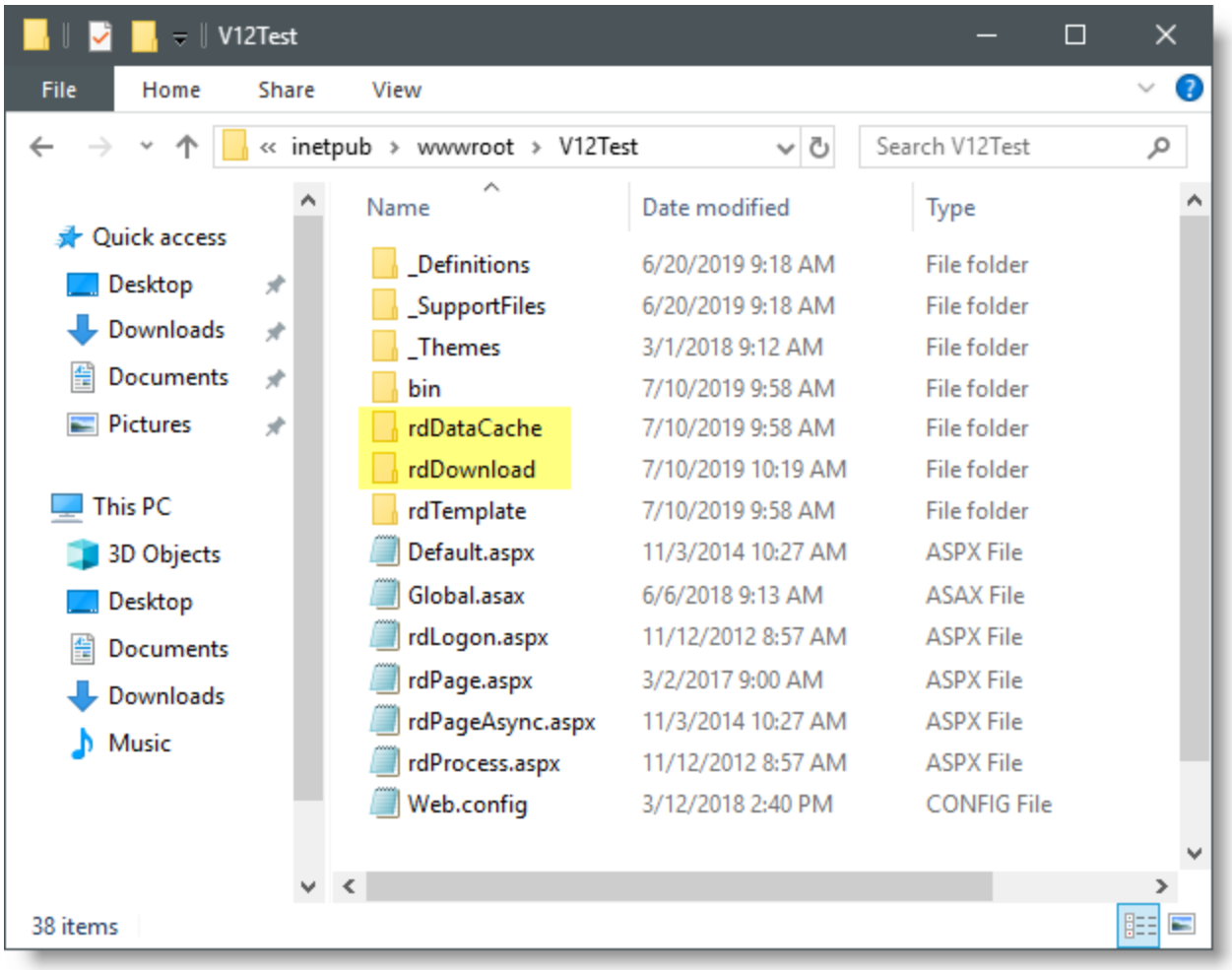


Creating folders in this manner and then creating or dragging files into them produces the kind of file names shown in the example above. As you can see, no actual folders are created in the file system and the pseudo folders are represented by files that have an .fld extension. In Studio, entire folders can be dragged into other folders to reorganize them.

The following types of files are found in a Logi application:

- **_Settings** - *The _Settings Definition* file provides application-wide configuration information. Paths, constants, data connections, diagnostic settings, etc. are all configured here.
- **Reports** - Report definitions are the basic source code files of a Logi application, with each report definition usually equating to one web page. Individual files, with an .LGX extension, are enumerated here; double-clicking a definition will open it in the definition editor in the Workspace panel. A Master Report, introduced in v12.1 and created using a wizard on the main menu Design tab, is a report template that other reports can incorporate. For more information, see *Master Reports*.
- **Mobile Reports** - These are report definitions specifically designed for creating reports for mobile devices.
- **Processes** - Process definitions are a special category of non-presentation source files containing code that executes on the web server. They can contain code for conditional and unattended processing and provide flexibility and coding depth not available in Report definitions.
- **Data** - Data definitions can be used to create an application that retrieves data from a variety of sources and then streams it out as JSON data, to Logi apps, non-Logi apps, and browser clients. See *Use Data Definitions* for more information.
- **Support Files** - These are the files that support the report and process definition files and include style sheets, images, data files, HTML files, and more. See *Support Files Management* for more information.
- **Widgets, Templates** - These folders contain files that are for special purposes.

To edit most files, double-click their name and they'll open in an appropriate editor in the Workspace panel. If you try to open and edit files in proprietary formats, such Excel spreadsheets, they'll open instead in the external application they're associated with in your computer's file system.



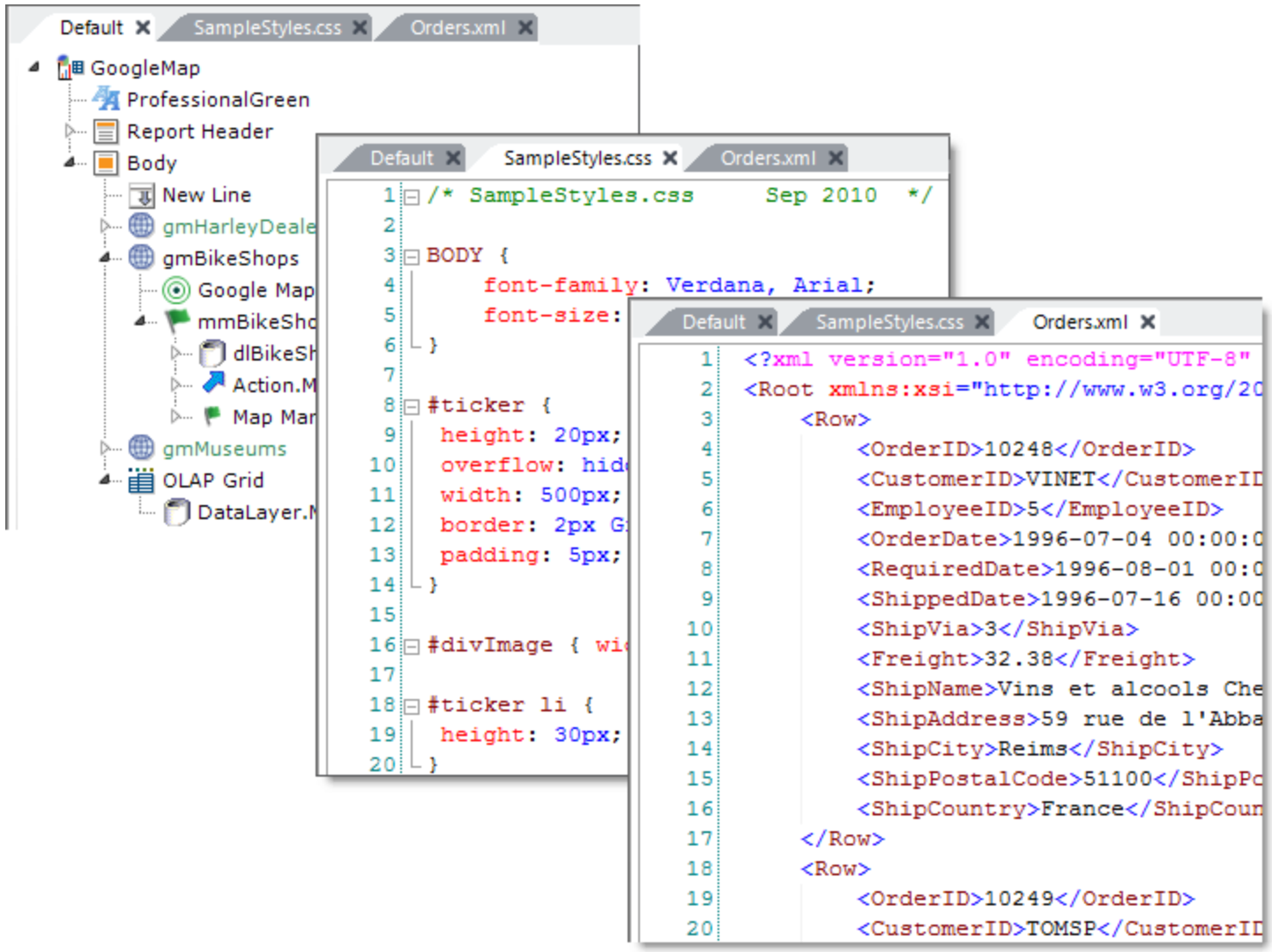
In the file system, a basic Logi application looks like the example shown above. All of the files for a single application are contained in one folder, called the "Application folder", which makes deploying the app easy.

In the example above, the `bin` folder and folders beginning with "rd" are system folders and must be present. Their contents should not be directly edited. The two folders highlighted above are created automatically when the app runs, so don't be concerned if you create a brand new app and don't see them at first. Other folders may also be created on-the-fly, based on features in your application.

Studio doesn't keep track of application files in a database or repository, so you can directly manage files in the Windows file system by copying, moving, renaming, or deleting them without causing any damage (or course, internal references to specific files may need to be updated). You can always update Studio's list of files by right-clicking any folder and selecting **Refresh Application**.

Editing Files

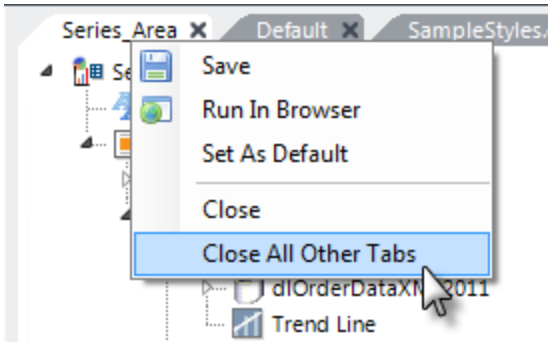
Logi Studio includes file type-aware editors for viewing and editing application files.



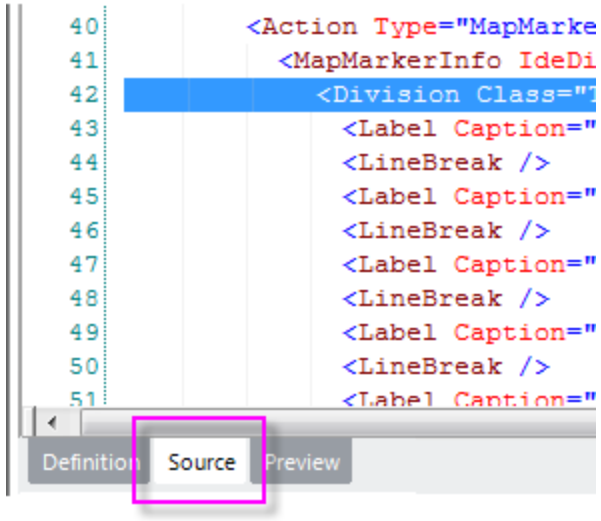
As shown above, when files are opened they're added to the **Workspace** panel as tabbed panels. The editor for each tab panel is appropriate for its file's type. Features include:

- Tabs can be reordered by dragging them left or right.
- Editors may, depending of file format, include line numbering, color coding, and collapsible regions.
- Elements and text can easily be copied and pasted between files in different tab panels.
- Copying an element also copies all of its child elements.
- Intelligent completion of element names and HTML tags is provided when typing.
- Undo-Redo (Ctrl-Z, Ctrl-Y) is available for all changes made since the last Save.
- The **Preview** panel for definitions includes Forward, Back, Refresh, and Stop buttons.
- Multiple elements that are hierarchical peers in the element tree can be selected at once, then moved or copied as a group.
- Elements have their valid child elements available for selection directly from their (right-click) context menus and from the Element Toolbox panel.

Files are closed, and their tabbed panel removed, by clicking the **X icon** in the tab by the file name. You'll be prompted to save any changes that may have been made.



Tab **context menus**, available by right-clicking, are available, as shown above. The menu options vary based on the type of file being edited in the tab panel.

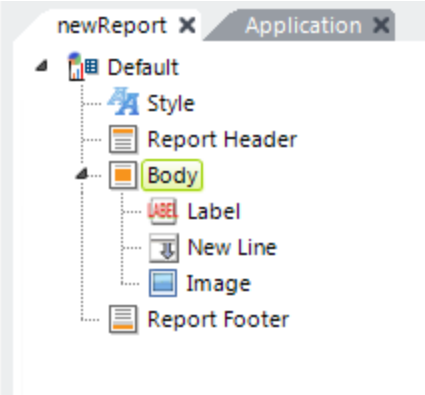


At the bottom of the Workspace panel are several tabs that appear when editing a *definition*. The **Source** tab provides a view of the definition's underlying XML source code. If an element is selected in the definition when the Source tab is clicked, its corresponding XML tag will be highlighted in the source code, as shown above. Skilled developers can edit or copy-and-paste source directly here if necessary.

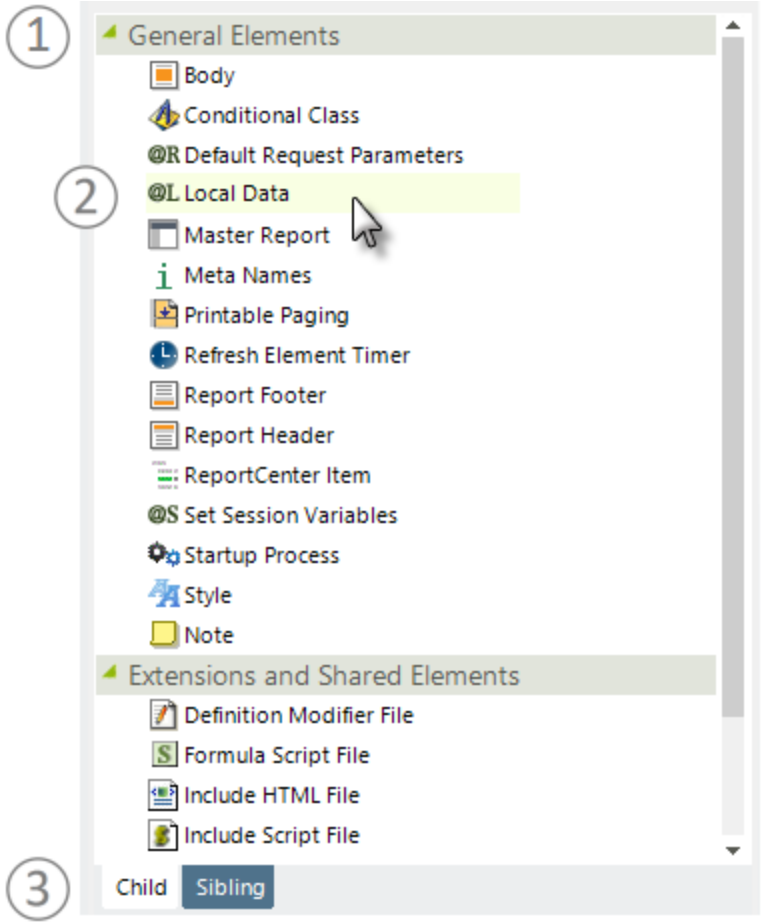
Copying the XML source is an easy way to send all or part of the definition to someone else; for example, to another developer or to Logi Support in an email. In fact, just selecting and copying an element right in the element tree itself and then pasting it into a email or another application will result in its XML source code being pasted.

Working with Elements

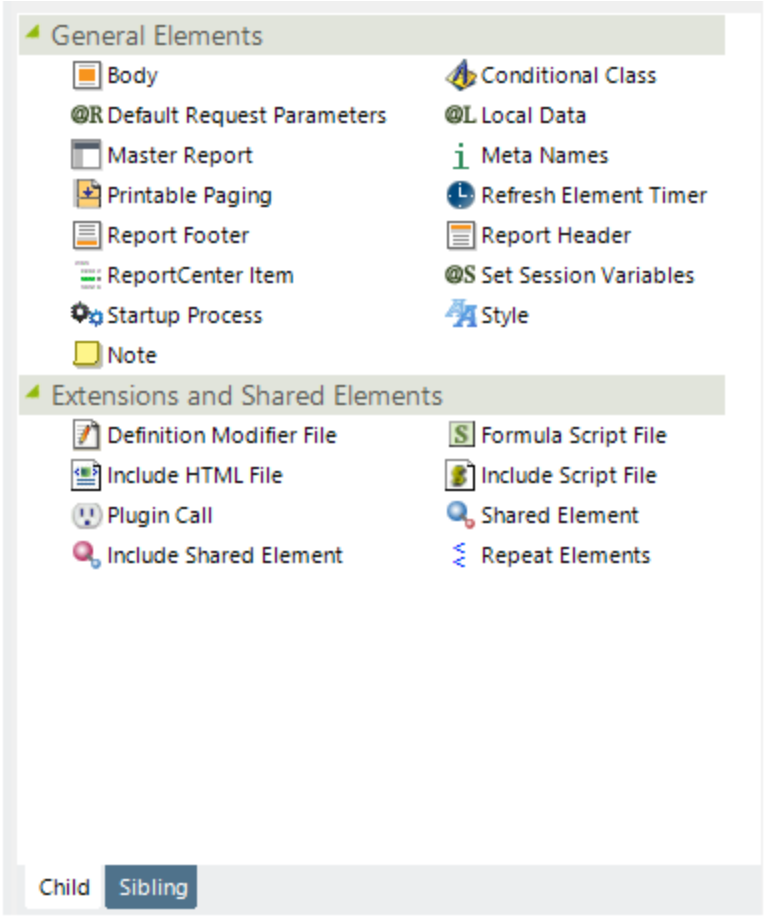
The primary source code files for Logi applications are report definitions. These files contain XML-formatted text and Studio allows you to create and edit this text by graphically manipulating objects, called "elements". In Studio, elements are arranged in a tree structure that mirrors the top-to-bottom structure of a web page and some elements have names that are similar to HTML tags.



The example above shows elements in the "Element Tree" in a report definition. An element can be the "parent" of another element, creating a **parent-child** relationship between them. In the example above, the **Label**, **New Line**, and **Image** elements are children of the **Body** element. These relationships are governed by *rules* which are enforced within Studio by restricting the availability of child elements for assignment to a parent element.



Single column

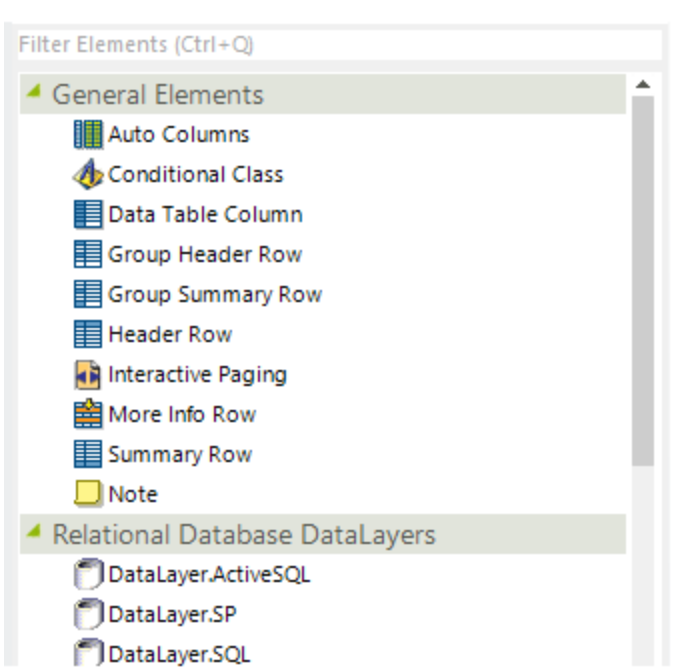


When panel is widened, multiple columns are displayed

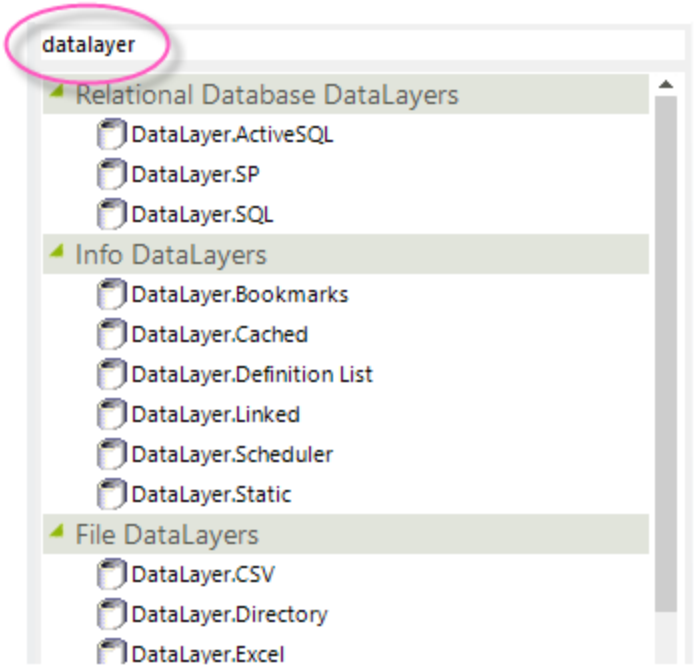
Elements can be added to the element tree by first selecting the parent element and then double-clicking the child element in the **Element Toolbox** panel, or by dragging-and-dropping the element. The Element Toolbox, shown above, includes:

1. Elements are grouped by function, and groups can expanded or collapsed.
2. Elements can be double-clicked, or dragged-and-dropped, in order to add them to a definition.
3. Tabs to specify the relationship of the displayed elements to the element selected in the definition: **Child** or **Sibling**. Only elements which apply to the specified relationship type will be displayed.

You can also filter the elements shown in the toolbox:



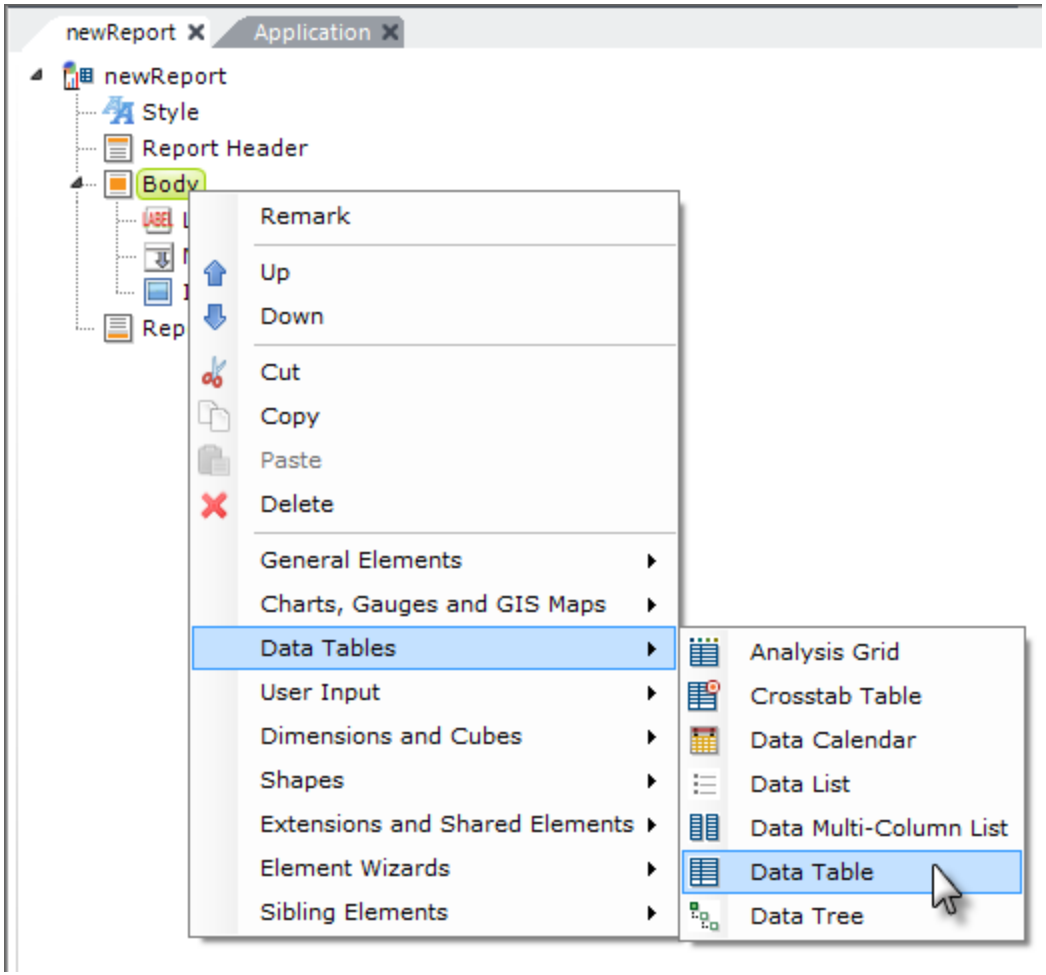
Unfiltered Element Toolbox



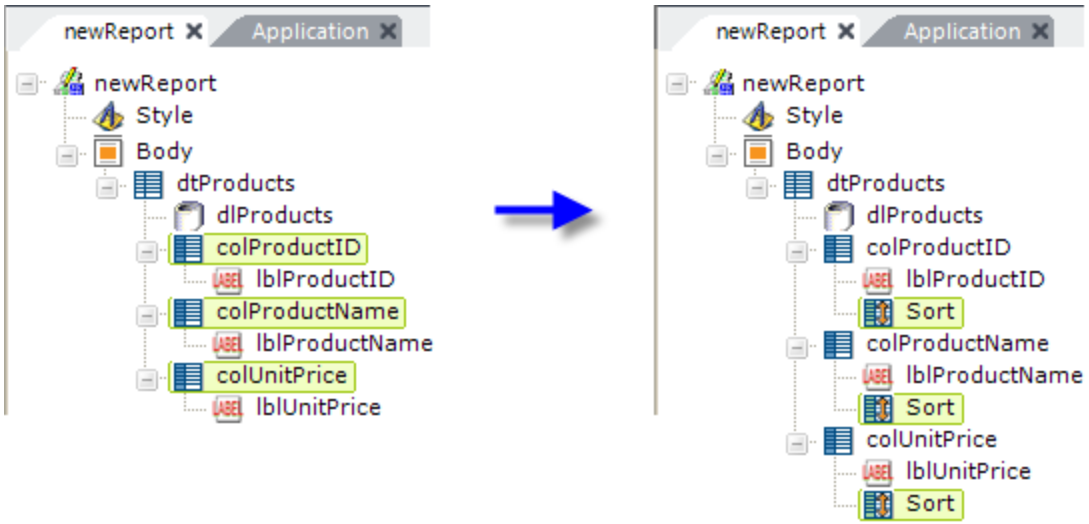
Filtered Element Toolbox

Just enter letters into the Filter Elements box, shown above, and the elements in the toolbox will be filtered as you type. And, of course, only elements that appear in the toolbox because they have a relationship with the element selected in the Element Tree will be included in the filtered list.

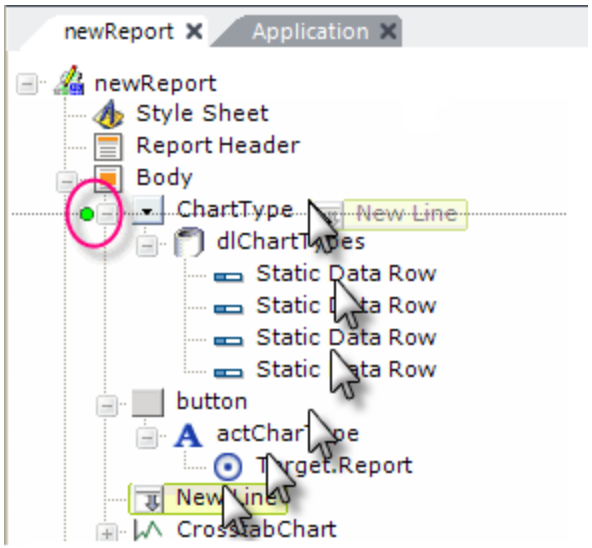
Child and sibling elements can also be added to the definition by right-clicking an element in the tree and selecting them from the **context menu** that appears:



An element's context menu, shown above, includes several useful options for working with elements, such as moving, copying, cutting, or deleting them.



You can also select *multiple* parent elements (hold down the "Ctrl" key while selecting them) and then add a child element beneath all of them at once. In the example above on the left, multiple **Data Table Column** elements have been selected, then the **Sort** element was chosen from the Element Toolbox. The result, on the right, shows a Sort element added beneath each selected Data Table Column element.



Once in the tree, elements can be moved up or down by dragging them or by using the **Up** and **Down** tool bar arrows or similar items in their context menus. The example above shows a **New Line** element being dragged upward in the tree; note the circled green positioning dot that helps you determine where to drop it. When an element is moved in this way, all of its child elements move with it. Even groups of elements (peers, on the same level in the tree) can be selected and moved at the same time.

Elements can also be copied and pasted, individually or in groups, back into the same definition, into another definition, or even into another Logi application in order to reuse them. All of their child elements will be copied with them.

Even XML source code, copied from the Workspace panel's **Source** tab, can be pasted right into the element tree and will instantly appear there as the correct elements. This is a great way to share code via email or discussion forums.

Two special elements, **Shared Element** and **Include Shared Element**, can be used to make one instance of a group of elements re-usable throughout your application. Elements placed under a Shared Element element in one definition can be referenced using the Include Shared Element element in other definitions. This is particularly useful, for example, for managing

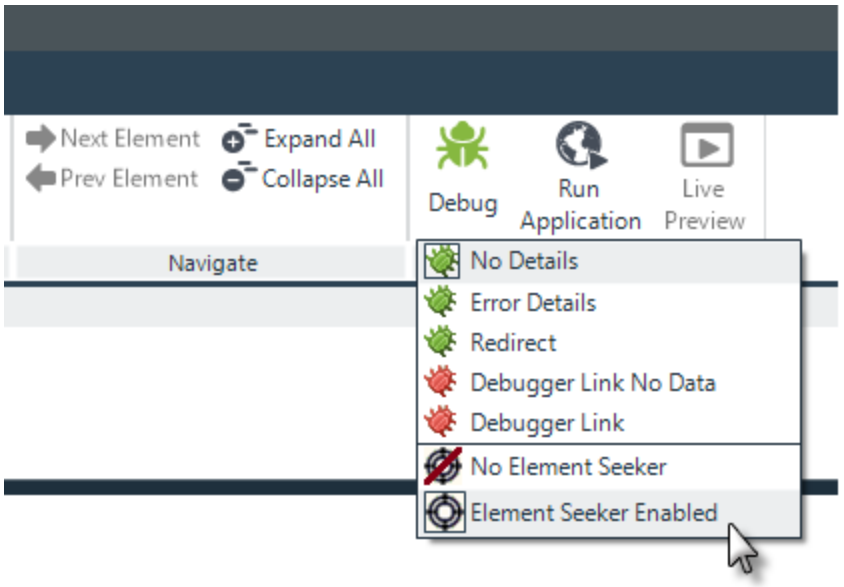
headers and **footers** that appear on many reports within an application. More information about these useful elements is available in *Shared Elements*.

Element Positioning

By default, Logi definitions use *Flow Positioning* to arrange the elements on a report page. This is a very common scheme in web applications in which each element's location is dependent on the location of the elements preceding it. This is the most flexible scheme and works best across multiple browsers.

Element Seeker

Logi Studio includes a feature, the **Element Seeker**, that lets you quickly locate an element in a definition file by selecting it in your preview or browser window. To use it, you must be working in Logi Studio v12.2 SP5+ and the Logi Server Engine version of your Logi application also has to be v12.2 SP5+. In addition, if you're using the Internet Explorer browser, you must be using IE 9 or later. Here's how it works:



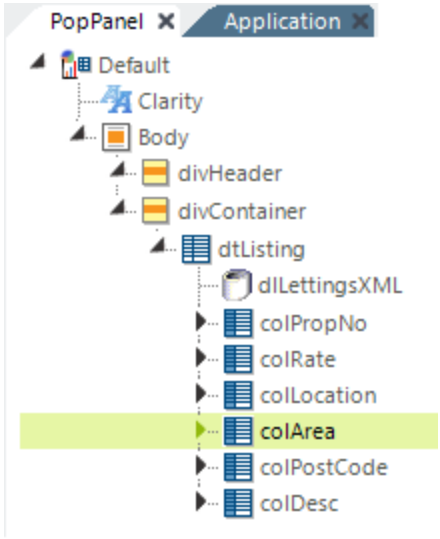
Seeker icon appears in lower right-hand corner of browser page

Enable the Element Seeker in Studio's Debug menu, as shown above. When you preview or browse a report page, you'll see the Seeker icon in the lower right-hand corner.

London Flats to Let

Prop #	Weekly	Location	Area	Post Code	Description
10047	£575	Sinclair Road	Brook Green	W14	A wonderful spacious lower ground two bedroomed flat with large reception good sized rooms and garden, located on the popular Sinclair Road.
10048	£550	Goldhawk Road	Stamford Brook	W6	Situated in a luxury development, this fabulous two bedroomed apartment offers spacious accommodation finished to the highest possible standards with beautiful Amtico wood floor and secure parking.
10049	£525	Blythe Road	Brook Green	W14	Smart two bedroomed flat, benefitting from recent refurbishment and boasting a large reception room, spacious bedrooms and great access to the amenities of High Street Kensington.
10050	£500	Kings Street	Ravenscourt Park	W6	Offering spacious accomodation throughout and benefiting from a delightful south-facing roof terrace, this two bedroomed split-level apartment boasts excellent neutral presentation, private entrance and garage

When you click the Seeker icon, it will turn green and enter selection mode. As you move your mouse cursor around the page, different regions will be framed in green, as shown above. Click a framed region...

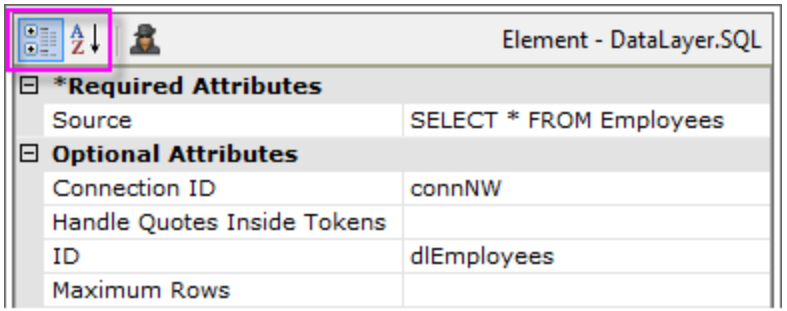


... and Logi Studio will open the correct report definition and select the element responsible for the chosen region, as shown above.

To turn *off* the Seeker's selection mode, you'll need to refresh your preview or browser page.

Setting Element Attributes

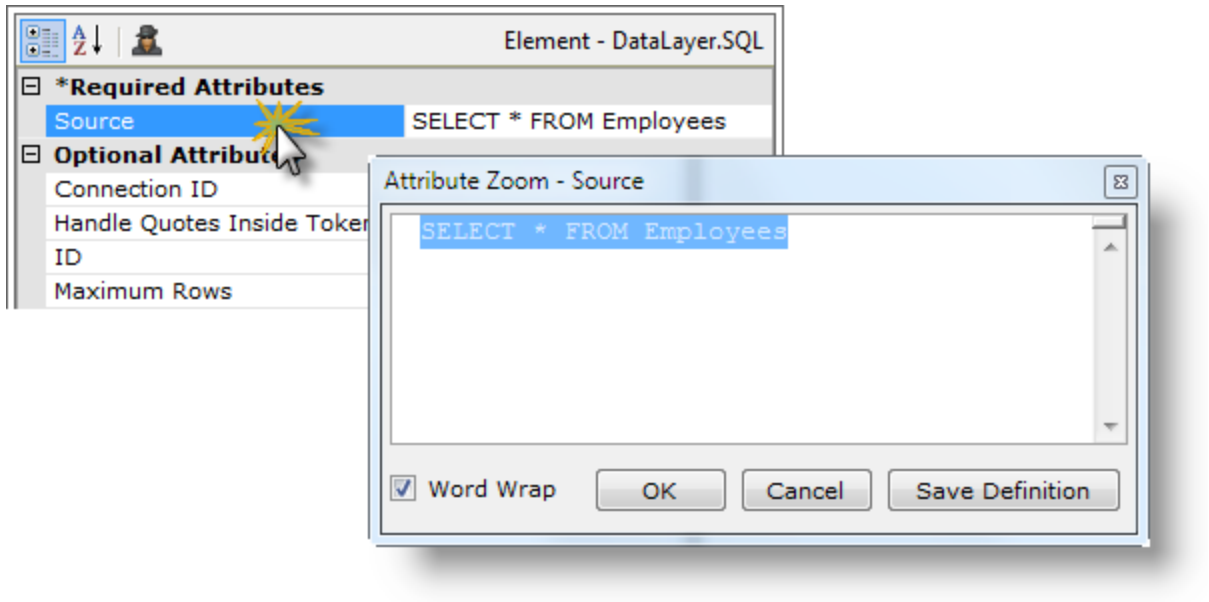
Elements can have a number of properties, or "**attributes**" in Logi-speak, that govern how they behave. The attributes of a selected element appear in Studio's **Attributes** panel and the number of attributes varies depending on the nature of the element.



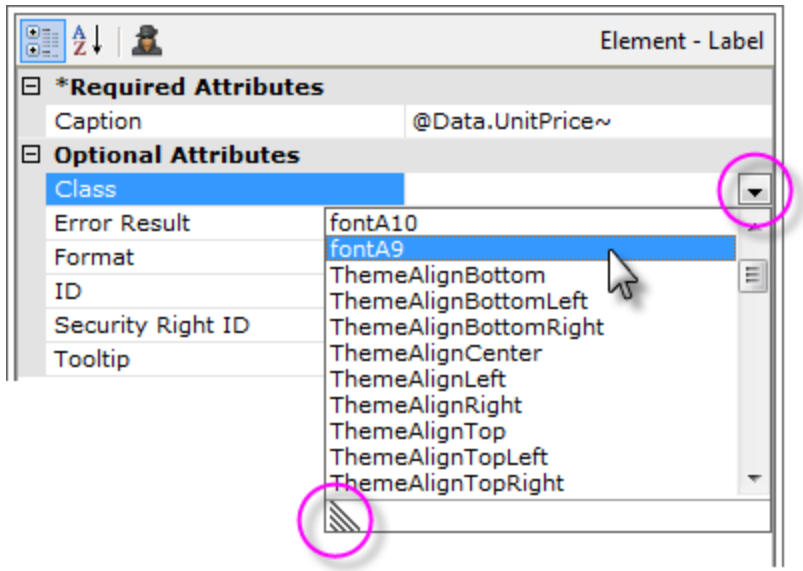
As shown above, attribute *names* are displayed in the left column and attribute *values* are entered in the right column. Some attributes are required while others are optional. Attributes can be sorted in two ways: by **Category** or **Alphabetically**, by clicking the highlighted icons. When sorted by category, the required attributes appear at the top.

Hovering your cursor over an attribute value will cause the complete value to appear in a tooltip, even multi-line SQL statements.

Depending on the attribute, values can be entered in several different ways. Generally, values can always be typed-in directly.

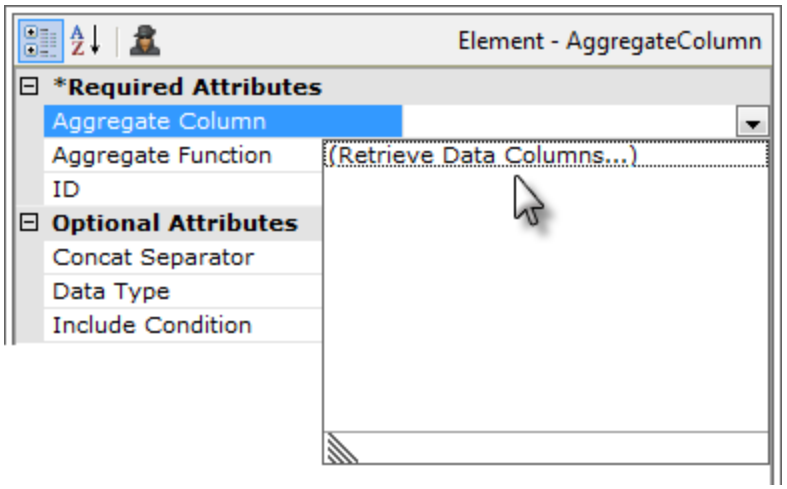


Double-click an attribute name to open its **Attribute Zoom** window, shown above, which contains a larger, multi-line text entry area and allows you to see longer values without scrolling.



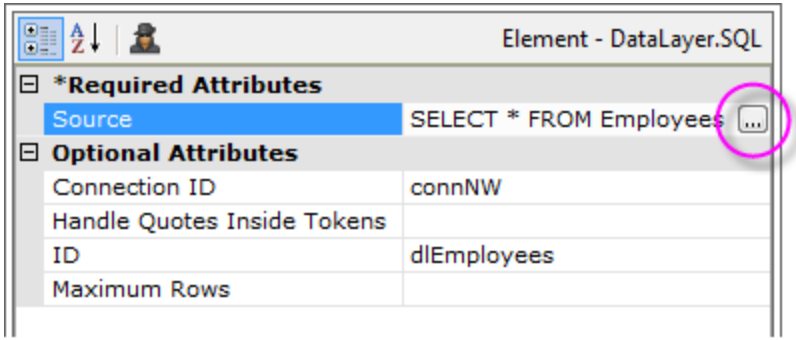
The values for some attributes consist of a specific set of valid choices and, as shown above, these are chosen by first clicking the circled down-arrow and then selecting a value from the drop-down list of choices. The length of the list can be expanded by *drag-ging* the indicated corner of the list.

After selecting a value from the drop-down list, you can hold down the Ctrl key and selected additional values in the list, creating a comma-separated list of attribute values.

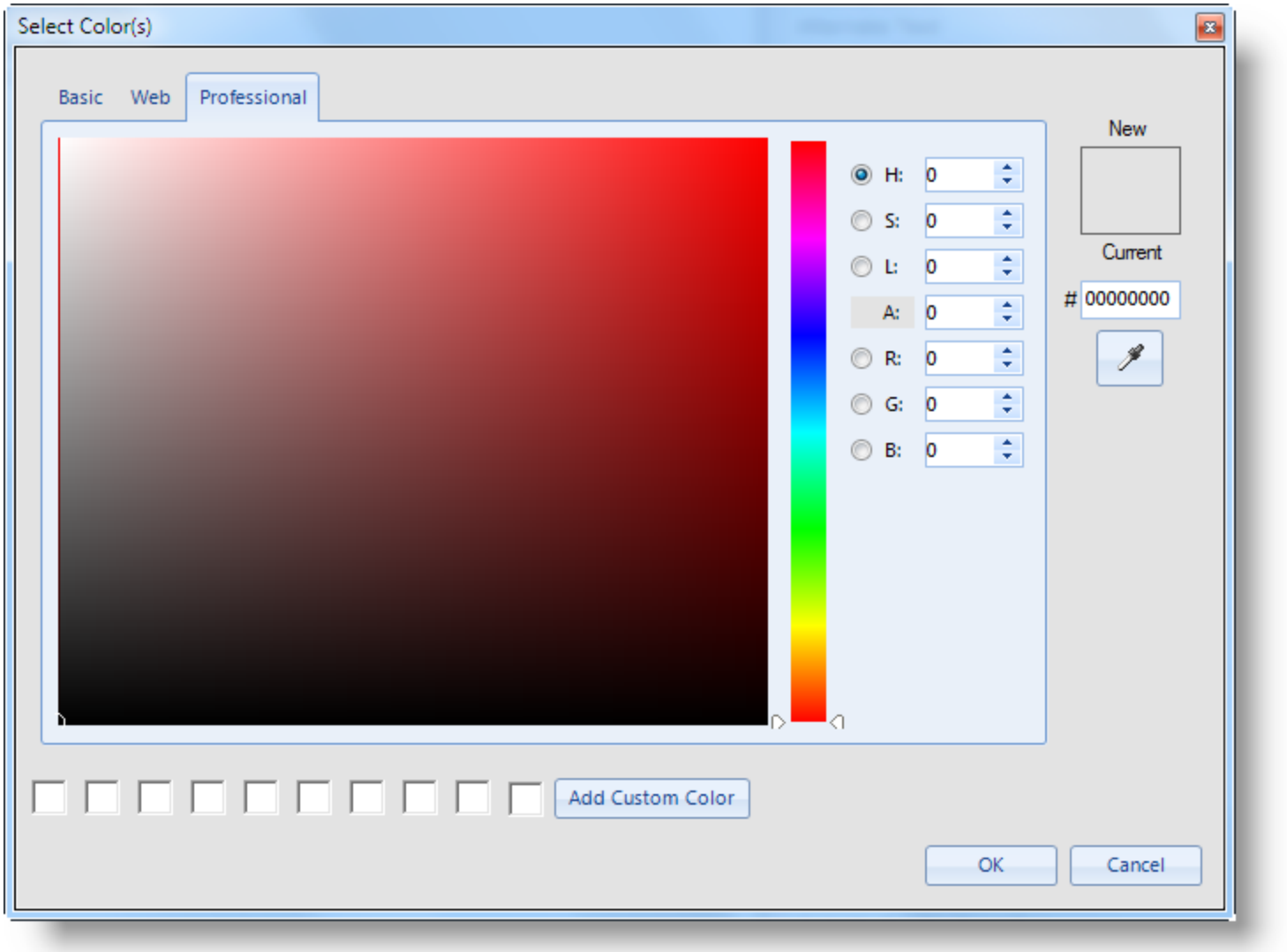


Where attributes require **column names** from a datasource, as shown above, a drop-down list is provided that will, first, retrieve the column names, and then, display them in a drop-down list for easy selection.

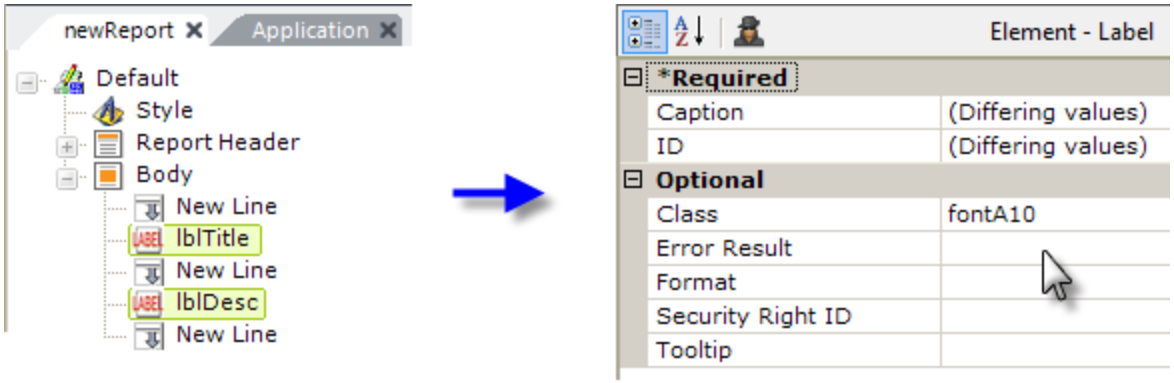
This feature only works with datasources that respond to **schema** requests, such as databases, and so does not work with flat files. In addition, the parent Data Table or chart *must have a valid ID* for column names to be retrieved.



For some attributes, clicking the browse button in the value column will launch a **wizard** or **tool**. For example, when the browse button is clicked in the image above, it will open the **SQL Query Builder** tool, or...

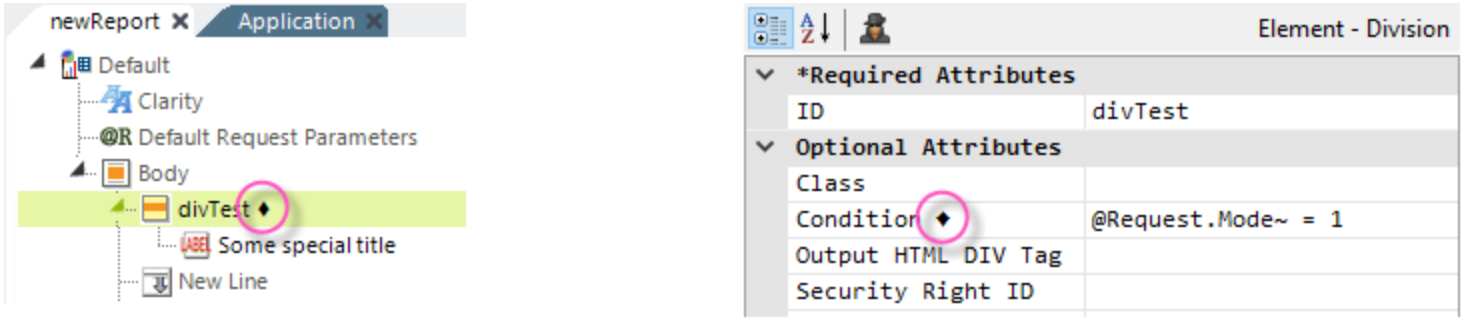


...in the case of Color attributes, clicking the browse button will cause the **Color Selector** dialog box, shown above, to be displayed.



Attributes can be assigned for several elements *at the same time*, by selecting **multiple elements** (hold down the Shift or Ctrl keys while clicking elements). The Attributes panel will show all of the attributes the selected elements have in common; attributes that already have different values assigned will be indicated by the "(Differing values)" message. As shown above, assigning an attribute like Class in this state will set the individual Class attributes for each of the selected elements.

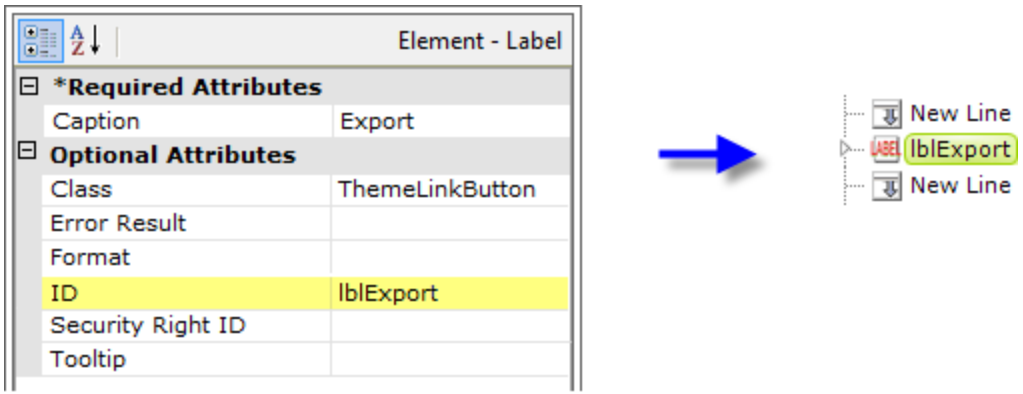
Elements that have a value in their **Condition** or **Include Condition** attributes are now indicated by a diamond icon:



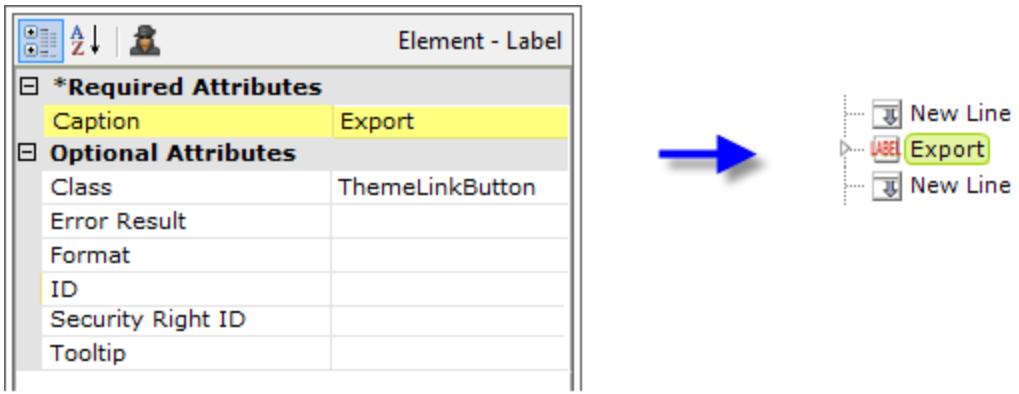
The diamond icon, shown circled above, appears in both the Element Tree (left) and in the Attributes Panel.

Element ID Attributes

Some elements *require* a unique ID attribute value, which has to be entered by the developer. Developers are encouraged to use a "naming scheme" that conveys both element *type* and *purpose*.



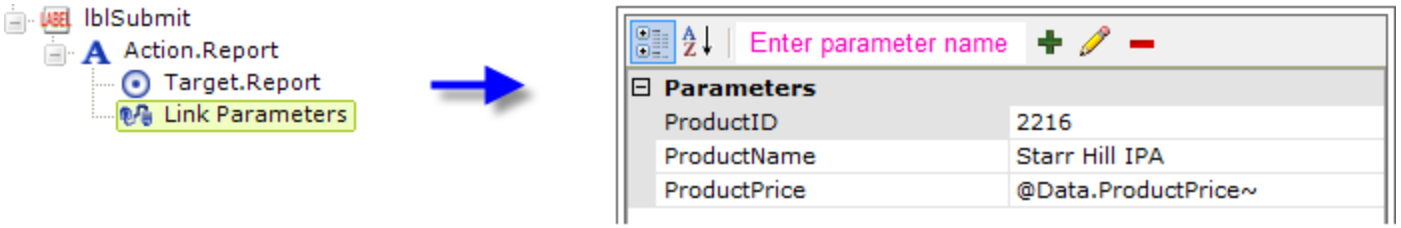
However, most ID attributes are *optional*.



In addition, if an optional ID value is left blank, Studio will identify the element in the Element Tree by using text from another attribute value, as shown above. The attribute that Studio selects for this purpose varies, depending on the element type. This generally eliminates "duplicate ID" errors (often caused when copying and pasting elements) and to relieves you, in most cases, of the need to provide IDs at all.

Variable and Parameter Lists

Certain elements represent lists of name-value pairs. In this case, the Attributes panel displays an interface that allows you to enter as many arbitrary attribute names and values as needed.



In the example above, a **Link Parameters** element has been selected and three attributes have been created.

Attributes are created by entering a name in the text box at the top of the Attribute panel and clicking the "+" icon. The new attribute then appears in the list and its value can be set by typing a value directly into the right-hand value column. Values can consist of numbers, text, tokens, and, in some cases, formulae.

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 will save the edited attribute name. Attribute values can be edited directly, as usual, by placing the cursor on them.

Attributes can be removed entirely by selecting their name and clicking the red "-" icon.

Express Attribute Navigation

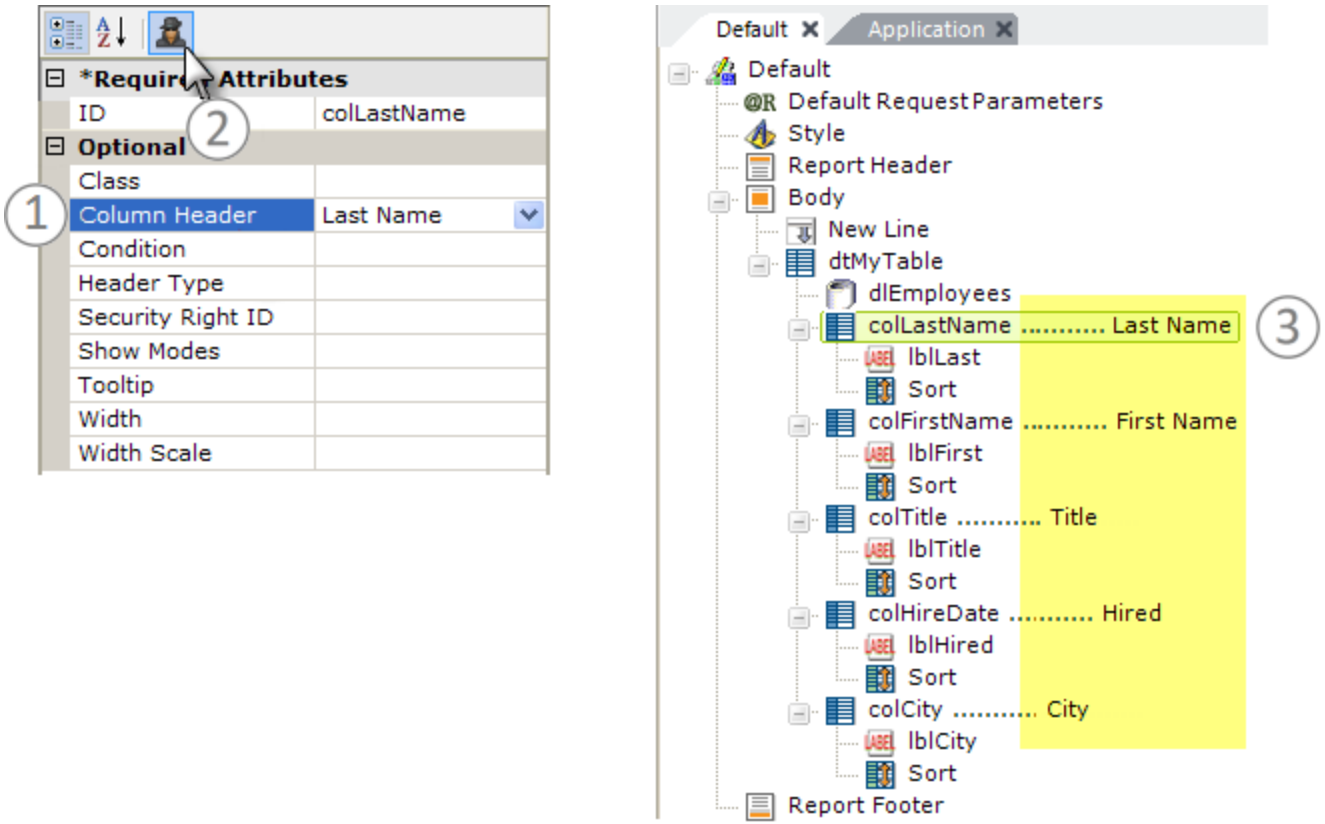
Some developers may prefer keyboard navigation to set attributes, avoiding constant switching between their keyboard and their mouse. Here are some "keyboard shortcuts" that allow this kind of mouseless navigation:

Keys	Action
	Once you've entered an Attribute value, the **Tab** key can be used to commit the value and move to the next value down. Tab can also be used to move from an element to its attributes: Highlight an element and press the Tab key three times. Focus is placed on the last value attribute visited, or the first empty attribute, and you can begin typing a value immediately. *No cursor bar appears in the attribute value.*
	**Commit Value and Move** - After typing an Attribute value, press **Enter** to commit it, and then the Up/Down arrow keys can be used to move to another attribute.

- **Abandon Value and Move** - After typing an Attribute value, press **Esc** to abandon it, and then the Up/Down arrow keys can be used to move to another attribute.

The Attribute Spy

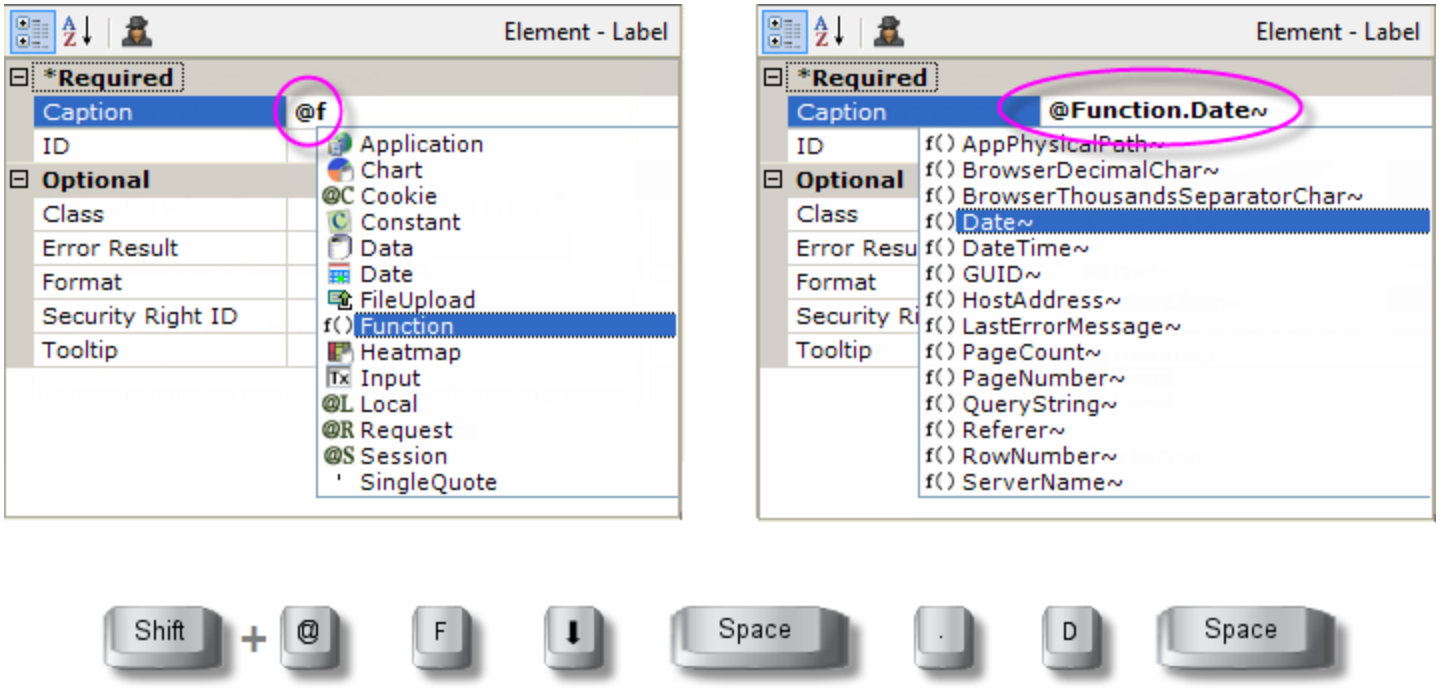
The **Attribute Spy** feature allows you to see all of the values assigned to a particular attribute for all elements at once. This is a very powerful tool that makes it easy to compare values, such as style sheet class assignments, or to get an overall view of values without excessive navigation and mouse-clicking.



As shown above, (1) simply select an attribute and then (2) click the **Attribute Spy** button at the top of the Attributes panel. Text will appear in the **Workspace** panel (3) indicating the values for that attribute for every element in the tree (excluding elements that do not have a value assigned for that attribute). Click the Attribute Spy button again to hide the text.

Intelligent Token Completion Feature

One of the most useful features in Logi Studio is the **token completion** feature. It's often difficult to remember all of the token types and identifiers that are available and this feature presents them and also prevents typing errors.



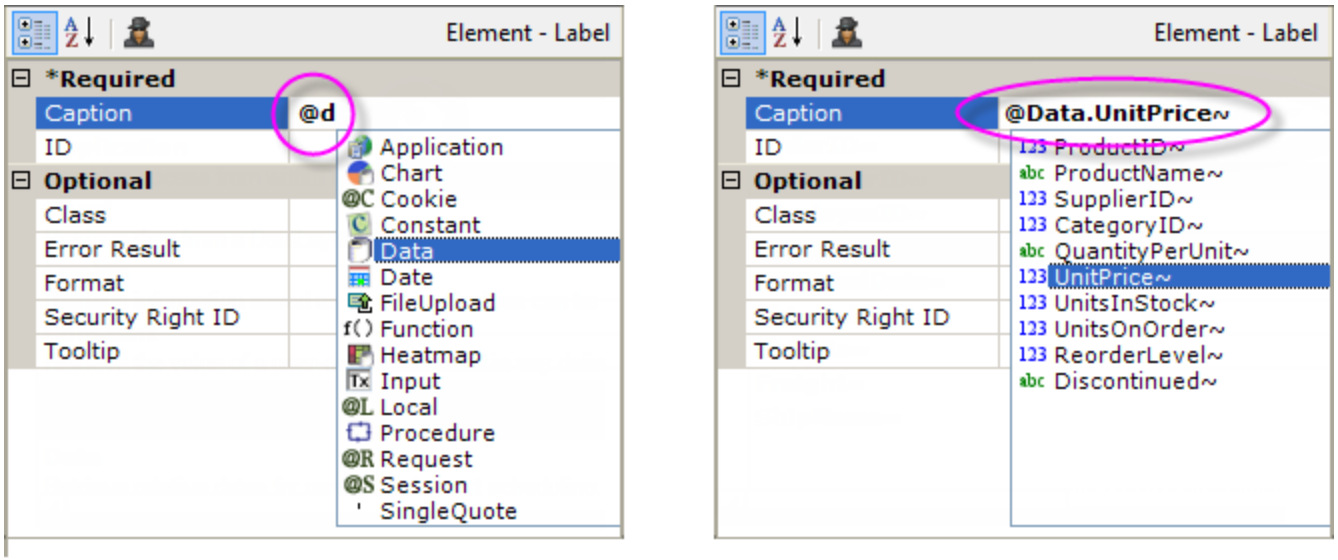
Here's how this feature works:

1. Type an **@** sign and the first letter or two of a token type in an attribute value and a list of **token types** will appear, as shown above left. Use the **Up** and **Down** arrows to navigate to the desired type, if necessary, and press the **Space Bar** to select it. The token type will be added in the attribute value. You can instead type @ + first letter, for example, @f, to

navigate to the desired token.

2. Type a **period** (.) after the token type, as shown above right, and a list of **token identifiers** will appear. Type a letter or two and/or use the arrow keys again to navigate and the Space Bar again to select the desired identifier. It will be appended to the token type in the value column, including the trailing tilde.

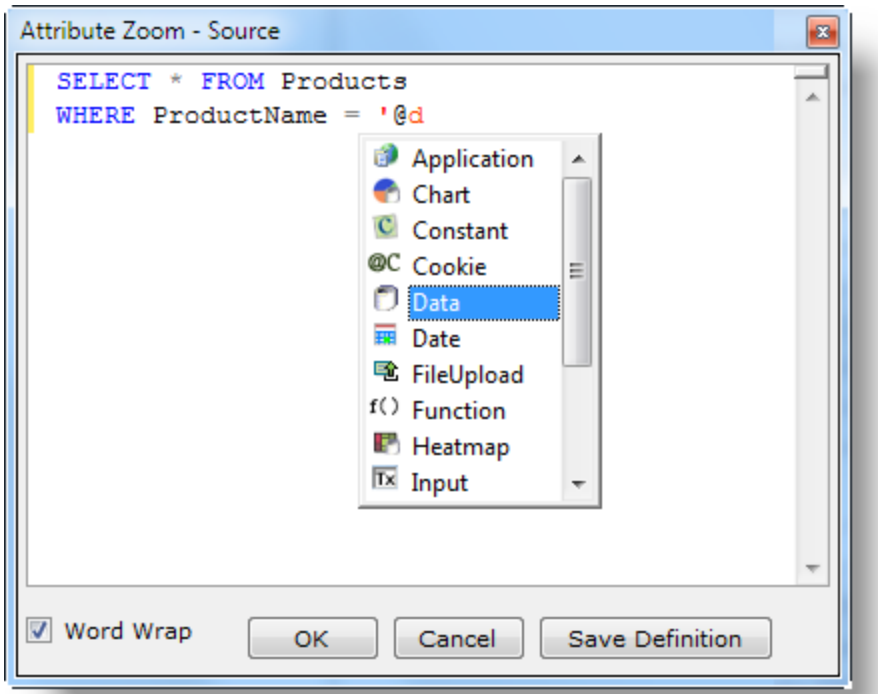
Similarly, you can use the intelligent completion feature to provide column names, for example, for **@Data tokens**:



1. Type an **@** sign and the letter "d" in an attribute value and a list of **token types** will appear, as shown above left. If it's not positioned there already, use the **Up** and **Down** arrows to navigate to the **Data token** item. Press the **Space Bar** to select it. The token type will be added in the attribute value.

2. Type a **period** (.) after the token type, as shown above right, and a list of **columns**, based on the datalayer in use, will appear. Use the arrow keys and **Space Bar** again to selection the desired column and it will be appended to the @Data token in the value column, including the trailing tilde. Under certain circumstances, you may instead see an item named "Retrieve columns..." which will get the column names and then display them. This feature only works with datasources that respond to **schema** requests.

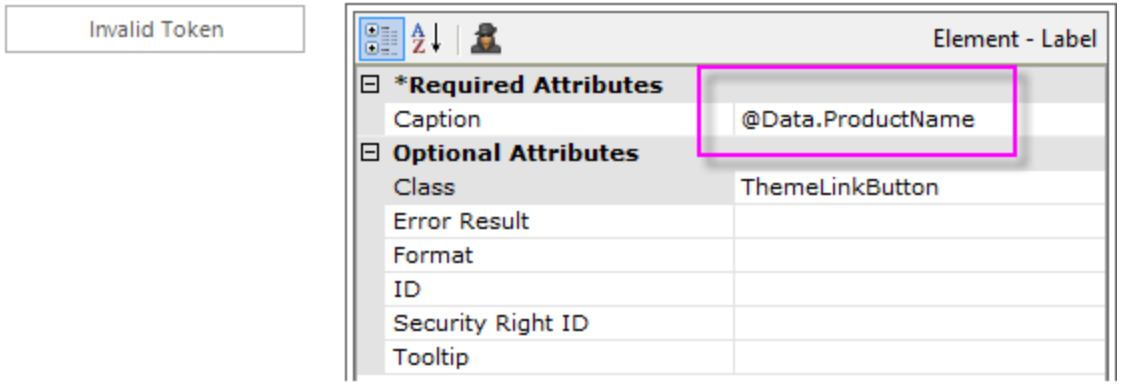
The token completion feature also appears in other places, too:



For example, when you use the **Attribute Zoom** window and enter tokens, as shown above. Also, in the **SQL Query Builder**, if you edit your query manually and use tokens, it will pop up. The Space Bar and Period keystrokes work in these instances in the same way as described earlier.

Token Validation

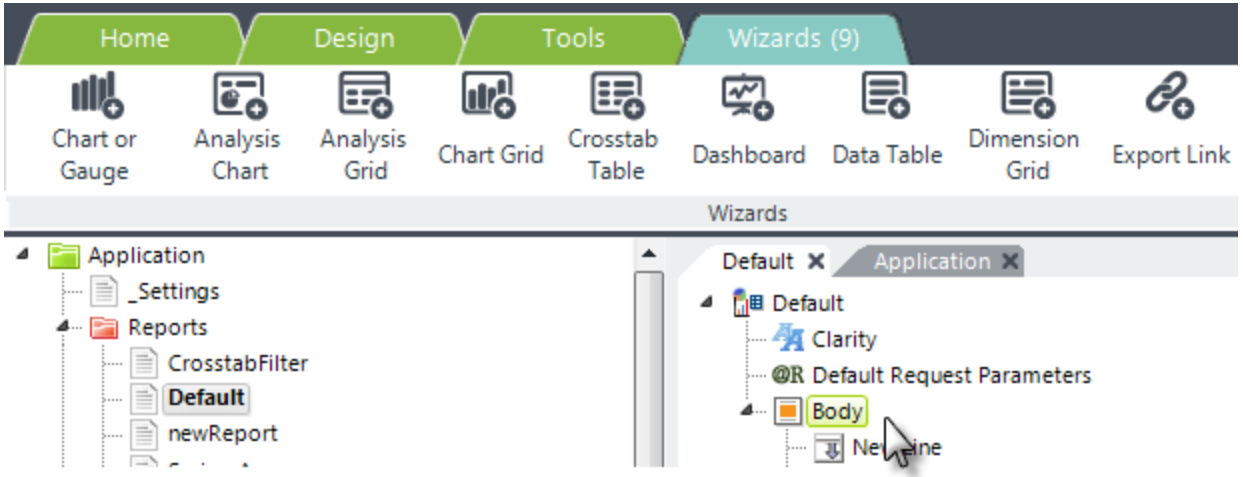
Studio includes a token validation feature which examines tokens as you enter them and flags those that are incorrectly spelled.



In the example shown above, the tilde ("~") has been left off of the end of the Data token. Studio responds by displaying an **Invalid Token** button in the current definition tab. Clicking the button will take you to the offending token. The validation routine looks for missing tildes and misspelled token types ("@Reqest" instead of "@Request", for example).

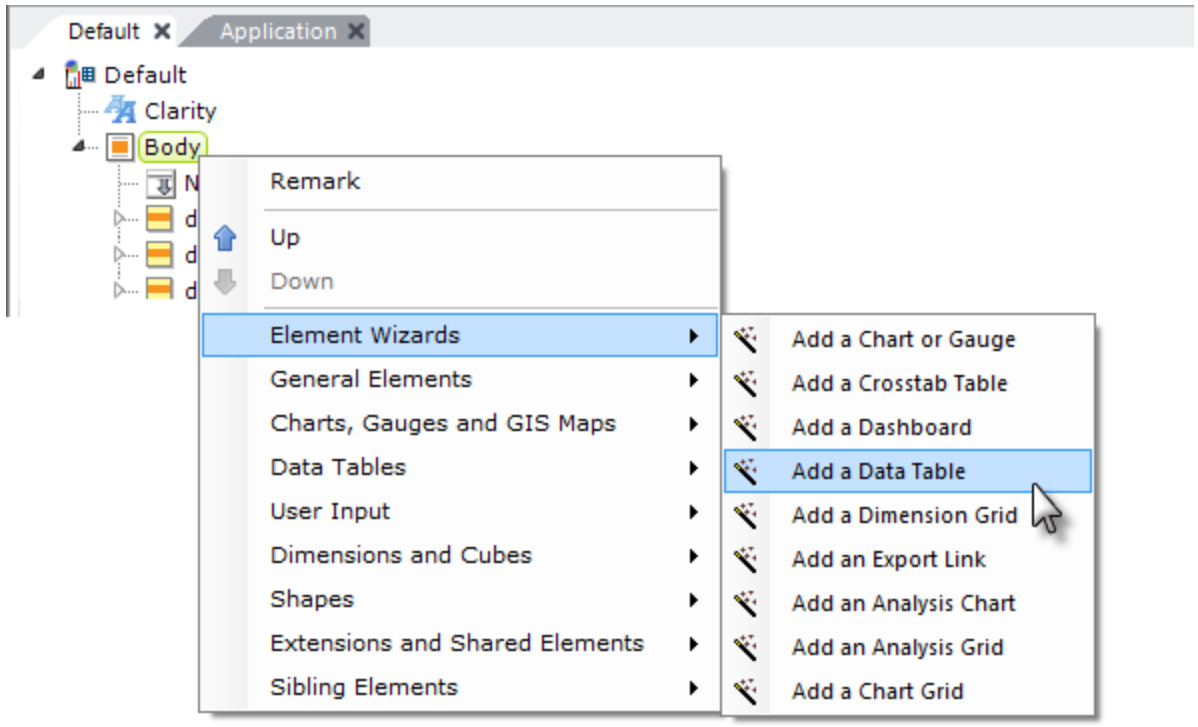
Studio Wizards

Studio includes "wizards", or automated procedures, which walk you, step-by-step, through complex tasks, from creating new applications to building charts. They save a lot of keystrokes and mouse clicks and provide repeatable, consistent productivity. Wizards can be invoked in several ways.



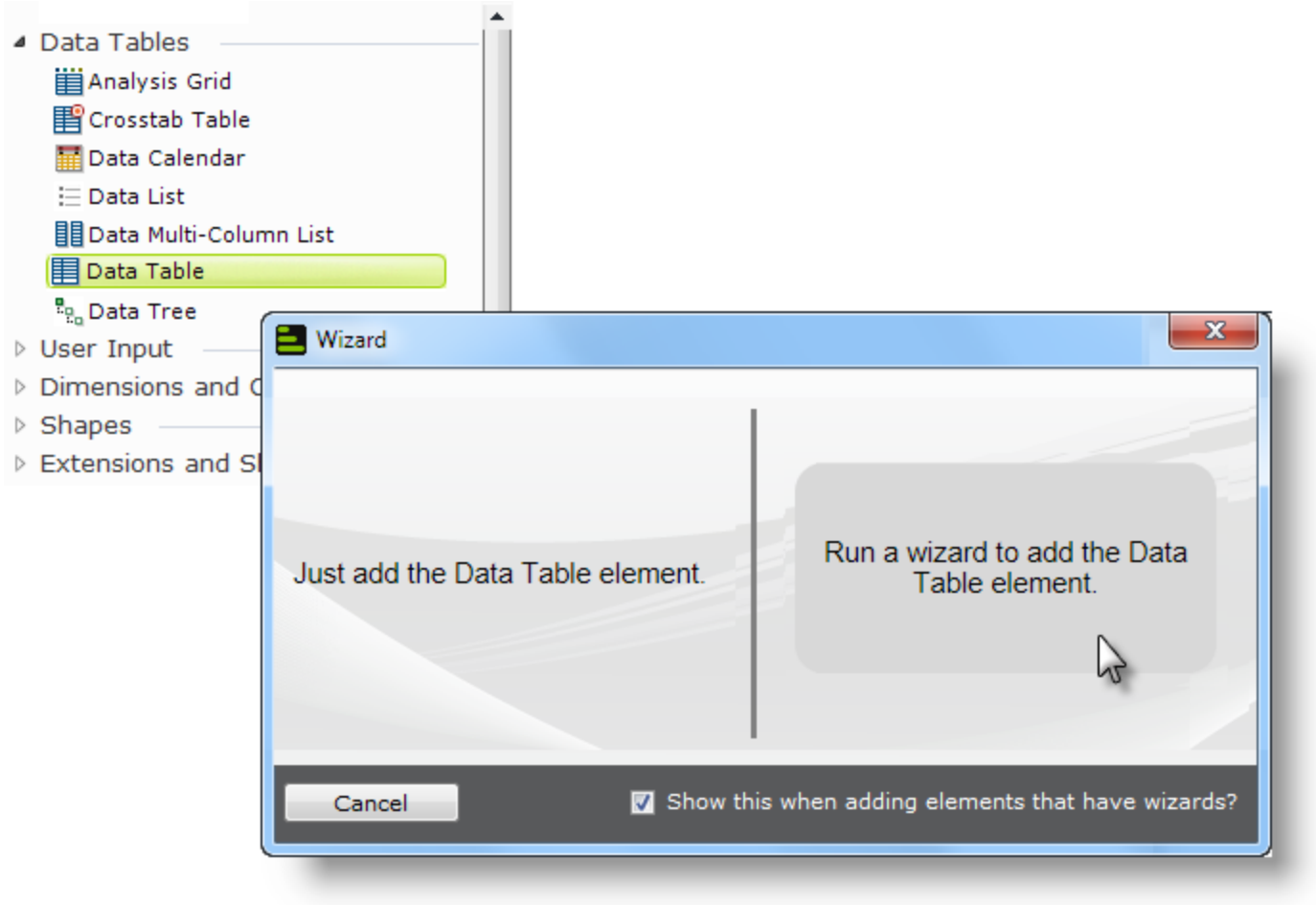
Wizards on the Main Menu

When you select an element that has wizard support in the Element Tree, a special tab will appear on the main menu, containing items for the appropriate wizards, a shown above.



Wizards on the Context Menu

Element wizards are also available in the context menus that pop-up when you right-click an element, as shown above. As in the main menu, their availability depends on which element is selected in the definition. The wizards shown above are specific to the **Body** element.



By default, if you double-click an element in the Element Toolbox that has a wizard associated with it, a dialog box like the one shown above will be displayed. It offers you the choice of inserting the element into the Element Tree directly (and configuring it manually) or running the wizard (which assists in configuring it). This behavior can be changed by unchecking the "Show this..."

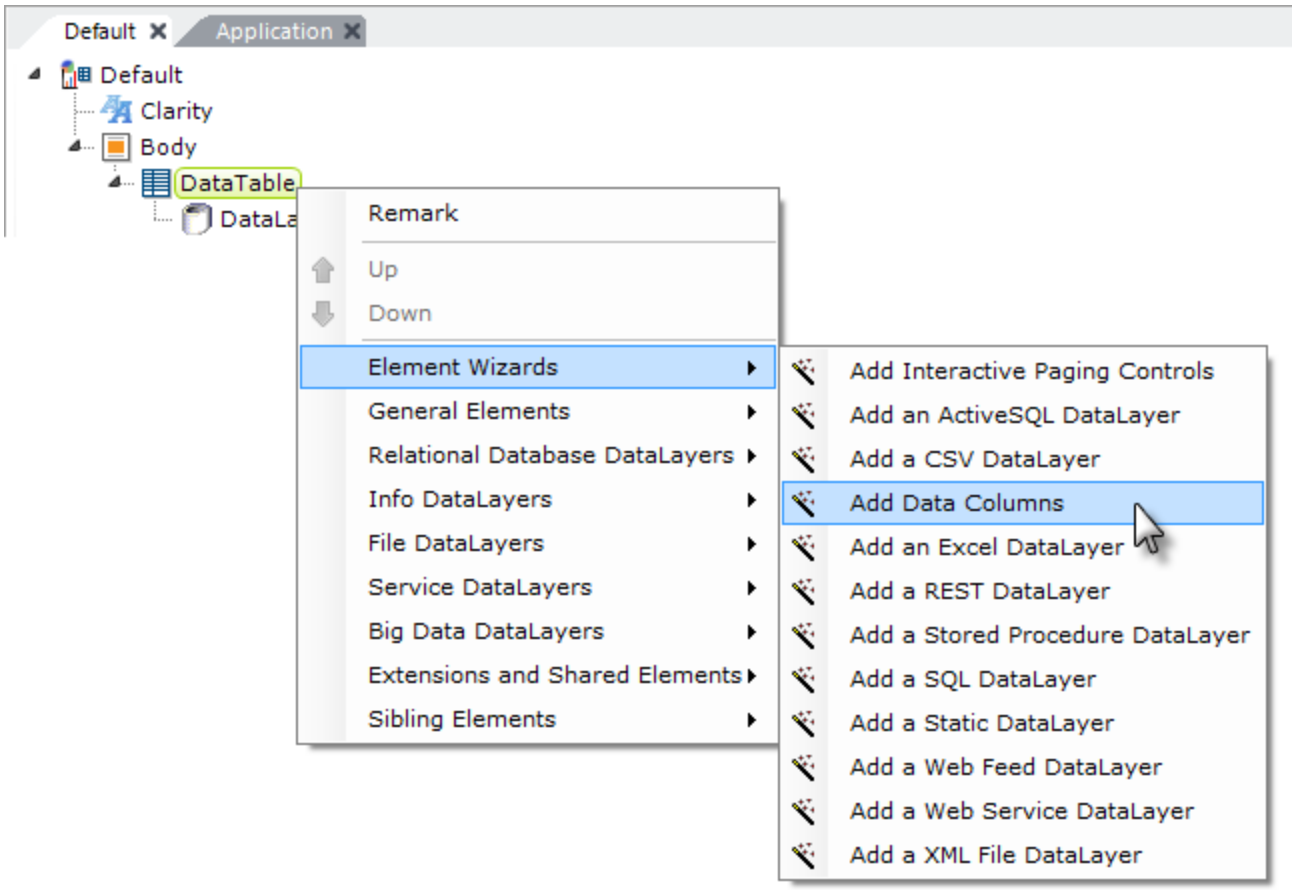
check box, in which case elements will just be inserted in the future. You can also control this behavior using the Tools → Options menu option.

Wizards are also capable of validating your definition and will let you know, for example, if you're missing an element that needs to be provided before the wizard can proceed.

Studio includes other kinds of wizards as well, such as the **Master Report Layout** wizard. This wizard, available on the main menu's Design tab, let's you select the components and layout for a master report that can be used to provide a framework for other reports, and then builds the definition for you. For more information, see *Master Reports*.

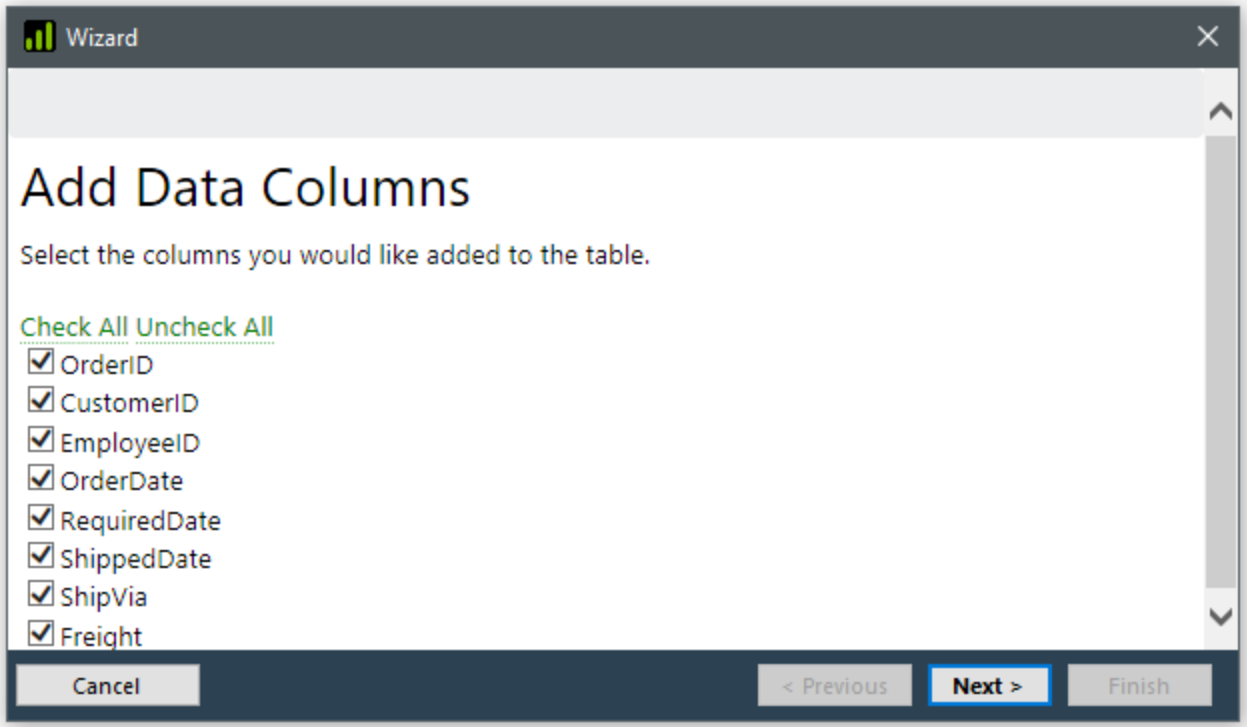
Wizards in Action

Studio's wizards are too numerous to document individually (there are more than 50!) but let's look at a representative example of a wizard in action:

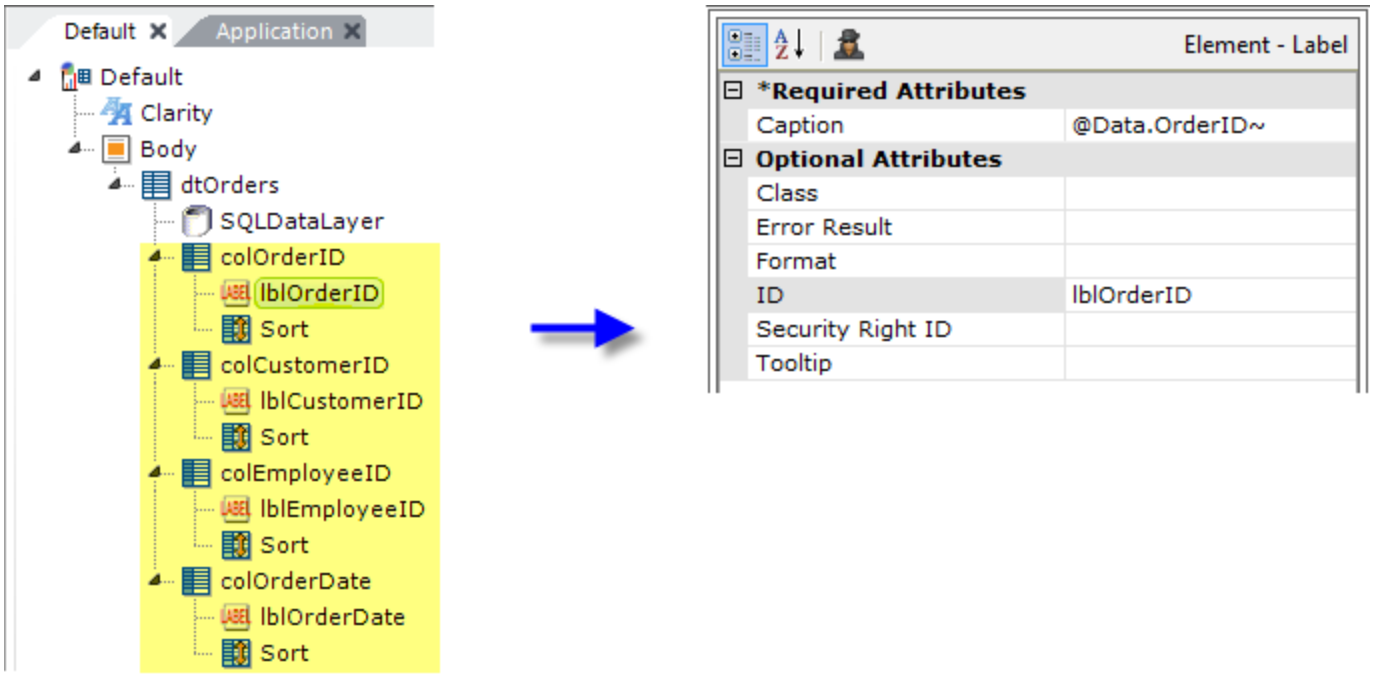


In the definition shown above, Data Table and DataLayer.SQL elements have been added beneath the Body element, and the datalayer has been configured with a *valid SQL query*. Right-clicking the Data Table displays its context menu; let's let the Element Wizard add data columns to the table by selecting the **Add Data Columns** wizard.

If the query is invalid or contains a typo, or you try this with a datasource that can't return a schema, the wizard will fail. In such a case, the wizard will display an error message that should assist you in correcting the problem.



As shown above, the wizard presents a dialog box displaying all of the columns that the SQL query returned. First, we'll click the *Uncheck All* link, then individually check the boxes for the first four columns, so that the selections look like the example shown above. Then we'll click **Next.**



The wizard will insert, and configure, *all* of the elements necessary for the Data Table columns we selected, as shown above.

Studio's wizards are a great resource for configuring elements, such as charts, that have a large number of configuration choices. For example, you could use the **Add a Chart** wizard to get the basic chart set up for you and then you could customize it from there.

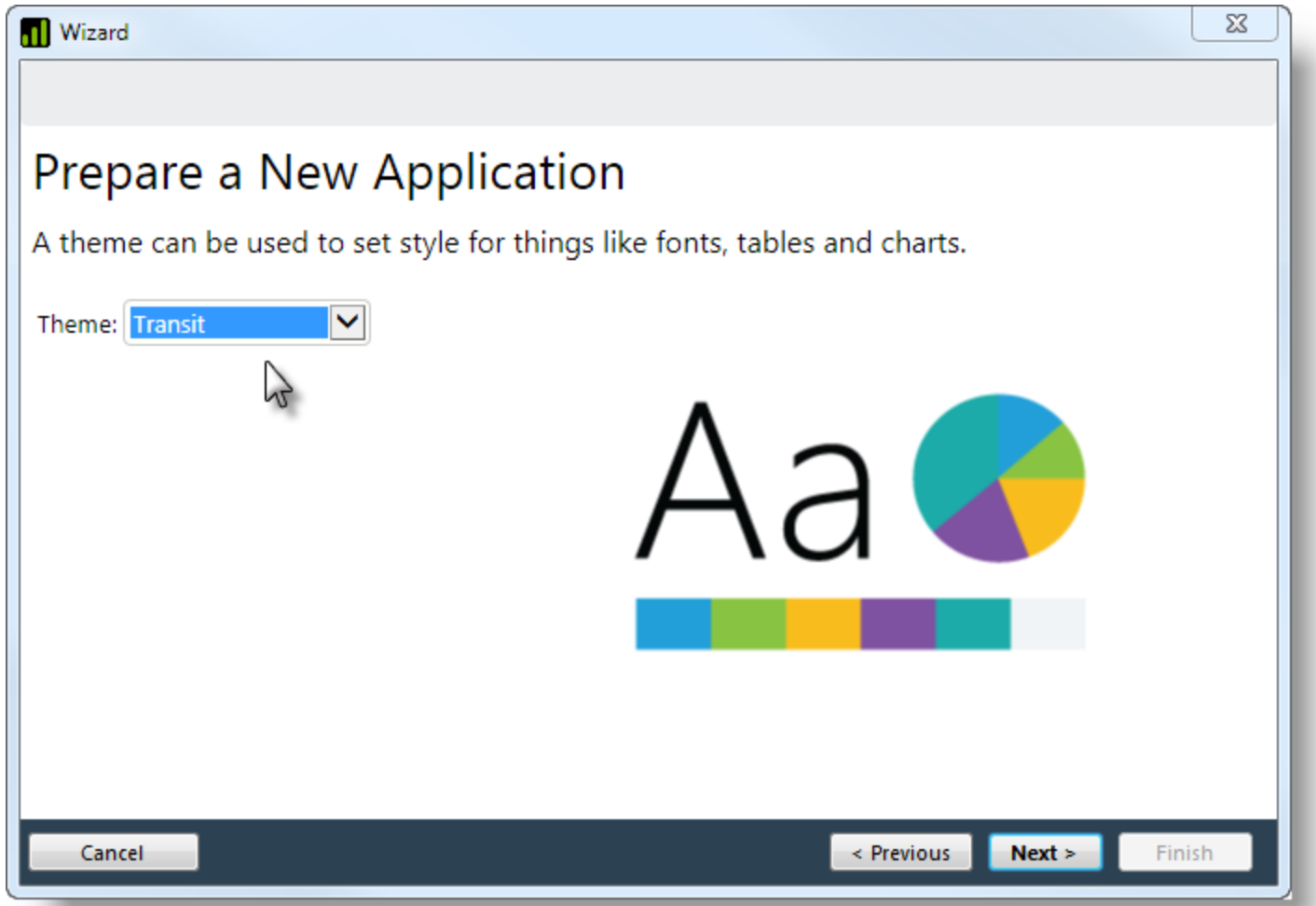
Assigning Themes and Style Sheet Classes

The *presentation* qualities of elements are set in two ways. In some cases presentation values, such as **Color**, may appear as element attributes that can be assigned directly. However, in most cases, this is achieved by assigning a **theme** to the application or definition, or by manually assigning style classes to an element's **Class** attribute.

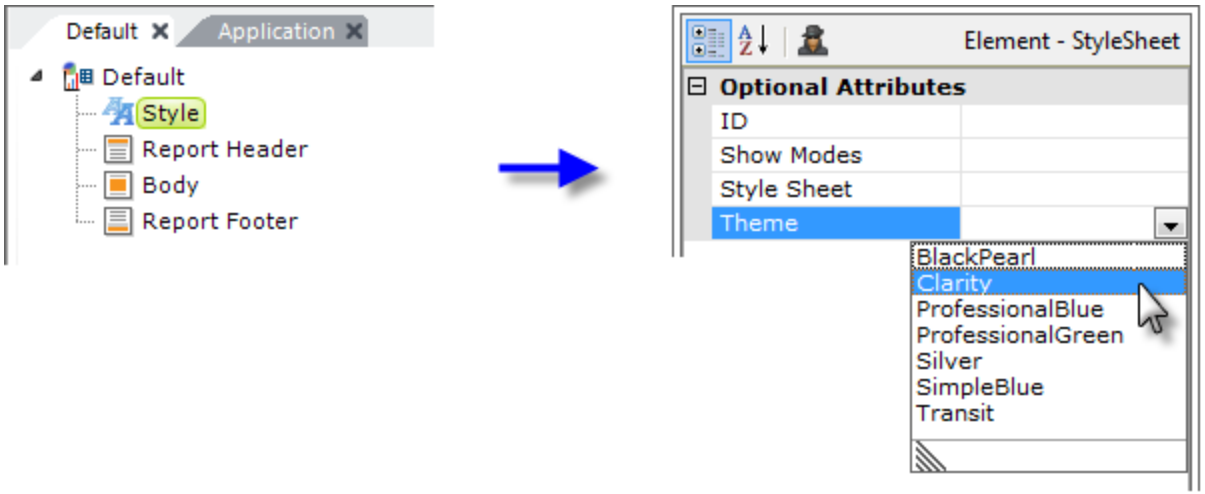
Using Themes

Themes allow developers to instantly apply a consistent "look" to their applications. A theme is a collection of files (images, style sheet, template modifier file, etc.) that impart a specific appearance to a Logi application or definition. Several standard themes are provided for your use, and new ones are available in DevNet's Samples pages.

Themes do the work for you, assigning style classes and setting other appearance attributes, making it easier to produce great looking reports without an in-depth knowledge of style classes and style sheets. You can easily switch between themes in order to experiment with them.



When you use the **New Application** wizard in Studio, one of the steps is the selection of a theme for the new application, as shown above.



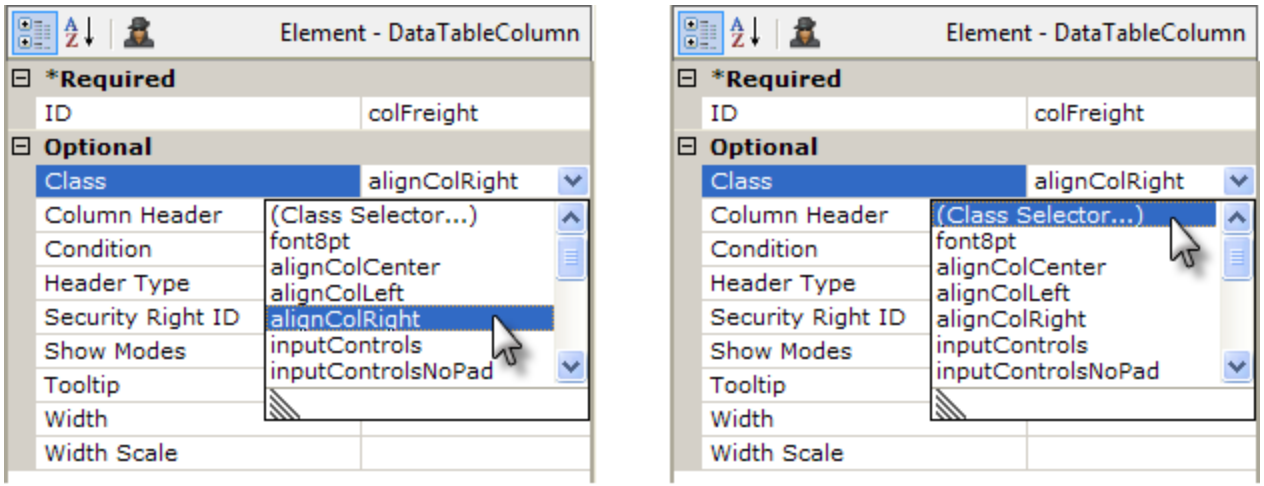
One way to apply a theme is to add a **Style** element to your report definition, then set its **Theme** attribute to the name of one of the available themes, as shown in the example above. The report will then have the theme applied to it. You can do the same for an entire application by using a **Global Style** element in the _Settings definition.

More information about themes is available in *Working with Themes*.

You can modify standard themes to create your own custom themes using the **Theme Editor** wizard in Logi Studio. For more information, see *Using the Theme Editor*.

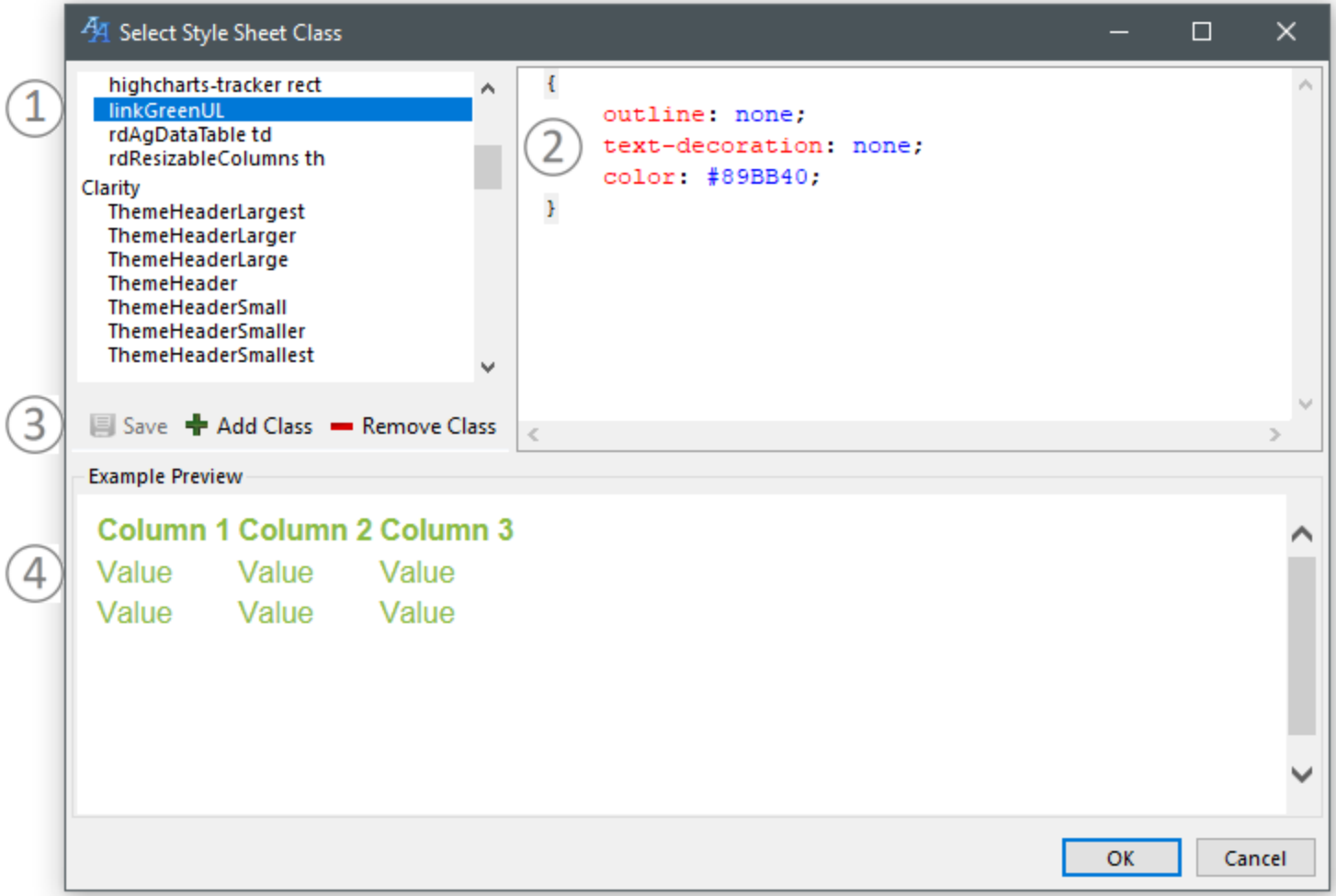
Using Style Classes

The first step in using style is to assign a style sheet. This .css file is typically placed (or created) in your application's _SupportFiles folder and assigned using the same **Style** or **Global Style** elements discussed in the previous example to assign a theme. Once the style sheet has been assigned to the definition, its style classes are available for use by elements.



As shown in the example above left, classes from the currently assigned style sheet are available for selection from a drop-down list in the **Class** attribute value of an element. The classes in the list are drawn from the style sheet assigned to the report definition or the application. This is a quick way to select classes, although class names can also be typed-in directly. Multiple classes can also be assigned, separated by spaces or commas. A preview of the effect of the assignment is shown in Studio's **Information** panel.

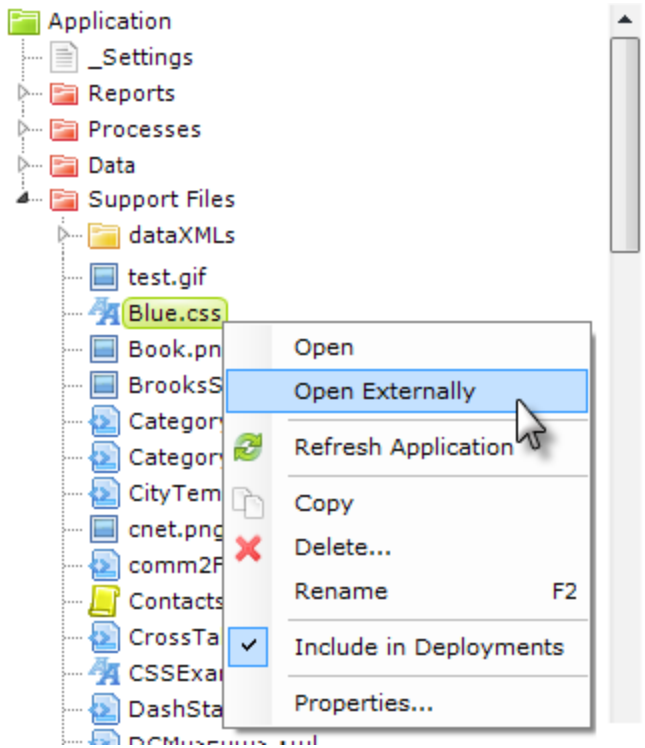
The list of class choices includes a **(Class Selector...)** item, shown above right, which opens the **Class Selector** tool.

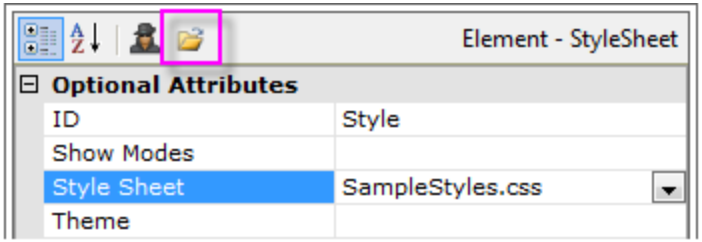


The **Class Selector** tool, shown above, allows a class to be assigned by selecting it in the class list (1) and clicking OK. The attributes of the class are shown in the upper right panel (2) and they can be **edited** there. Controls (3) allow you to save, add, and delete classes and the effect of the selected class is shown in the lower **preview** panel (4). This tool provides pretty comprehensive class management right within Studio.

A more convenient editing option is to open the style sheet, by double-clicking it in Studio's **Application** panel. It will open in a special editor in the Workspace panel and the intelligent editor will provide style selector choices and completion suggestions as you type.

Finally, you can use an external style sheet editor, if you've installed one:





You can open and edit style sheets in an external style sheet editor by selecting and right-clicking the file in the **Application** panel, as shown above left, then selecting **Open Externally** from the pop-up menu. Another method is to select the Style element in the element tree, select its Style Sheet attribute, and click the **Open File...** icon that appears at the top of the **Attributes**

panel, as shown above right. These actions will launch any style sheet editor application associated in the file system with the .css file extension and open the style sheet in it.

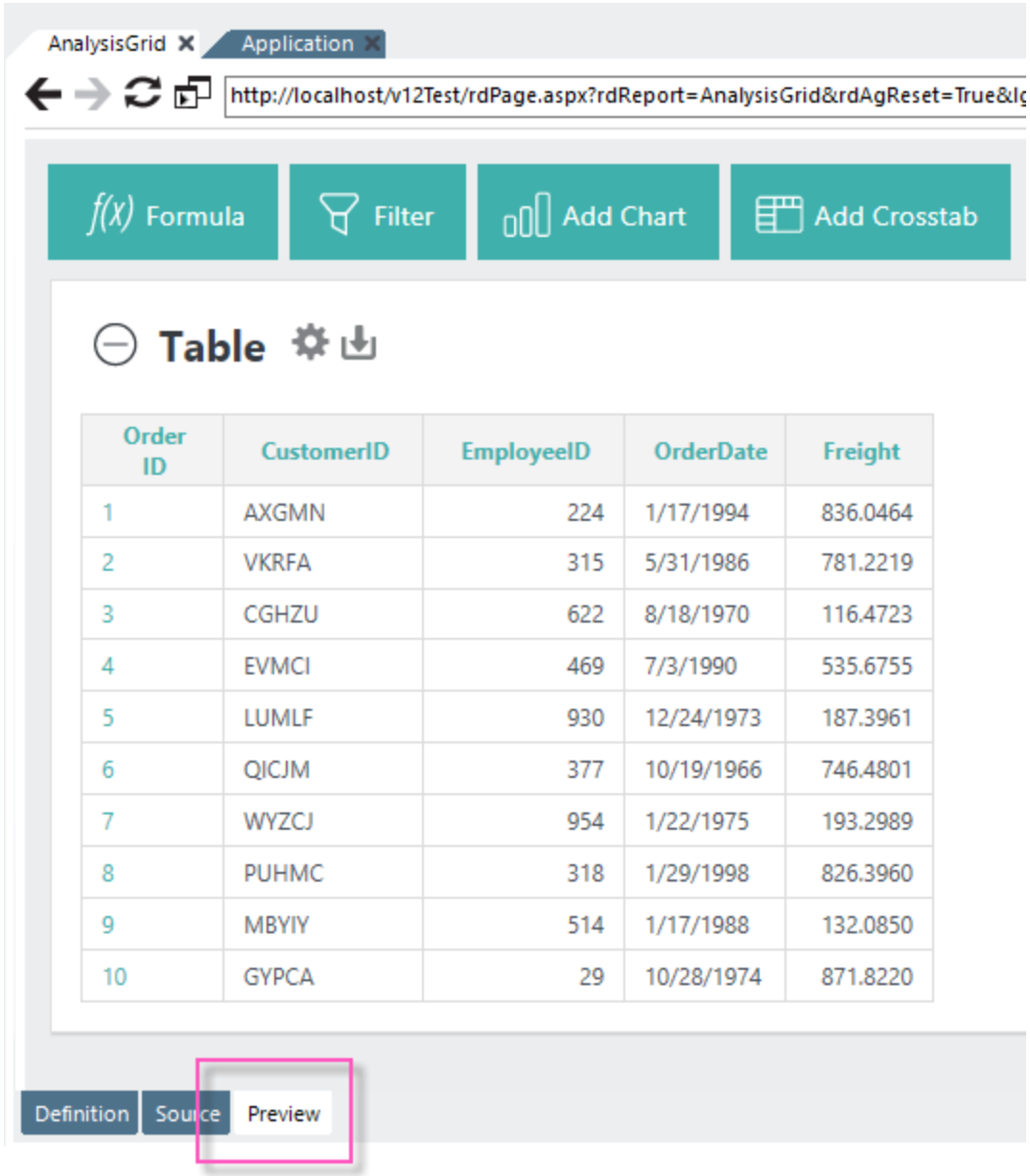
More information about working with style classes is available in *Style Sheets*.

Testing and Debugging Applications

While developing a Logi application, you'll want to periodically check your work by viewing it, and Logi Studio provides several ways to do this.

Using the Preview Tab

The **Preview** tab, at the bottom of the Workspace panel, provides the developer with a fast way to view the report definition currently being edited.

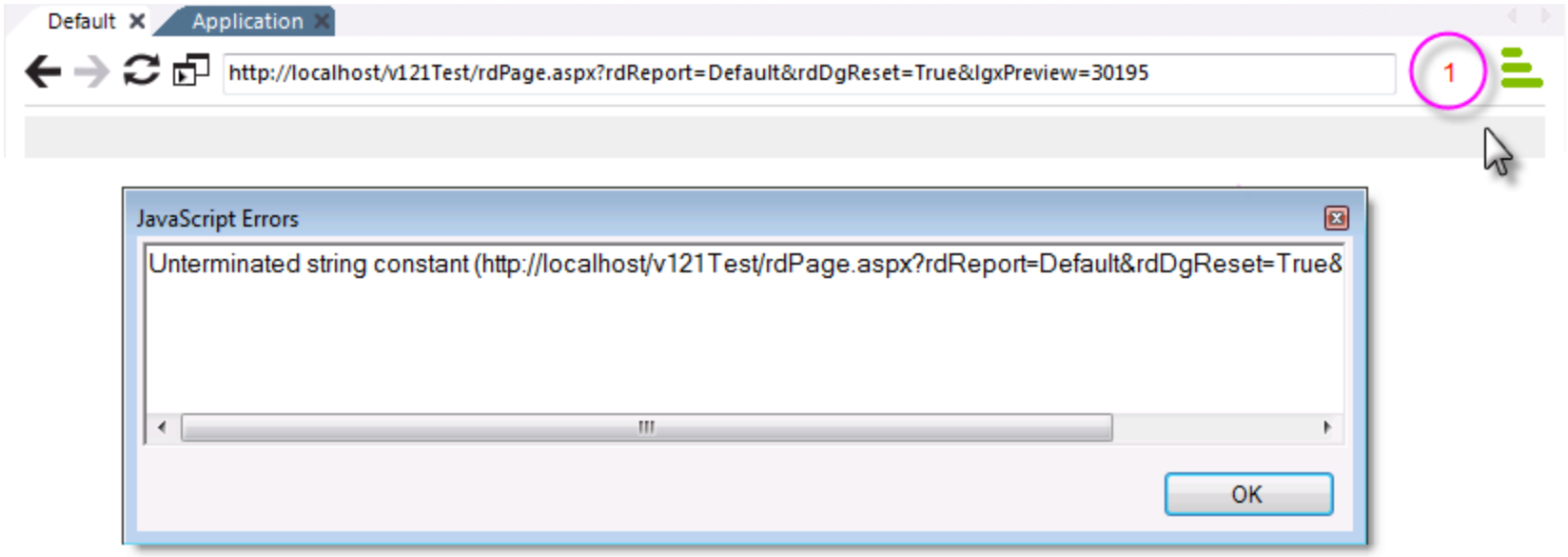

AnalysisGrid
Application
http://localhost/v12Test/rdPage.aspx?rdReport=AnalysisGrid&rdAgReset=True&lg
f(x) Formula
Filter
Add Chart
Add Crosstab
Table
Order ID
CustomerID
EmployeeID
OrderDate
Freight
1 AXGMN 224 1/17/1994 836.0464
2 VKRFA 315 5/31/1986 781.2219
3 CGHZU 622 8/18/1970 116.4723
4 EVMCI 469 7/3/1990 535.6755
5 LUMLF 930 12/24/1973 187.3961
6 QICJM 377 10/19/1966 746.4801
7 WYZCJ 954 1/22/1975 193.2989
8 PUHMC 318 1/29/1998 826.3960
9 MBYIY 514 1/17/1988 132.0850
10 GYPCA 29 10/28/1974 871.8220
Definition
Source
Preview

To preview a report definition that's open in the Workspace panel:

1. Ensure that the desired report definition is the selected definition in the workspace.
2. Click the **Preview** tab at the bottom of the Workspace panel, as shown above.
3. Browse and interact with the report definition using the toolbar at the top of the Preview panel.

Clicking Preview *will save* all open definitions before displaying the report.

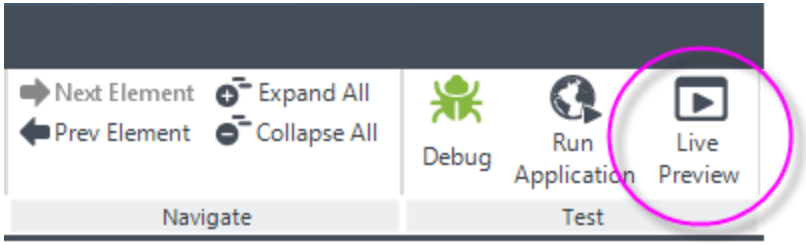
When using Preview, any JavaScript errors thrown will cause a count of the errors to appear next to the preview URL:



Click the number, as shown above, to display a dialog box containing the JavaScript error message. In earlier versions of Studio, the dialog box will simply be displayed automatically, interrupting execution.

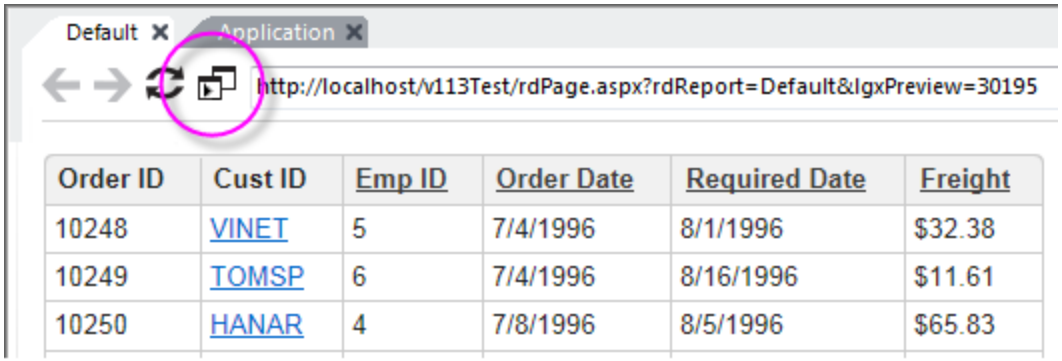
Using the "Live Preview" Feature

The **Live Preview** feature allows you preview your work in a separate, "live" desktop window.



The Main Menu's **Live Preview** item is shown above. The live preview will be displayed in a separate window and any changes made to the definition will immediately be reflected in that window. You can resize and relocate the new preview window as desired; you can even drag it onto a separate display, if you have one.

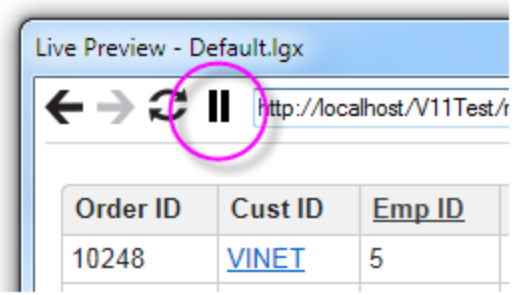
You can also switch to Live Preview from regular Preview mode:

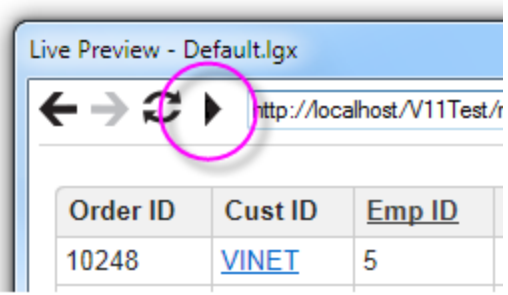


To start Live Preview, first preview the report definition, using the Preview tab, and then click the Live Preview icon, circled above.

The preview will be displayed in a separate window, and Studio will switch back to its Definition tab. You can resize and relocate the new preview window as desired; you can even drag it onto a separate display, if you have one.

Any subsequent changes you make to the definition code will immediately be reflected in the Live Preview window.



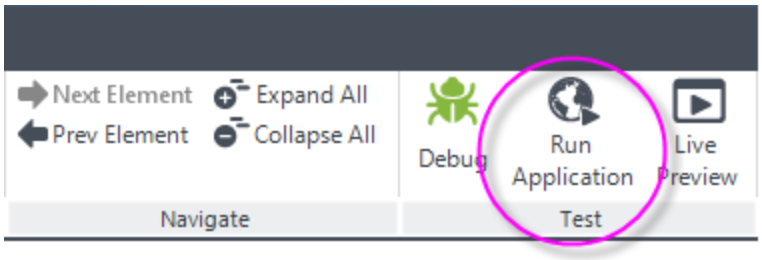


If you don't want definition changes to be updated immediately in the Live Preview window, you can pause and restart them, using the Pause and Play icons in the Live Preview window, circled above.

To exit Live Preview, just click the Live Preview window's **X** icon.

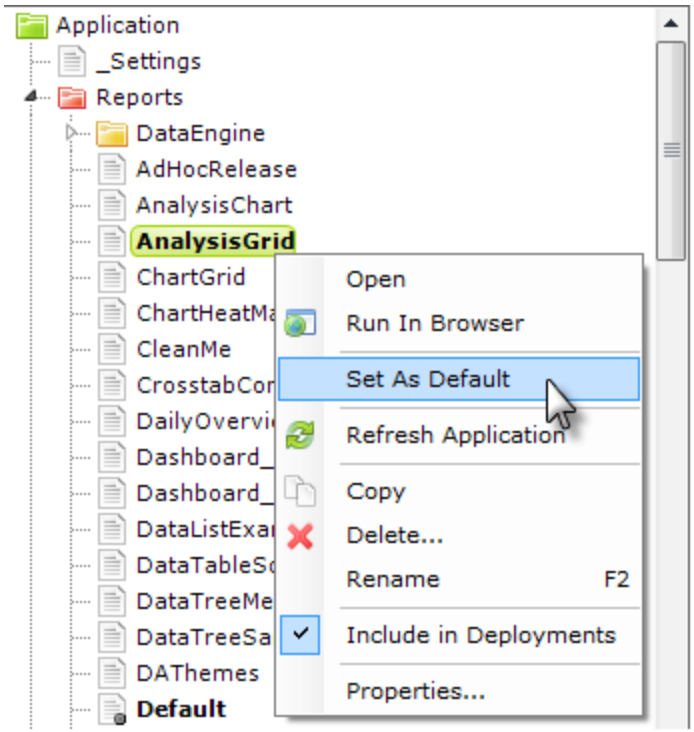
Browsing the Default Report Definition

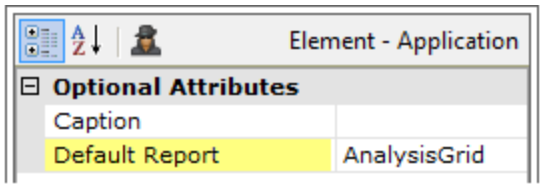
In a Logi application, one of the report definitions is designated (in the _Setting definition) as the "default" report - the page that will be shown if no definition is specified in the URL. If you don't change it, this is usually the **Default** report definition.



You can browse the default report by clicking the **Run Application** menu item, shown above, in Studio's Main Menu, or by pressing the **F5** key. Your computer's default browser (the one associated with .htm/.html file extensions) will be used for this.

You can change the "default" report designation using two methods:





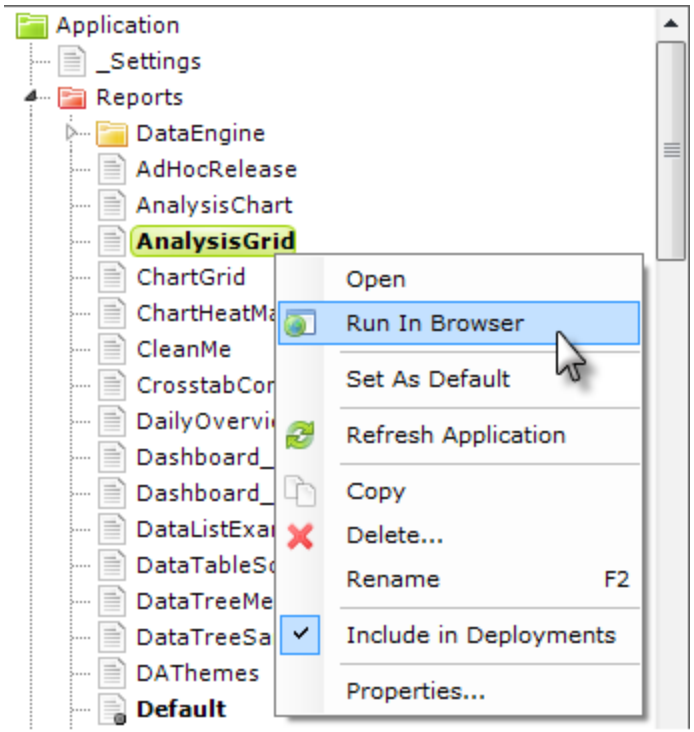
_Settings definition, Application element

As shown above, left, in the Application panel, select the desired report definition, right-click it, and select **Set As Default** from its context menu. This sets the **Application** element's **Default Report** attribute, shown above, right, in the _Settings definition. You can also manually edit this attribute value directly in the _Settings definition.

Using the Run Application menu item will save all open, unsaved definitions before displaying the report.

Browsing Any Report Definition

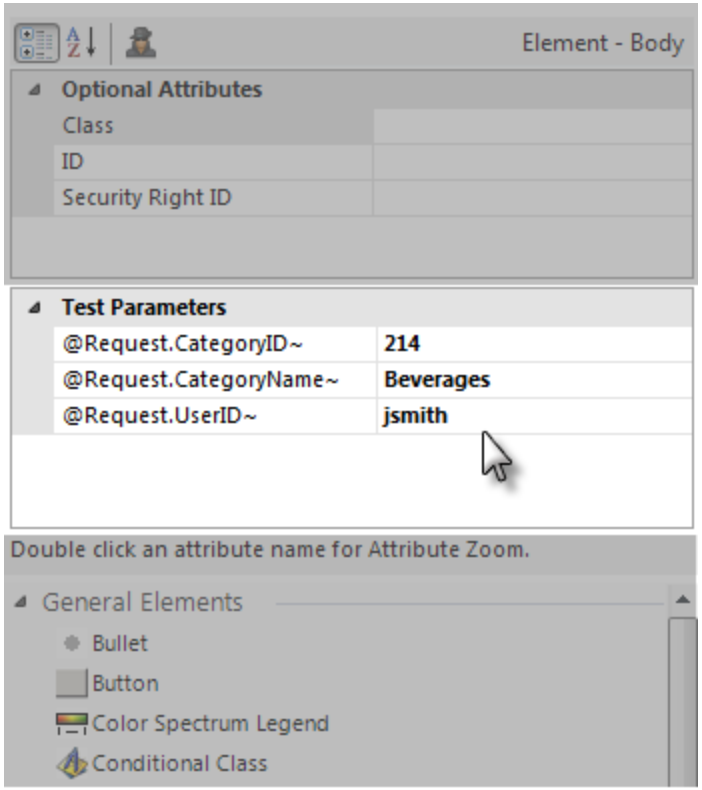
You can also browse any report definition *without* opening it in the Workspace panel, or setting it as the default report:



As shown above, select the desired report definition, right-click it, and select **Run In Browser** from the popup menu. Using this menu item will save all open definitions before displaying the report definition.

Using the Test Parameters Panel

One of Studio's least well-known panels is the **Test Parameters** panel. Studio makes this panel visible when the current definition expects to receive and use parameters. It provides a way for developers to supply request variable values for their definitions when previewing them in Studio, and it makes experimenting with different values very easy.

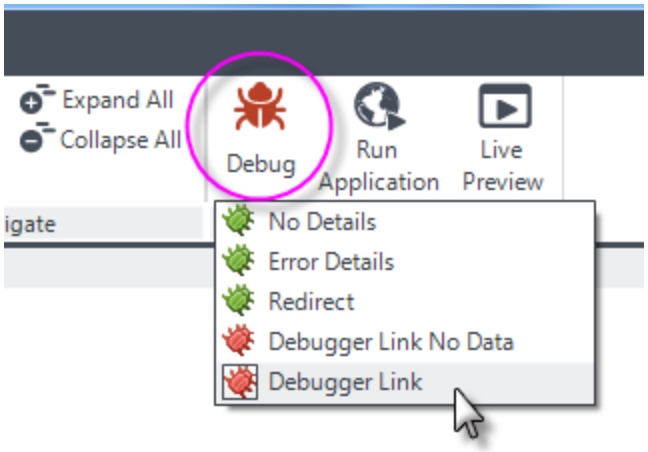


The example above shows this panel in use. Studio will automatically populate the left column with the tokens for any request variables expected by the currently selected definition in the Workspace. You can enter values for those tokens, and they'll be fed to the definition when you Preview it in the Workspace panel, or right-click it in the Application panel and select "Run in Browser".

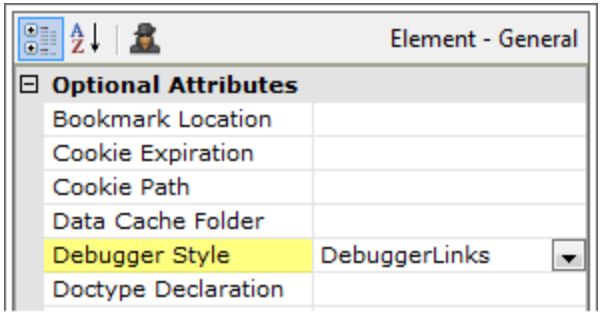
The token values in Test Parameters *are not* fed to the application when you click the **Run Application** menu item or press the **F5** key.

Using the Debugging Features

Logi Studio also provides **debugging** features for an application, but they have to be enabled to work. They are *not* enabled by default.



You can turn on comprehensive debugging for all your report pages by clicking the **Debug** menu item, shown above, and selecting *Debugger Link* from the selection list.

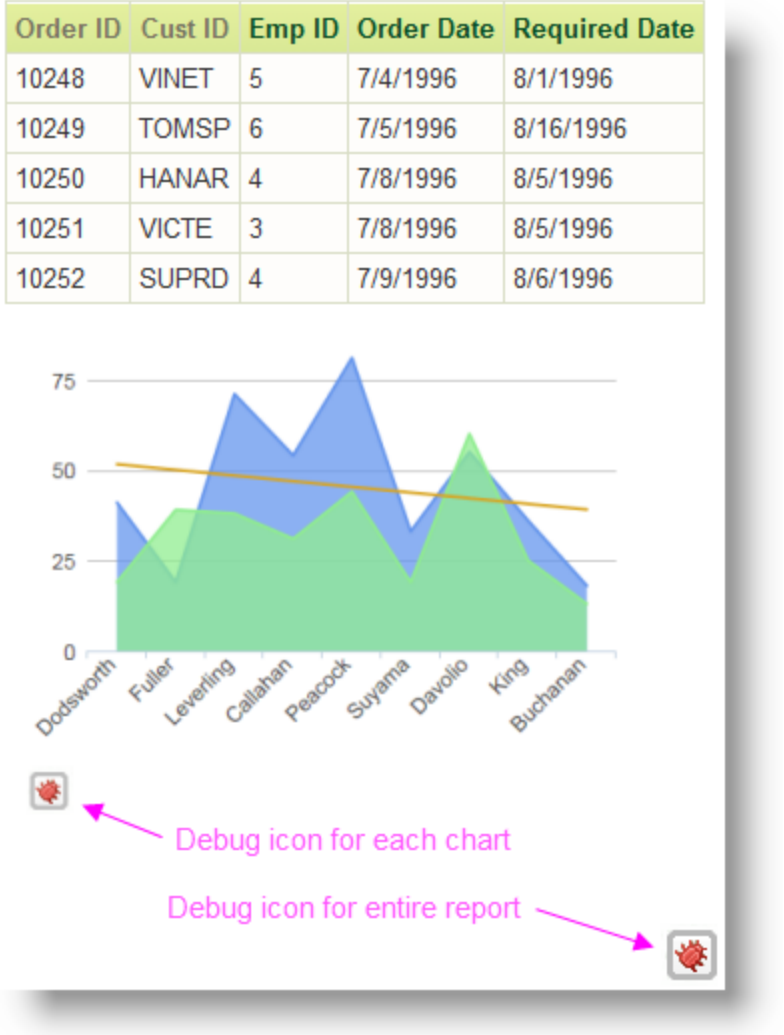


_Settings definition, General element

Selecting a debugger option in the menu sets an attribute value in the _Settings definition, as shown above. This value can also be set manually.

When the Debugger Style has been set to *DebuggerLinks* and the report is run, a **Debugger Trace Report** is generated. It can be accessed in three ways: using the debug icon which will now appear in your report pages, from a link that appears on a runtime error page, or by setting two attributes to pull runtime error and debug information into log files directly. The trace report appears either in the Preview panel or in your browser, depending on which method you were using to view your report.

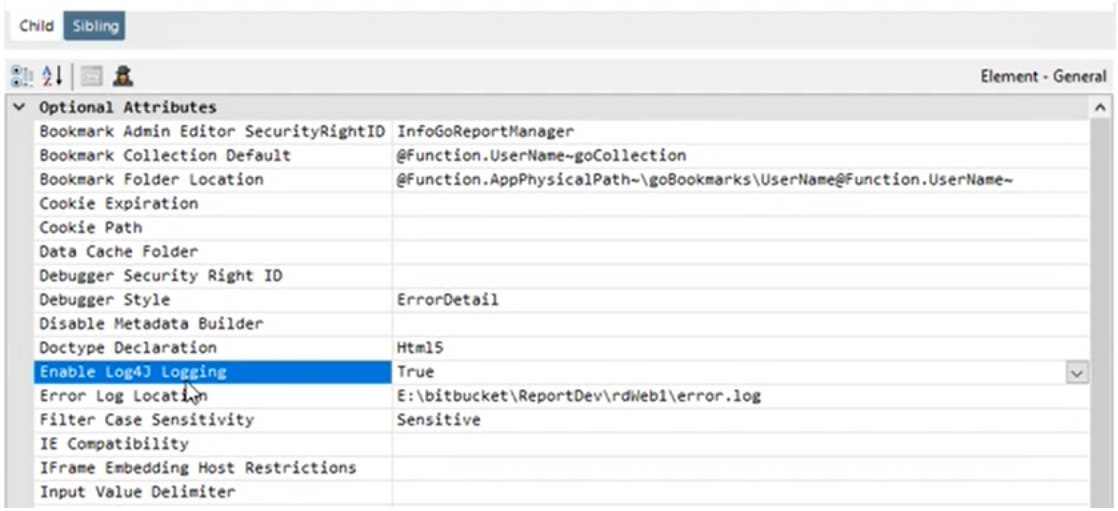
Order ID	Cust ID	Emp ID	Order Date	Required Date
10248	VINET	5	7/4/1996	8/1/1996
10249	TOMSP	6	7/5/1996	8/16/1996
10250	HANAR	4	7/8/1996	8/5/1996
10251	VICTE	3	7/8/1996	8/5/1996
10252	SUPRD	4	7/9/1996	8/6/1996



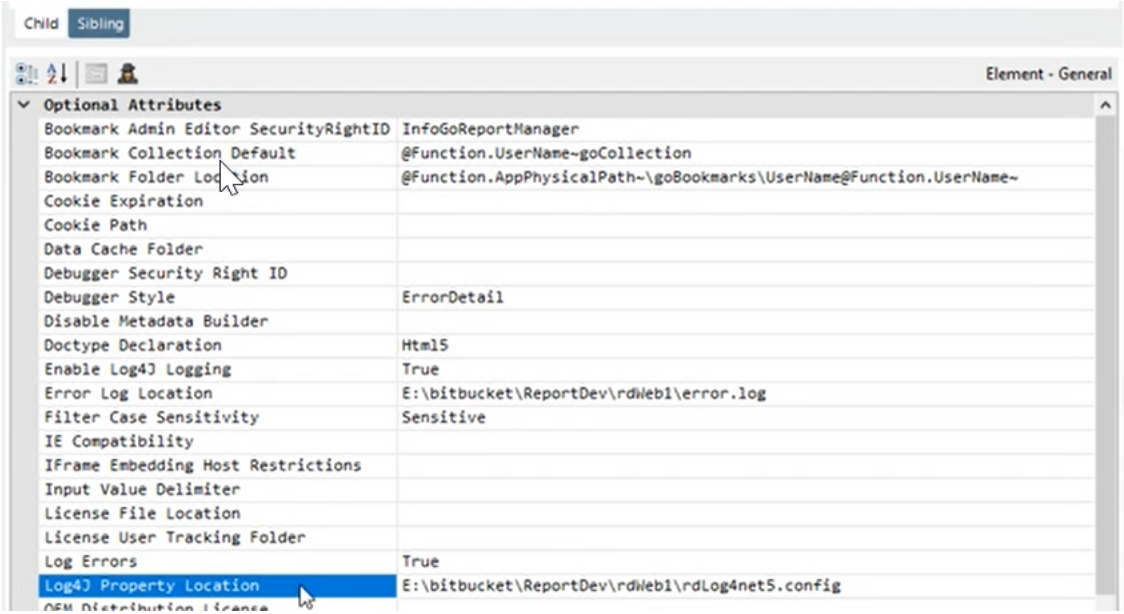
As shown in the example above, the debug icon for the report "floats" in the lower right-hand corner of the browser window, so you don't have to scroll down to use it. Any charts, which are processed separately by the Logi Engine, will have their own debug icon and trace report.

Developers can integrate modern logging framework using Log4Net and Log4J. You must update to Log4j v2.15 to mitigate this 0 day vulnerability. For more information about this vulnerability, refer to this statement.

To pull runtime error and debug information into log files directly, set the attribute `Enable Log4J Logging` to '*True*', like below:



Then, enter a file location in the `Log4J Property Location` attribute, as shown below:



Below is an example of a Log4J properties file:

```
#app logger option
log4j.rootLogger=INFO, file

# file
log4j.appender.file=org.apache.log4j.DailyRollingFileAppender
#log4j.appender.file.File=c:\\logs\\SSMJava12_6.log
log4j.appender.file.File=${catalina.base}/logs/<APP NAME>.log
log4j.appender.DatePattern='-'yyyy-MM-dd
#log4j.appender.file.MaxFileSize=10MB
#log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n


# Console - not referenced for rootLogger, so logs will not be displayed in console
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.SimpleLayout
```

Developers can configure the application to log different logging levels through the property file.

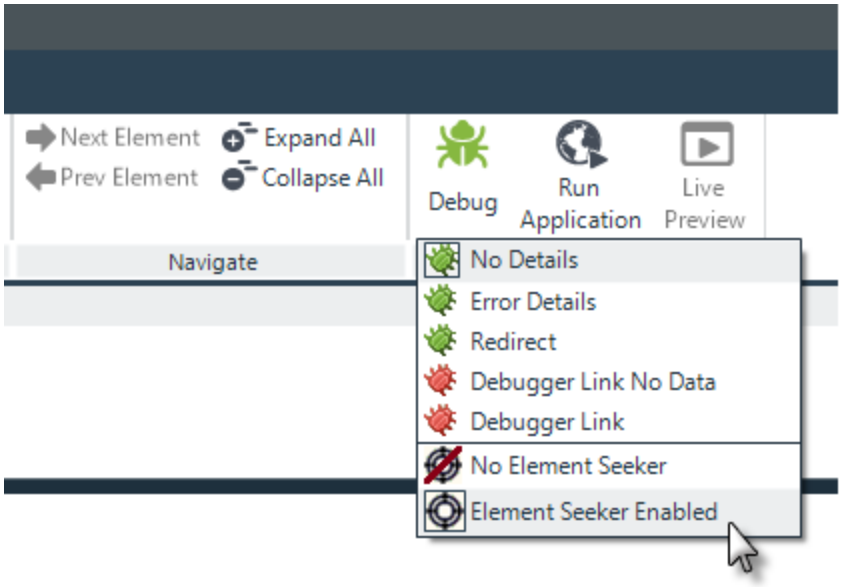
Log4J	Logi Info
DEBUG	Debugger No Data
INFO	Event Logging
WARN	Warning
ERROR	Error Details

Log4J	Logi Info
OFF	Disabled Logging

For more information about debugging, see *Debug Reports*.

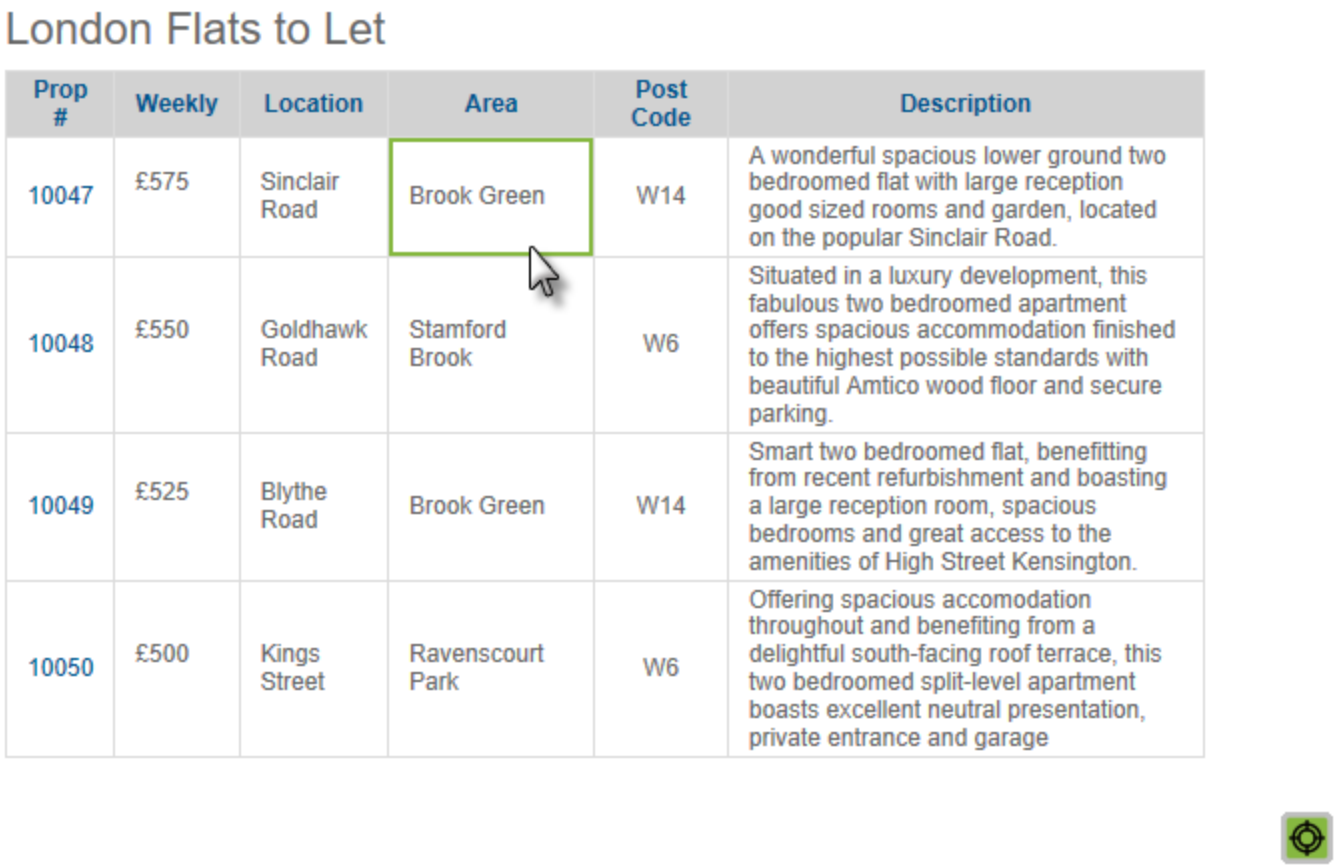
Using the Element Seeker

Logi Studio includes a feature, the **Element Seeker**, that lets you quickly locate an element in a definition file by selecting it in your preview or browser window. To use it, you must be working in Logi Studio v12.2 SP5+ and the Logi Server Engine version of your Logi application also has to be v12.2 SP5+. In addition, if you're using the Internet Explorer browser, you must be using IE 9 or later. Here's how it works:

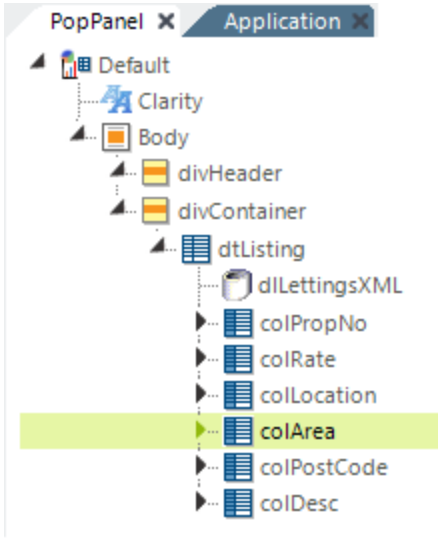


Seeker icon appears in lower right-hand corner of browser page

Enable the Element Seeker in Studio's Debug menu, as shown above. When you preview or browse a report page, you'll see the Seeker icon in the lower right-hand corner.



When you click the Seeker icon, it will turn green and enter selection mode. As you move your mouse cursor around the page, different regions will be framed in green, as shown above. Click a framed region...



... and Logi Studio will open the correct report definition and select the element responsible for the chosen region, as shown above.

To turn *off* the Seeker's selection mode, you'll need to refresh your preview or browser page.

Deploying Logi Applications

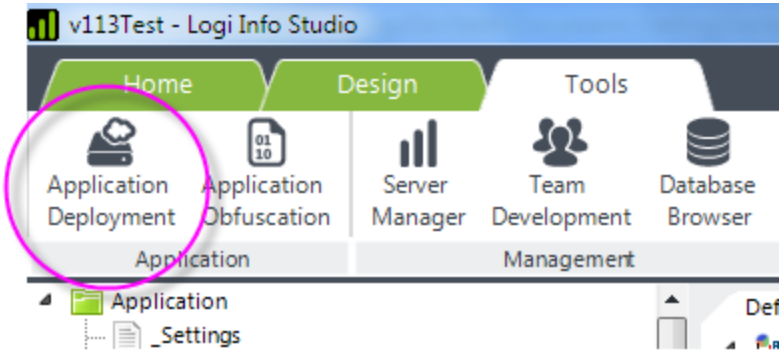
After you've developed your Logi application on your development machine, it's time to deploy it to your production server. There are several ways to this, and they're all discussed in *Deploy a Logi Application*.

If your application uses Discovery 3.0+ and you wish to preserve the Dataviews, visualizations, and other objects created during development, or if you're moving your application to another server and want to preserve the same objects, you'll also need to migrate the Platform Database. For more information, see *Deploy a Logi Application*.

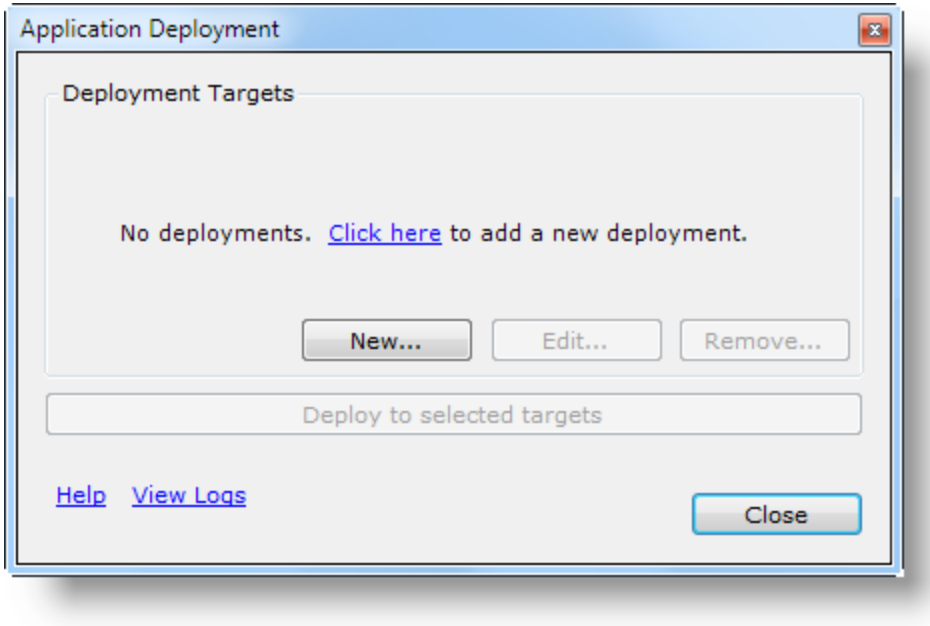
The easiest of these methods is to use Studio's **Application Deployment** tool. Before you use it, however, ensure that:

1. Logi Server is installed on your production server. On a development machine you generally install both **Logi Studio** and **Logi Server** and you may wish to install them both on a production server, but it's not required and most developers only install Logi Server there.
2. The .NET framework, or Oracle JDK orOpenJDK 8, 11, or 12, 13, or 14 has been installed on the production server.
   Oracle has changed its Java usage policies - see *Java Usage Policy* for important information.
3. You have an appropriate **Logi license** for the production server.
4. You have the appropriate **security credentials** for writing to the production server.
5. The production server has connectivity to the database server or other datasources needed by your application, and you know the security credentials they require.
6. You know any **storage conventions** for folder locations on the production server that you need to observe (such as, "all applications must be installed on the D: drive, not the C: drive")?

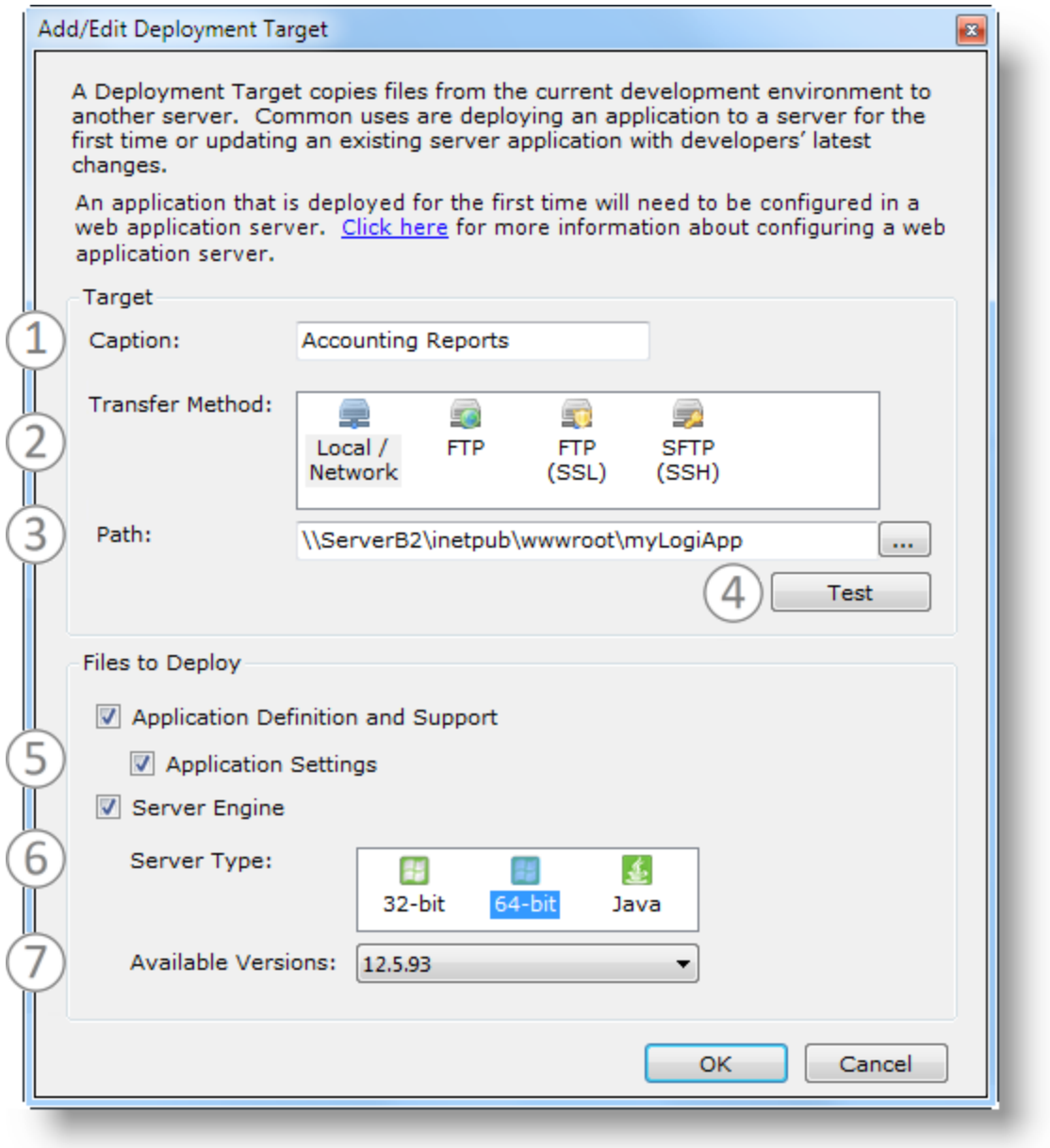
Studio's Application Deployment tool is capable of *copying* all or part of your application to the production server, using either file system copy commands to local or shared network drives, or one of three flavors of **FTP** (standard, SSL, or SSH). In Studio, the details of each operation are called a "deployment target" and they can be saved for later reuse.



The Application Deployment tool is started in Studio by selecting it from the Main Menu **Tools** tab, as shown above.



If no deployments have been defined, the dialog box shown above appears. Click the link or **New...** to define your first deployment target.



The Add/Edit Deployment Target dialog box contains the following:

1. **Caption**: Enter an arbitrary name for the deployment, for easy future reference.
2. **Transfer Method**: Select copy to Local or Network drive, FTP, FTP (SSL), or FTP (SSH).
3. **Path**: If Local/Network was selected in #2, enter the UNC file path to the destination app folder on the production server. For example:

   ```
   \\myWebServer\myLogiApp
   ```

   If an FTP variant was selected in #2, enter the FTP protocol **URL** to the destination app folder and arguments. Examples:

   For FTP & FTP SSL: `ftp://myWebServer/myLogiApp, ftp://myWebServer.com:8082/projects/myLogiApp`

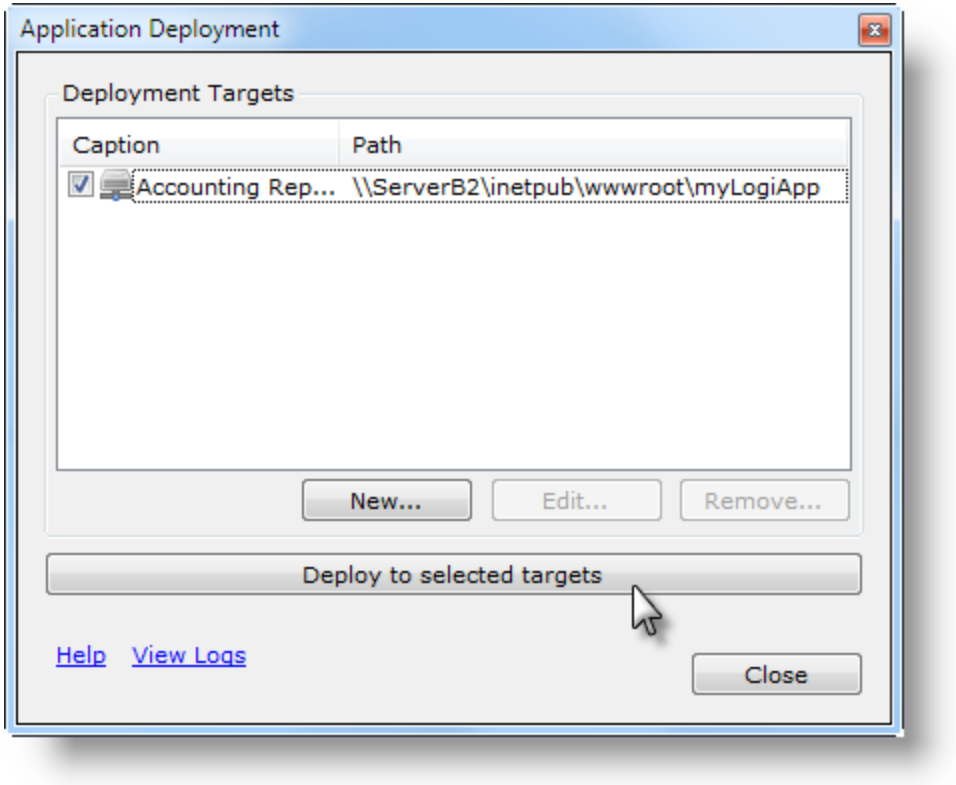
   For FTP SSH: `sftp://test.myServer.local//opt/tomcat/webapps/myLogiApp`
   Note use of *two* forward slashes after server name. Also you'll have to create the target application folder and grant permissions, as described in "Logi Java Application Folder Permissions" on page 100, because the application is not being deployed to a user's home directory.

   If necessary, you'll be prompted to supply a user ID and a password when the tool run; do not include them in the URL.

4. **Test**: This button can be used to ensure connectivity to the destination.
5. **File Types**: Select the file types that are to be deployed. *Files are overwritten at the destination without warning!*
6. **Server Type**: If "Server Engine" was selected in #5, these options are shown. Select the appropriate server type.
7. **AvailableVersions**: If "Server Engine" was selected in #5, available server engine versions are shown. Select the appropriate version.

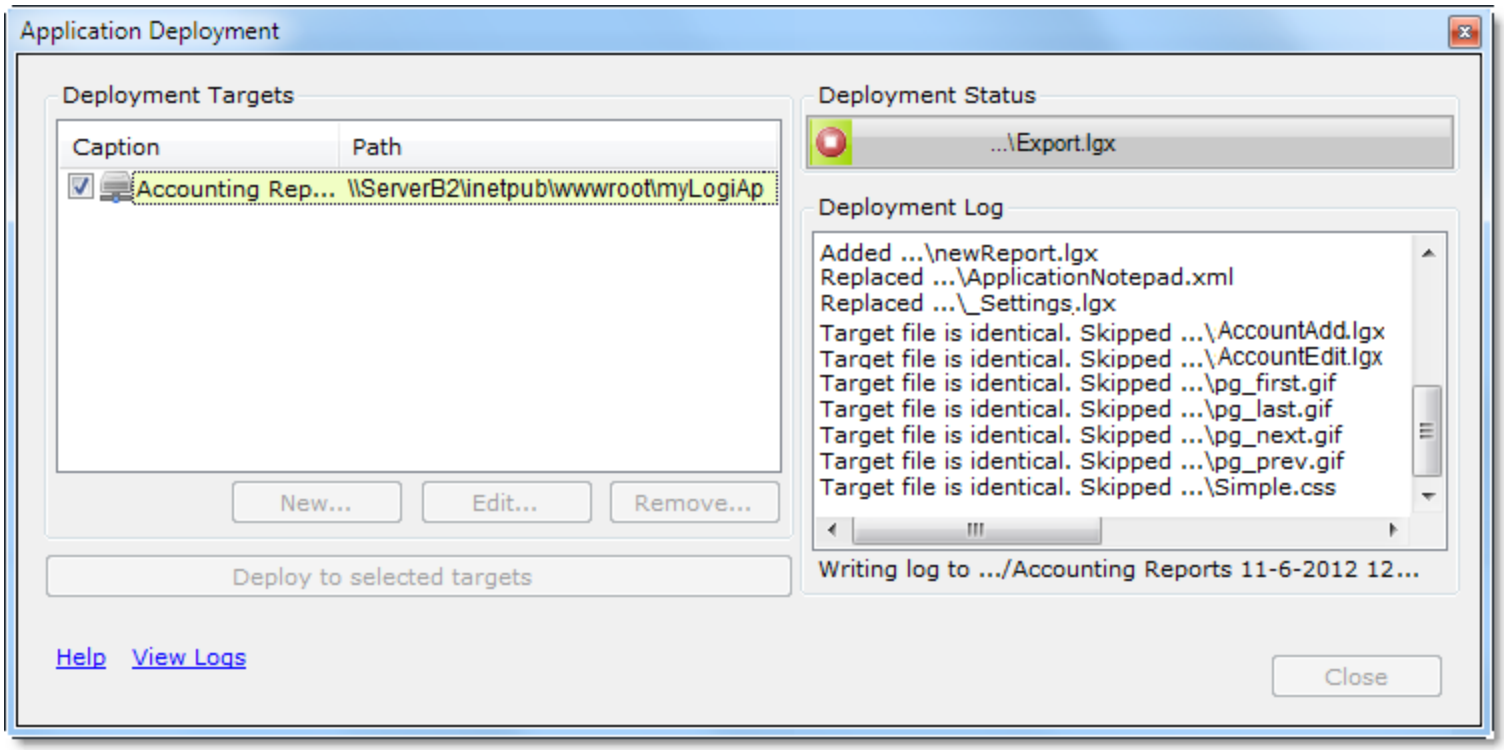
If the destination requires a user ID and password in order to access it, when you click the **Test** button, you'll be prompted for login credentials. If the credentials are successfully authenticated, the user ID will automatically be saved with the deployment target details and, optionally, the password as well.

Once a deployment target has been defined, it appears in a list...

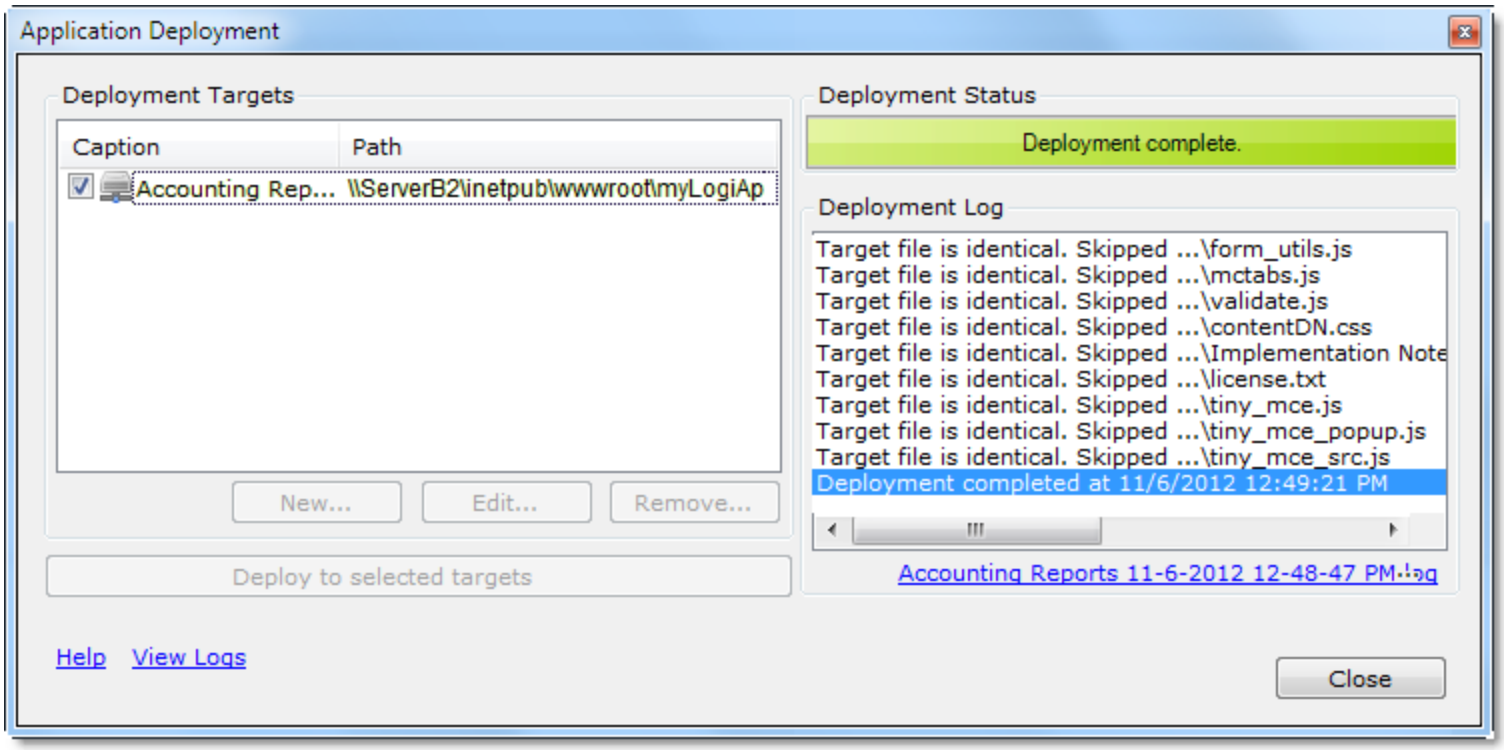


...as shown above, where it can be managed and edited. To run a deployment, check its check box and click the **Deploy** button. If multiple deployments are selected, they'll run consecutively.

The connection with the destination server will be established and then the files will be deployed to it.



The Deployment dialog box will expand to display the results of the process, as shown above. The running log indicates which files are added, replaced, or skipped. Deployment process logs are created in the rdDeployment folder beneath each Logi application folder as text files and are accessible through the **View Logs** link.



Finally, the deployment will complete and a direct link to its log file will be displayed.

It's very important that you understand that copying the files, using the Deployment tool, *does not register the application* with a web server on the target machine. So, after a first-time deployment is run, you must complete this registration on the target machine using Logi Studio, Server Manager (for .NET applications only), or the web server's management tools.

Logi Java Application Folder Permissions

After deployment of a Logi Java application, you can elect to grant 775 (full access) permissions manually on the application root folder and all of its child folders and files. If you prefer a more selective approach, typical Logi application folders require these permissions:

Folder	Permission
Logi app root	440
assemblies	
rdDataCache	660
rdDownload	660
rdTemplate	660
WEB-INF	
lib	
PhantomJS (binary)	550
_Definitions	
_SupportFiles	

Folder	Permission
goBookmarks (InfoGo app only)	660
(your custom folders)	660

Controlling Which Files Are Deployed

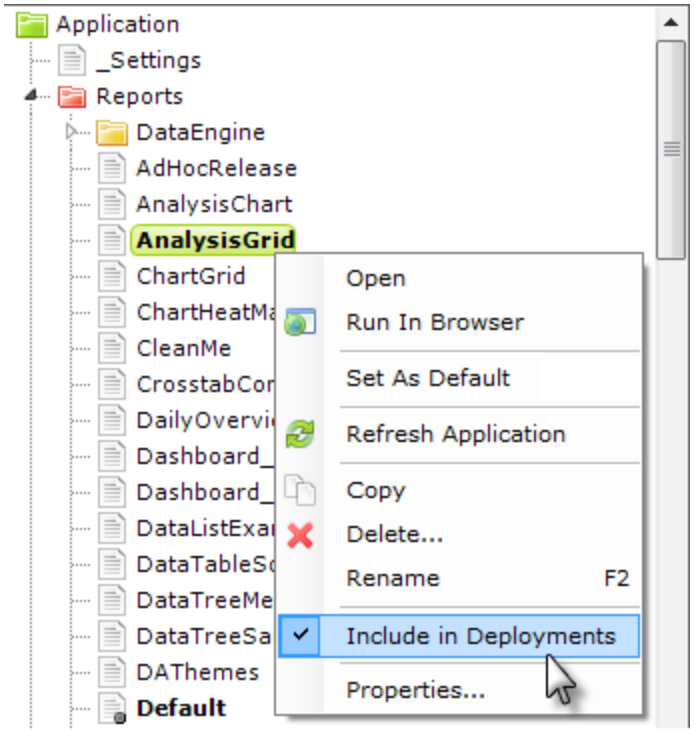
The Add/Edit Target dialog box allows you to select the files to be deployed by category:

Category	Description
Application Definition and Support	Copies all files in all folders with names beginning with an underscore, such as _Definition, _Script, _Support Files, etc. This includes folders like _Images and _Stylesheets, which are from a legacy folder scheme. Identical files will be skipped.
Application Settings	Copies _Settings.lgx only. Identical files will be skipped.
Server Engine (.NET app)	Copies the rdTemplate and bin folders - files are overwritten without warning. Also copies any .aspx, .asax, and .config files if they do not already exist; does not overwrite them if they do exist.
Server Engine (Java app)	Copies rdTemplate, Assemblies, and WEB-INF folders - files are overwritten without warning. Also copies any .aspx, .asax, and .config files if they do not already exist; does not overwrite them if they do exist.

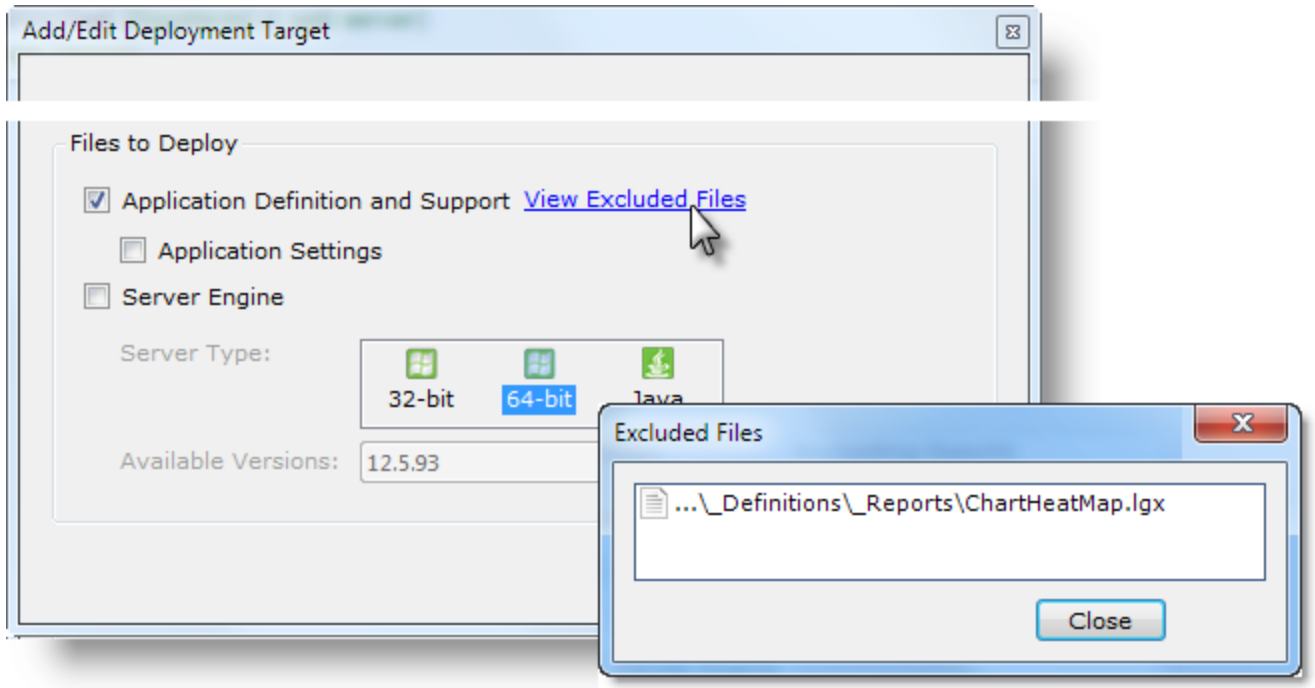
Why not just deploy *all* of the categories *all* of the time? You'll want to do so for your first deployment but, generally, you won't

want to for subsequent deployments in order to avoid overwriting files that may have been *customized* for the production server, such as _Settings.lgx. And, unless you're changing engine versions, deploying the engine files with every deployment is unnecessary and a waste of time. So the tool offers you the opportunity to tailor your deployments as needed.

However, developers may want to *exclude* specific application definition and support files from a deployment, and Studio provides an easy way to do that:



By default, all files in Studio's Application panel are included in deployments. However, if you wish to *exclude* a file, select it in the Application panel, right-click it, and uncheck the **Include in Deployments** item in the popup menu, as shown above.



If files *have* been excluded in this manner, in the bottom section of the Add/Edit Deployment Target dialog box, a **View Excluded Files** link will be displayed, as shown above. Clicking the link will display a list of the excluded files.

License Files

License files are *never included* in deployments, so you will have to manually deploy your license file the first time you deploy your application, or configure it to use a centralized license file. For more information about license files, see Product Licensing.

After a First-Time Deployment

After the first time an application is deployed, you must:

- Provide a license file for the application, unless you're using a centralized license file.
- Register the application with the web server.
- Ensure that the datasource connection strings or parameters, constants, and other server-specific settings are correct in the _Settings definition.
- For a Java app, complete any special configurations required by your web server.

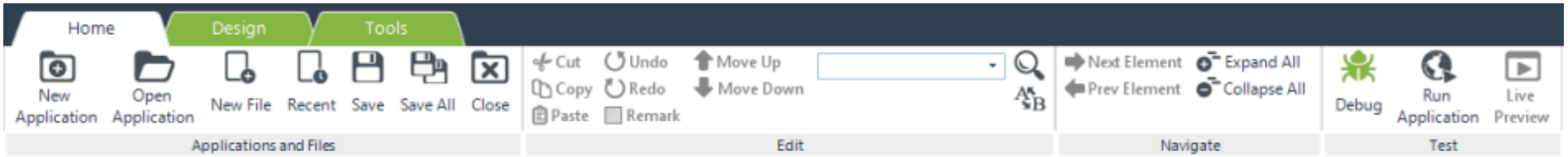
For subsequent deployments, you need only run the Deployment tool to update the files on the target server.

For more information, see Deploy a Logi Application.

Ribbon Menu, Shortcuts, and Search

Studio operations can be controlled using the **Ribbon Menu** at the top of the screen. Here are its tabs and major menu items:

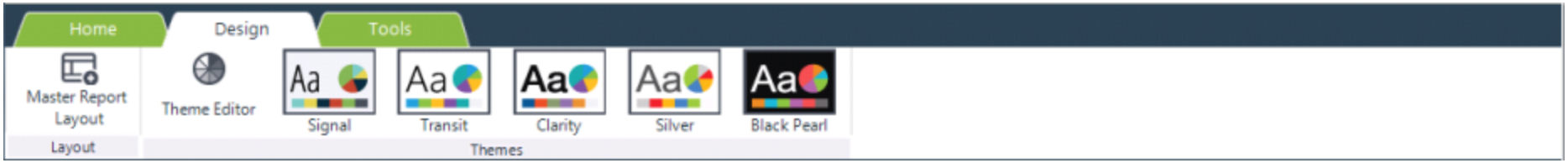
The Home Tab



Feature	Description
New Application	Create a new application with the assistance of the New Application wizard.
Open Application	Open an existing application from the File Explorer, the recent apps list, or a list of apps registered with the local web server.
New File	Create a new file by selecting a file category.
Recent	Re-open a recently closed file.
Save, Save All, Close	Save the file in the current Workspace editor tab, save all edited files, and close the current application, with a prompt to save any unsaved files.
Edit Actions,	Standard editing actions including Cut, Copy, Paste and more. Comment and uncomment elements and

Feature	Description
Remark, Move Up/Down	rearrange their order.
Search/Replace	Search current, or all, definitions and supporting text files, and replace text, if desired.
Element Navigation, Expand/Collapse	Navigate to previous or next element in the Element Tree navigation history. Expand or collapse all child elements beneath the selected element.
Test Actions	Control debugging, and preview a definition in a browser or in a separate "live" window.

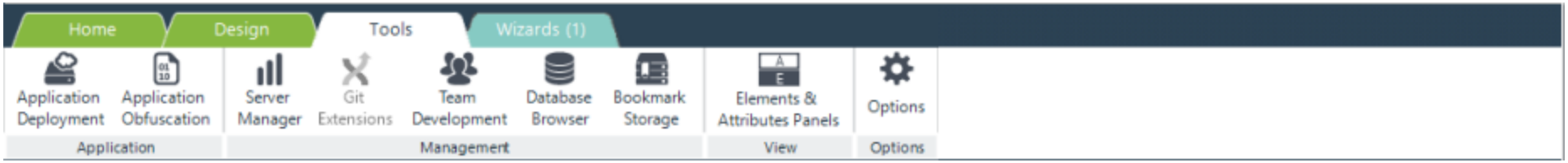
The Design Tab



Feature	Description
Master Report Layout	Launch the Master Report Layout wizard, which allows you to design a header-menu-footer template for use with each report definition. For more information, see *Master Reports*.

Feature	Description
Theme Editor	Launch the Theme Editor wizard, which allows you to create and edit your own custom themes. For more information, see *Using the Theme Editor*
Themes	Apply the selected standard theme to the current report, or to the entire application. *Other standard themes available in versions prior to 12.1 are no longer being included.*

The Tools Tab



Feature	Description
Application Deployment	Deploy the application to other servers using the Application Deployment Tool.
Application Obfuscation	Render definitions unreadable by humans to prevent tampering and theft using the Application Obfuscation Tool.
Server Manager	View registered applications and change their Logi Engine versions individually or in bulk.

Feature	Description
Git Extensions	Provides quick-and-easy access to Git, when it's being used as the application's source control repository.
Team Development	Manage built-in Logi source code control with file check-in/out and change history.
Database Browser	View schema and basic data samples from databases with connections in _Settings file.
Bookmark Storage	Create a SQL database table for Bookmark storage and migrate Bookmarks stored in the file system into it. For more information, see "Bookmarks" on page 265.
Dataview Authoring *Deprecated in Info v12.6+.*	Create, edit, and manage Dataviews - representations of data that can be used and re-used throughout an application. For more information, see *Dataview Authoring*.*Menu item is only visible when Discovery 3.x is installed.*
SuperWidget Authoring *Deprecated in Info v12.6+.*	Create, edit, and manage client-side components, such as charts and Dashboards, to create single-page applications that make use of Dataviews. For more information, see *SuperWidgets*. *Menu item is only visible when Discovery 3.x and SSRM 12.5+ are installed.*
Elements/Attributes Panel	Specify the arrangement and order of the Element Toolbox and Attributes panels in Studio.

Feature	Description
Options	Specify Studio options including file encoding and definition editor colors.

The Tools tab items **Application Deployment, Application Obfuscation, Server Manager**, **Team Development**, and the **Database Browser** are discussed more detail in separate sections in this topic.

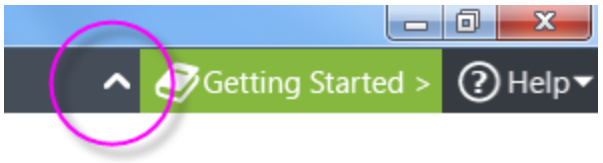
The Wizards Tab

The **Wizards** tab appears when an element that has supporting wizards is selected. The number of wizards available varies depending on the selected element.

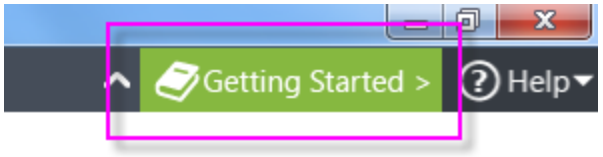
Some European keyboards use a key combination that includes the Alt key to produce characters like @ and ~ which are used frequently in Logi code. By default, the Alt key will shift focus to the Ribbon Menu, making it difficult to type these characters on European keyboards. An option to disable this focus shift is available in **Tools** → **Options**.

Ribbon Menu Controls

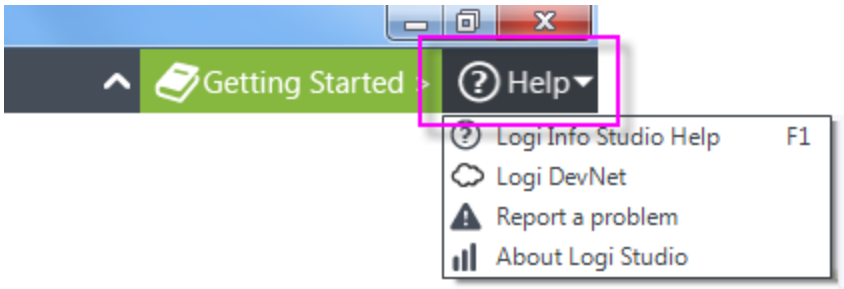
The following controls are located outside the tabs, at the right end of the Ribbon Menu:



The Ribbon Menu items can be *collapsed* vertically to get more screen real estate, using the arrow highlighted above. The menu can be expanded again by clicking the arrow again, or by selecting a different menu tab.



The menu's **Getting Started** button, shown above, will redisplay the Getting Started dialog box whenever needed.



The menu's **Help** button, shown above, displays a drop-down list of resources for getting assistance and reporting problems.

Keyboard Shortcuts

The following keyboard shortcuts are available for those who may prefer to save mouse clicks:

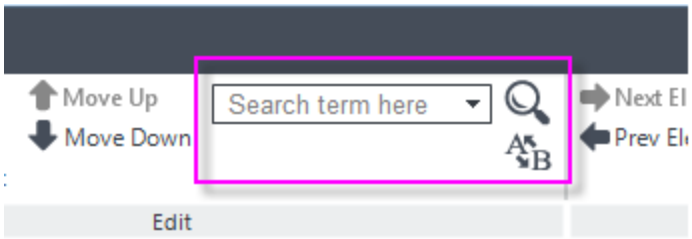
Application and Files	Editing
Ctrl n = New application	Ctrl f = Find
Ctrl o = Open application	Ctrl h = Replace
Ctrl l = Open application from local web server	Ctrl z = Undo
Ctrl s = Save selected file	Ctrl y = Redo
Ctrl, Shift s = Save all open files	Ctrl r = Remark/unremark
F1 = Studio Help	Ctrl x = Cut
F2 = Rename selected file	Ctrl c = Copy
F5 = Run current definition in browser	Ctrl v = Paste
F7 = Move element up (includes children)	Ctrl + = Next navigation history item

Application and Files	Editing
F8 = Move element down (includes children)	Ctrl - = Previous navigation history item
	Ctrl ↑↓ = With focus in Attribute panel and value committed, move to Prev/Next element in Element Tree

You can download and print our handy *Studio Keyboard Shortcuts* quick-reference card, if desired.

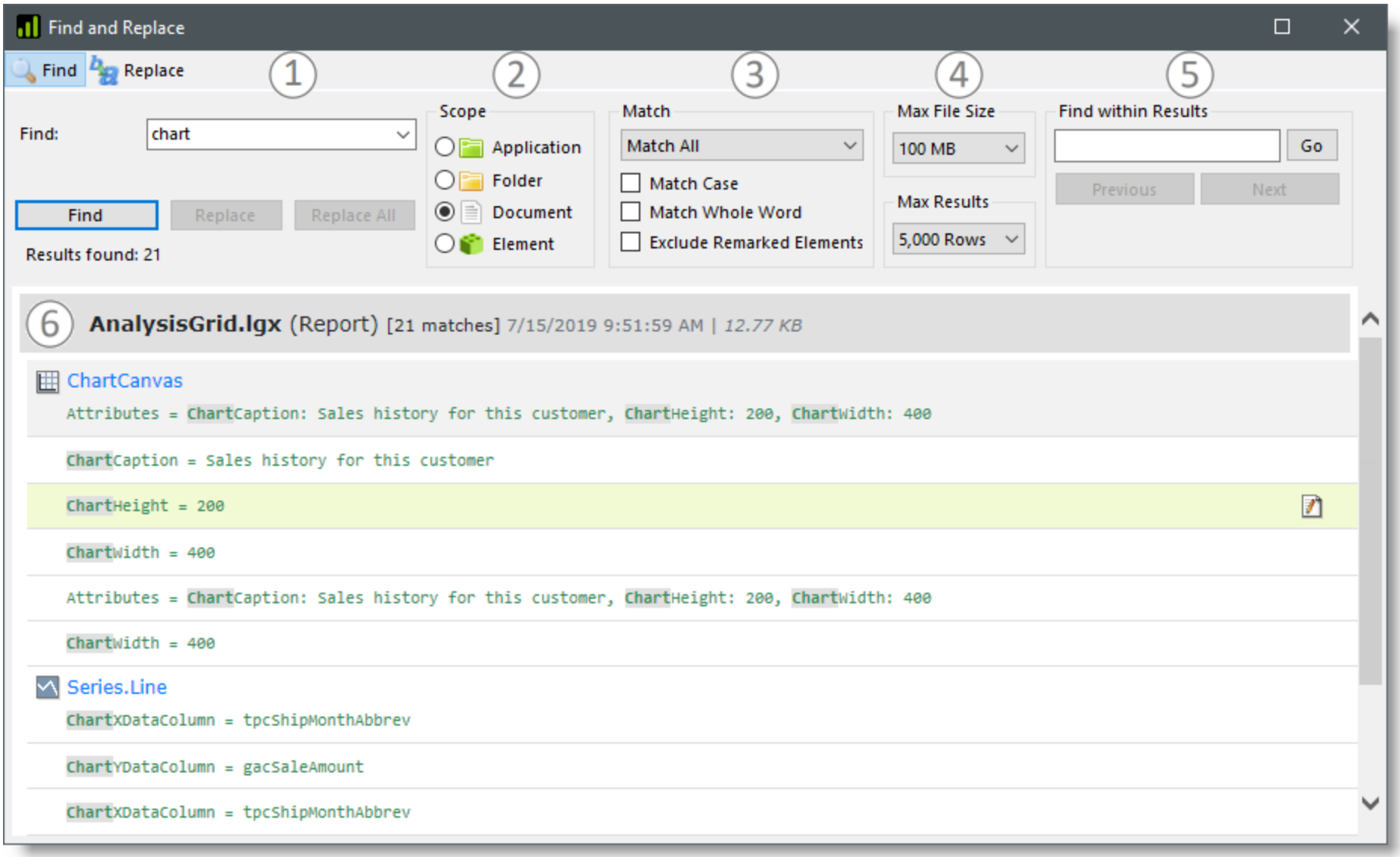
Search & Replace

Logi Studio includes a Search feature, including a **Search** control right on the main menu:



As shown above, the ribbon menu includes a text input control for entering new search terms and it keeps a historical list of terms

sought during the current program session. Clicking the magnifying glass icon starts a new search using the displayed terms. Searches are *not case-sensitive* initially and can also be initiated using the **Ctrl + F** keys. Clicking the A-B icon, or using the **Ctrl + H** keys, will open the Search and Replace feature.



Search results are displayed in the **Find and Replace** dialog box, shown above.

The search results have been modified to provide more flexibility and to mimic web search results.

Controls in the dialog box, keyed to the numbers at the top, are:

1. **Find / Replace** - Enter or select the desired search term and, if Replace is selected in the top menu, enter the term to replace it. Click **Find**, **Replace**, or **Replace All** to achieve the desired result.
2. **Scope** - Select the desired scope: to search the entire application or to limit the search to the current folder, the current definition or document, or the current element.
3. **Match** - Return results for all occurrences of the search term, for occurrences found in an element attribute value, or for occurrences found in element and attribute names. You can also filter results to match search term spelling case, to match only whole words, and to ignore elements that are remarked.
4. **Maximums** - Limit searches to files no larger than the selected maximum file size, or limit the result set to the selected number of rows.
5. **Find within Results** - Search within the results of the previous search.
6. **Results** - Search results are grouped by file. Click a result row link to open the file and view the related element; click the row to view the element, or click the pencil/paper icon at the right end of a row to open the Attribute Zoom window for a specific attribute.

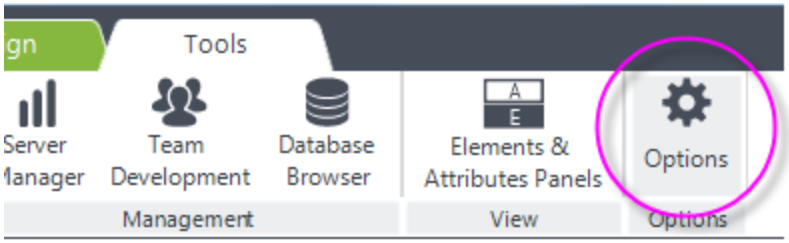
Additional searches can be launched from here with more selective scope and criteria, if desired. Results returned are listed by definition and are active links: clicking a result row will open the corresponding definition in the Workspace panel and highlight the specific occurrence in it.

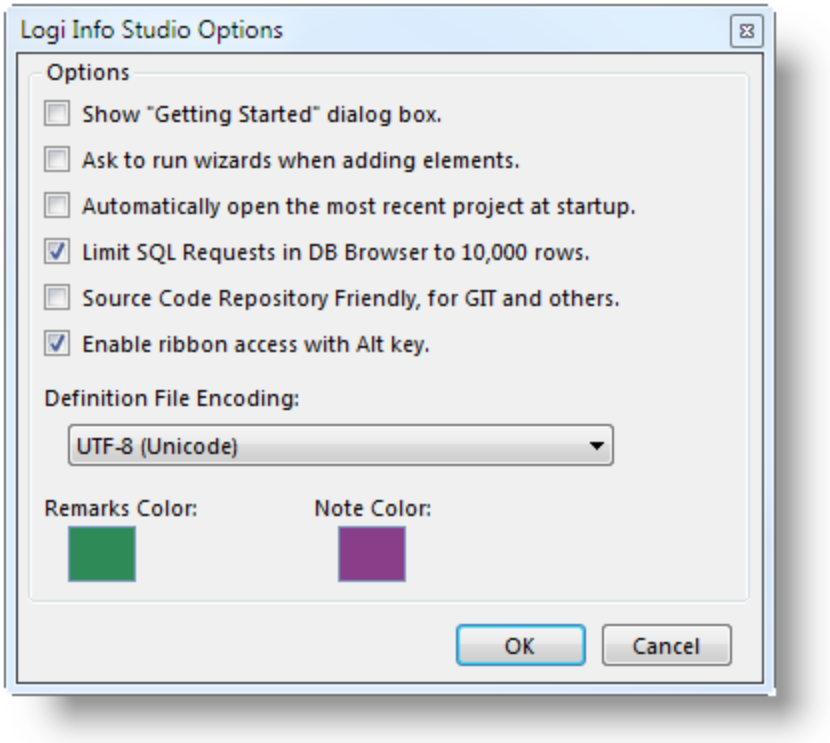
Clicking the **Replace** tab in the dialog box reveals controls in which you can enter a replacement term and selectively make replacements.

Studio Options

This topic introduces the settings that can be configured in Logi Studio.

Logi Studio can be configured using the Tools → Options menu item:





Options include:

- **Show "Getting Started" dialog box** - Select this option to display the dialog box designed for new users when Studio starts.
- **Ask to run wizards...** - Controls whether or not a dialog box prompts you to insert an element or run its wizard when you double-click an element in the Element Toolbox.
- **Automatically open...** - Select this option to have Studio, at startup, reopen that last application open when Studio was closed.
- **Limit SQL Requests**... - Uncheck this option to allow very large result sets to be returned, but be prepared for possible delays.
- **Source Code Repository Friendly** - When checked, prevents writing of the standard `SavedAt`, `SavedBy`, and `EngineVersion` attributes in definition file source code, which can interfere with some code repositories like GIT. Also adds `.gitignore` and `.tfignore` files in the application folder and makes formatting and other changes to the source code files. See the following section for more information.
- **Enable ribbon access**... - Uncheck this option to prevent shifting focus to the Ribbon Menu when the Alt key is pressed. This is useful with some European keyboards that use key combinations that include the Alt key to produce characters like @ and ~.
- **Definition File Encoding** - Specifies the character encoding scheme to be used in Logi definition files.
- **Remarks Color** - Colored block indicates the color to be used for remarked (commented) elements in the element tree. Click the block to open the Color Selector dialog box and choose a new color.
- **Notes Color** - Colored block indicates the color to be used for **Note** element text in the element tree. Click the block to open the Color Selector dialog box and choose a new color.

Source Code Repository Friendly Features

Logi Studio v12+ includes features designed to make Logi definition source code easier to use with 3rd-party source code repository and control products like GIT and TFS. In addition to changing the source code format, the features also create `.gitignore` and `.tfignore` files in the application folder.

 If you enable these features your source code *will* be reformatted. However, it *is* reversible.

```
<?xml version="1.0" encoding="utf-8"?>
<Report ID="MyDefault" SavedBy="LOGIXML\lee" SavedAt="4/2/2015 3:43:55 PM" EngineVersion="12.0.029">
<StyleSheet Theme="Signal" />
<Body>
<DataTable ID="dtOrders">
<DataLayer Type="SQL" Source="SELECT * FROM Orders" ConnectionID="connNW" />
<DataTableColumn ID="colOrderID" Header="OrderID">
<Label ID="lblOrderID" Caption="@Data.OrderID~" />
<DataColumnSort DataColumn="OrderID" DataType="Number" />
</DataTableColumn>
</DataTable>
</Body>
<ideTestParams />
</Report>
```

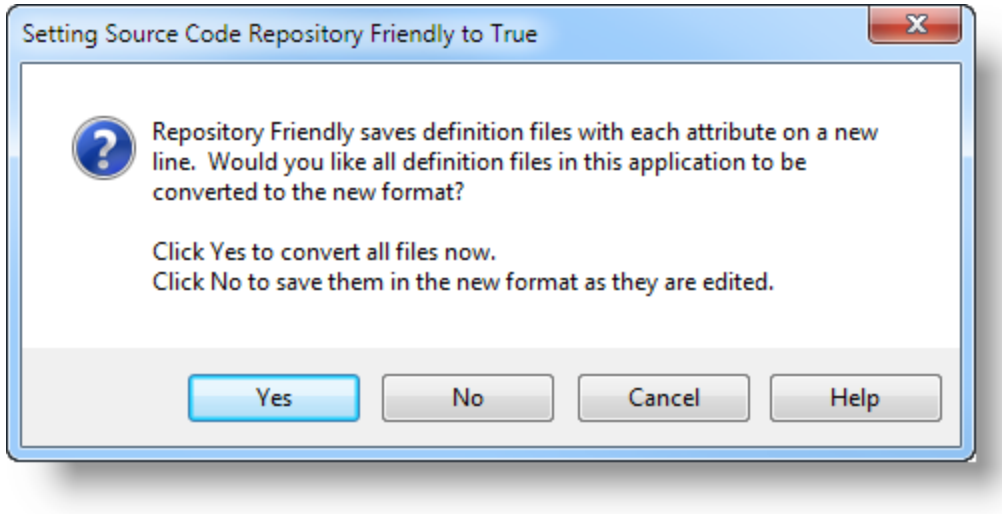
The example above shows the standard (non-"repository friendly") definition source code format. It includes "SavedBy",

"SavedAt", and "EngineVersion" information and XML elements and their attributes appear on the same line. Also, special attributes are provided (but not shown here) to describe the "collapsed state" of the elements in Studio's Element Tree.

```
<?xml version="1.0" encoding="utf-8"?>
<Report
ID="MyDefault"
>
<StyleSheet
Theme="Signal"
/>
<Body>
<DataTable
ID="dtOrders"
>
<DataLayer
ConnectionID="connNW"
Source="SELECT * FROM Orders"
Type="SQL"
/>
<DataTableColumn
Header="OrderID"
ID="colOrderID"
>
<Label
Caption="@Data.OrderID~"
ID="lblOrderID"
```

```
/>
<DataColumnSort
DataColumn="OrderID"
DataType="Number"
/>
</DataTableColumn>
</DataTable>
</Body>
</Report>
```

The same code is shown above after being reformatted to be "repository friendly". It has no timestamp, author, or engine version information and each XML attribute is displayed, in alphabetic order, on its own line. SQL queries will similarly be shown on multiple lines, making them easy to read. Any collapsed state attributes are saved in a separate file: `_Definitions\Settings.lgp`.



When you check or uncheck the Source Code Repository Friendly box in the Studio Options, you'll see the dialog box shown above.

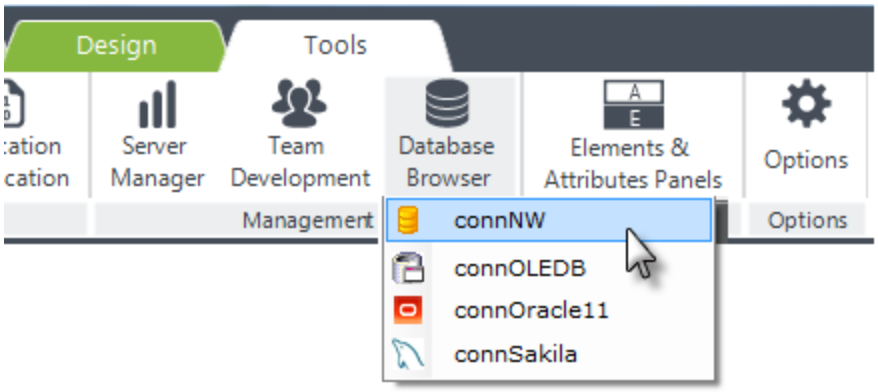
- If you click **Yes**, *all* of your definition files will be converted *immediately* to the other format and saved.
- If you click **No**, then your files will be individually converted the next time you edit and save them.
- Click **Cancel** to make no changes.

Checking or unchecking the box sets the General element's **Repository Friendly** attribute to *True* or *False*, in the _Settings definition. When Studio's New Application wizard is used to create a new application, this attribute is set to *True* by default.

You can change all, or individual, definitions back and forth between formats without concern. *The format has no effect on the function or performance of a definition when it's executed.*

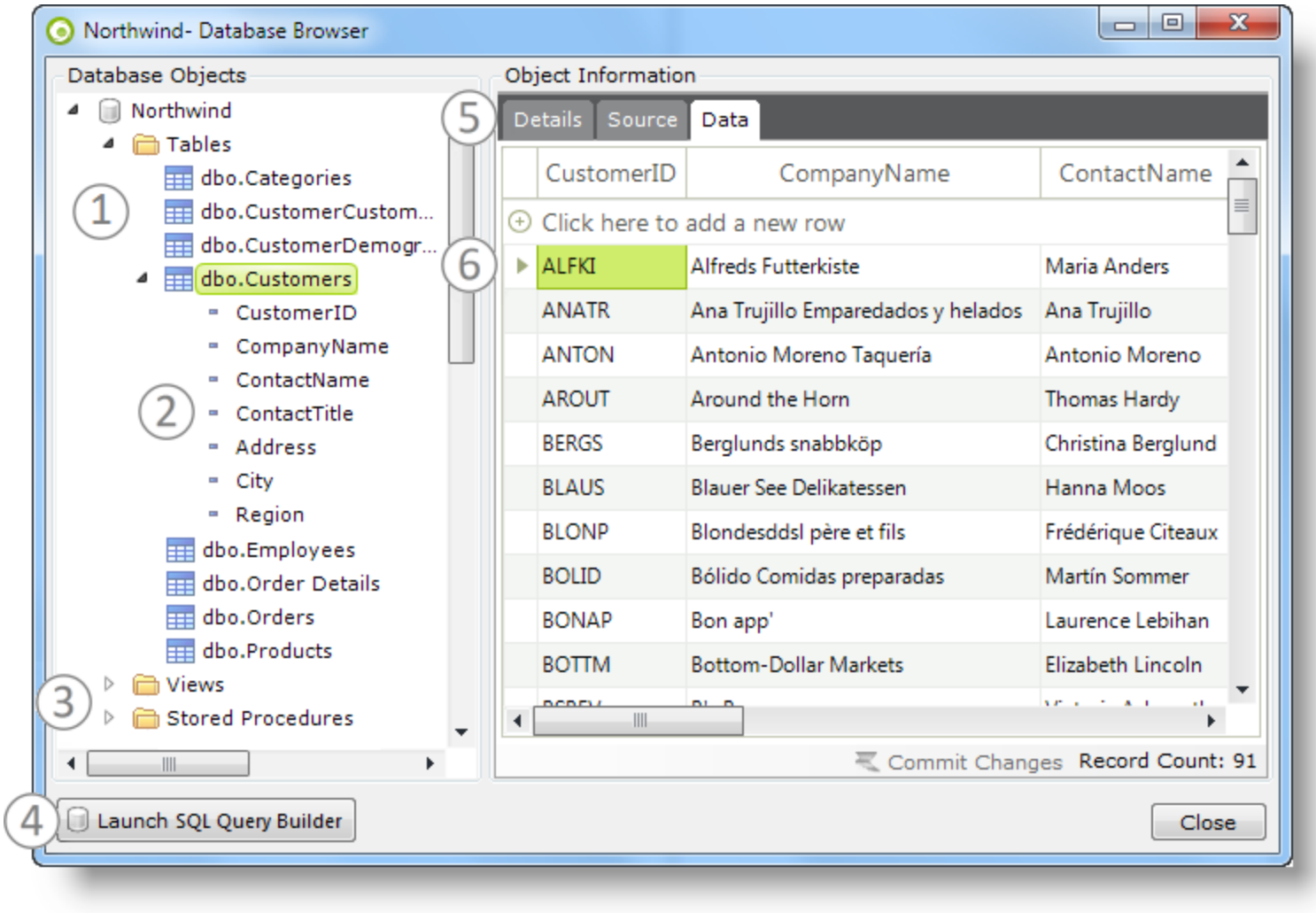
The Database Browser

The **Database Browser** tool allows developers to view the **objects** and **data** in connected databases. The objects available include Tables, Columns, Views, and Stored Procedures. The database browser *doesnot* work with datasources that do not return a *schema* when queried, such as XML data files.



The Database Browser is launched from the main menu's Tools tab, as shown above. Select a database from the list of connections that have been configured in the _Settings definition for this application.

If you select a Connection element in the _Settings definition and click the Database Browser menu item, it will open the database immediately. If no Connection element is selected, then a list of databases like the one shown above will be displayed so you can select one.



The Database Browser, shown above, provides the following features:

1. A list of the **Tables** available in the database is presented, as shown above.
2. Each table icon can be expanded to display its **Columns**.
3. The database's **Views** and **Stored Procedures**, if any, are also available.

4. The **Query Builder** tool can be launched from here with this button.
5. The **Details** tab provides table or column properties, the **Source** tab allows Stored Procedure code to be viewed (but not edited), and the **Data** tab displays the table's actual data, limited to 10,000 rows.
6. Rows of data displayed in the Data tab can be right-clicked and **edited** or **deleted**.

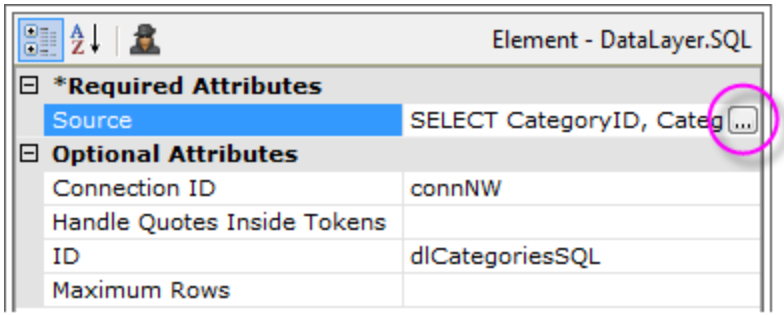
The Database Browser *will* work with the databases for which we provide "native" Connection elements, such as Oracle, MS SQL Server, and MySQL. It *may* work with other databases that use a connection that mimics that of a database for which we provide a native connection. It is *notguaranteed* to work with *every* database and, in that case, we recommend that you use tools provided with the database to examine and manipulate it outside of Studio.

The Database Browser may not support all of the functions and views available in Microsoft Access databases.

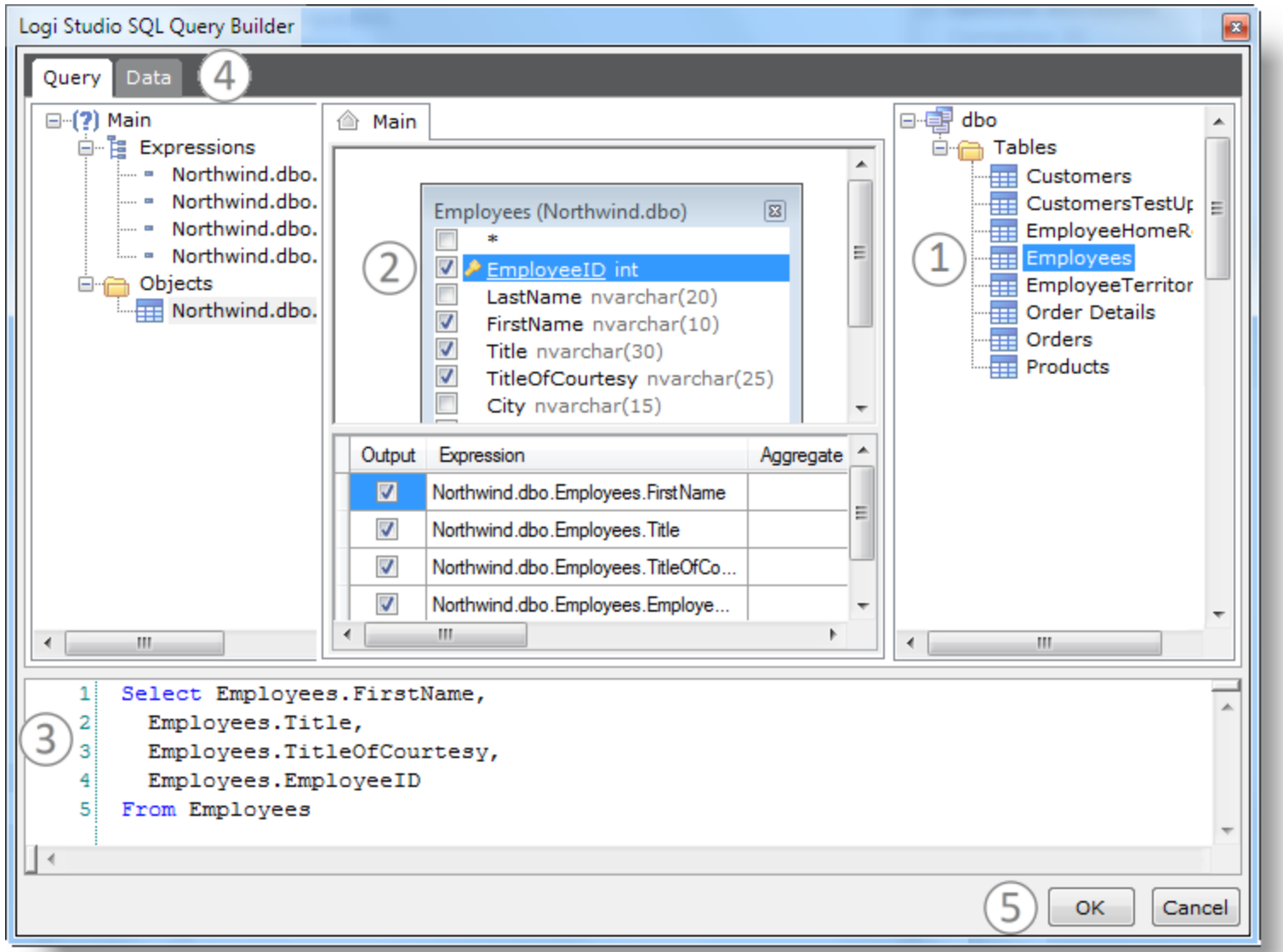
By default, the Database Browser will only return 10,000 rows of data. You can turn this limit off in Studio in Tools → Options. The Database Browser is a **non-modal dialog box** and can remain open, for reference, while you work elsewhere in Studio.

The SQL Query Builder

The **SQL Query Builder** tool allows you to examine data, and construct and test your queries, from within Studio. It's launched either from the **Database Browser** or...



...as shown above, by clicking the browse button at the end of the **DataLayer.SQL** element's **Source** attribute.


Logi Studio SQL Query Builder
Query
Data
4
Main
Expressions
Northwind.dbo.
Northwind.dbo.
Northwind.dbo.
Northwind.dbo.
Objects
Northwind.dbo.
Main
Employees (Northwind.dbo)
*
EmployeeID int
LastName nvarchar(20)
FirstName nvarchar(10)
Title nvarchar(30)
TitleOfCourtesy nvarchar(25)
City nvarchar(15)
2
Output
Expression
Aggregate
Northwind.dbo.Employees.FirstName
Northwind.dbo.Employees.Title
Northwind.dbo.Employees.TitleOfCo...
Northwind.dbo.Employees.Employe...
dbo
Tables
Customers
CustomersTestUp
EmployeeHomeR
Employees
EmployeeTerritor
Order Details
Orders
Products
1
Select Employees.FirstName,
Employees.Title,
Employees.TitleOfCourtesy,
Employees.EmployeeID
From Employees
3
5
OK
Cancel

The SQL Query Builder presents **database objects** in a (1) list, from which tables and views can be **dragged** into the (2) center **work area**. Manipulating the visual objects results in a SQL query being generated in the (3) bottom command area, where it can also be manually edited. The query can use **multiple lines** in the command area and can include DECLARE statements and other more complex SQL statement constructions.

Clicking the (4) **Datatab** at the top of the SQL Query Builder **executes** the SQL statements and displays the results in that tab panel. The **F5** key can also be used to execute statements; portions of statements can be highlighted and selectively executed using F5 as well. SQL statements such as INSERT and UPDATE can be used to modify the data but there is no graphical way to directly make modifications in the Data tab. By default, the SQL Query Builder will only return 10,000 rows of data. You can turn this limit off in Studio in Tools → Options.

Clicking (5) **OK** will cause the query constructed in the command area to be copied into the datalayer's **Source** attribute, if the SQL Query Builder was called by clicking the attribute's browse button. Otherwise, the query text can be selected, copied, and pasted as needed.

- The SQL Query Builder *will* work with the databases for which we provide "native" Connection elements, such as Oracle, MS SQL Server, and MySQL. It *may* work with other databases that use a connection that mimics that of a database for which we provide a native connection.
- The SQL Query Builder *will not* support all of the functions and views available in **Microsoft Access** databases, nor does it support other non-standard SQL languages, such as *Salesforce Object Query Language*. It is *notguaranteed* to work with every possible database and, in that case, we recommend that you use tools provided with the database to create and test queries, and then copy and paste them into Studio.

When a Query Has Been Entered Manually

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

Use with Oracle and Multiple Schemas

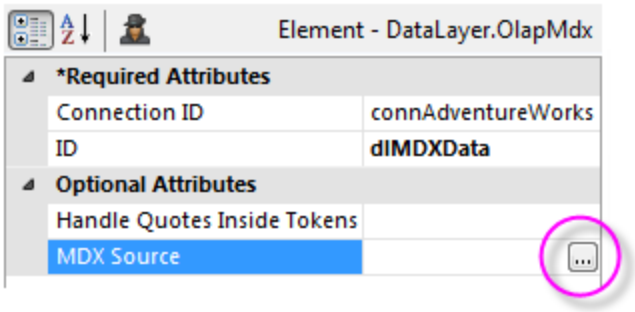
When connecting to an Oracle database, SQL Query Builder will retrieve a list of schemas, prompt you to select one, and then retrieve the related objects.

The MDX Query Builder

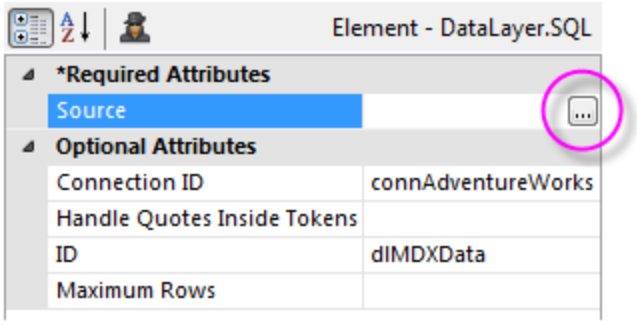
The **MDX Query Builder** provides an easy way to construct MDX queries, which are essentially two-dimensional queries against three-dimensional (cube) data. The notation used to select cube data has strict formatting requirements and the MDX Query Builder helps you build valid queries.

You can invoke the MDX Query Builder from two datalayer elements: **DataLayer.MDX** and **DataLayer.SQL**.

DataLayer.MDX is used to retrieve data for the OLAP Grid and OLAP Table elements, exclusively, and DataLayer.SQL is used to retrieve data for a variety of other elements, including standard Data Tables and charts. The datalayers must reference a valid **Connection.OLAP** element to access the server.

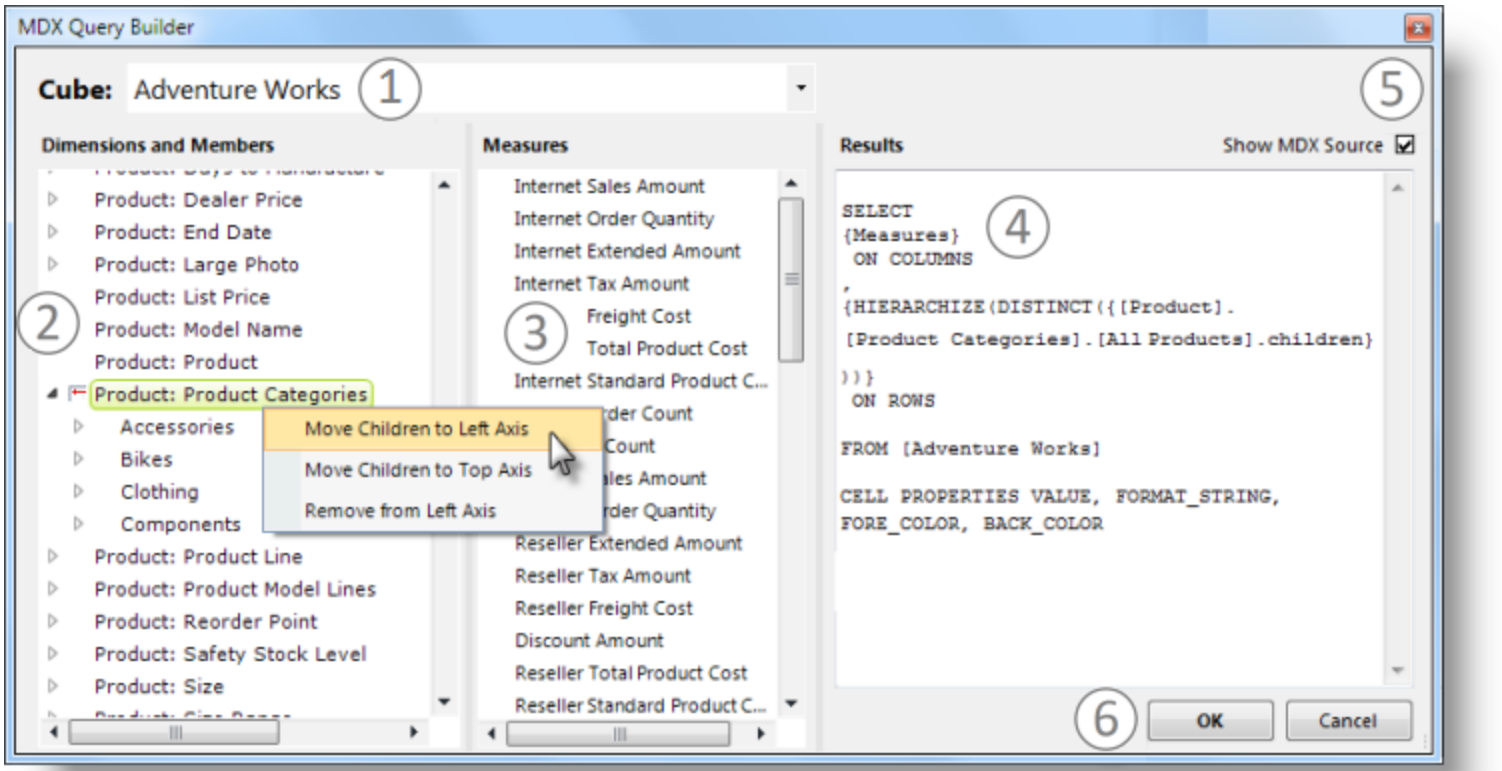


From DataLayer.MDX



From DataLayer.SQL

As shown above, the MDX Query Builder is invoked by clicking the browse button at the end of the datalayer's **MDX Source** or **Source** attribute.



When the MDX Query Builder is displayed, as shown above, you must first select a (1) Cube. Based on that selection, 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 or 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.

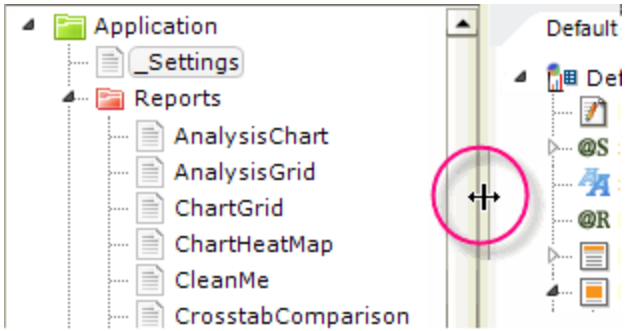
When a Query Has Been Entered Manually

If you *have already entered a query manually* in the MDX Source or Source attributes 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.

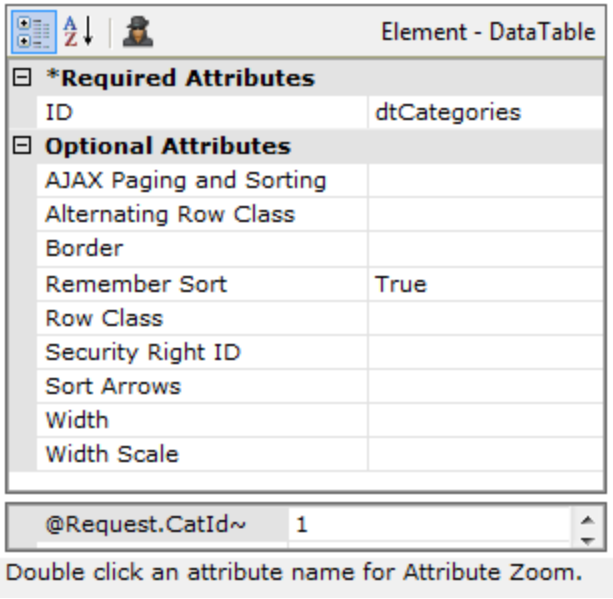
Working with Studio Panels

Studio provides a **intelligent** and **dynamic** development environment for the creation of Logi applications. Though almost all of the source files used in an application are text files and can therefore be edited with something as simple as Notepad, the wizards, editors, and other tools within Studio make the development process **faster** and **easier**.

The primary visual feature of Studio is its set of **panels**. These are used to manage, organize, and edit the files that make up a Logi application. The panels are flexible and can be sized to suit personal tastes.

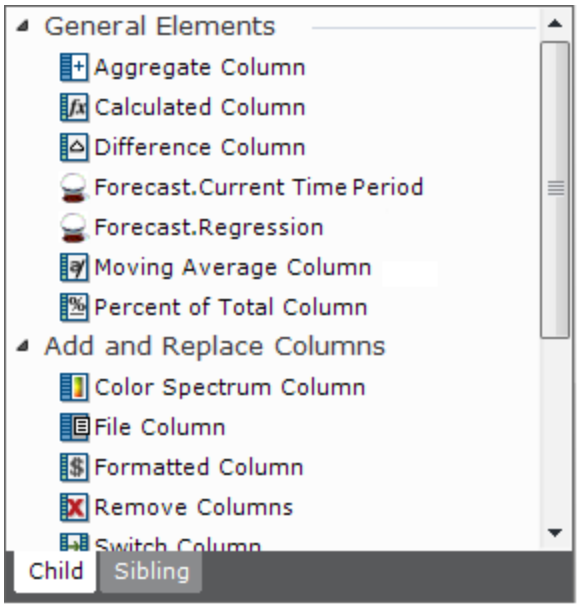


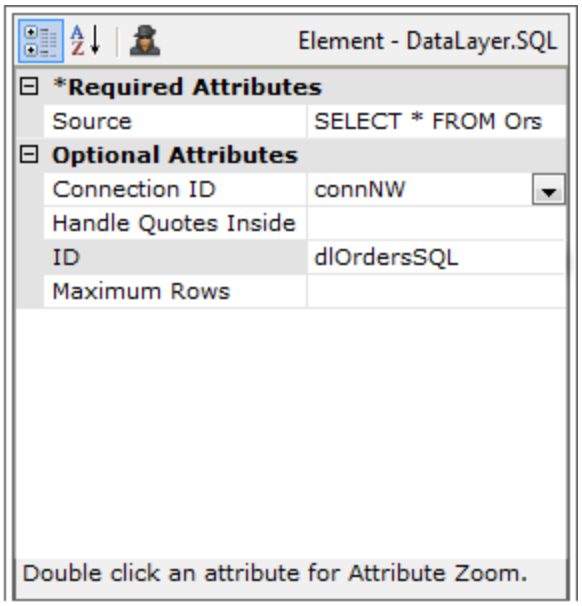
To **resize** a panel, simply position your cursor over the panel separator bar (where it will change to the resizing cursor, as shown above) and **drag** it in the desired direction. This applies to both vertical and horizontal separators.

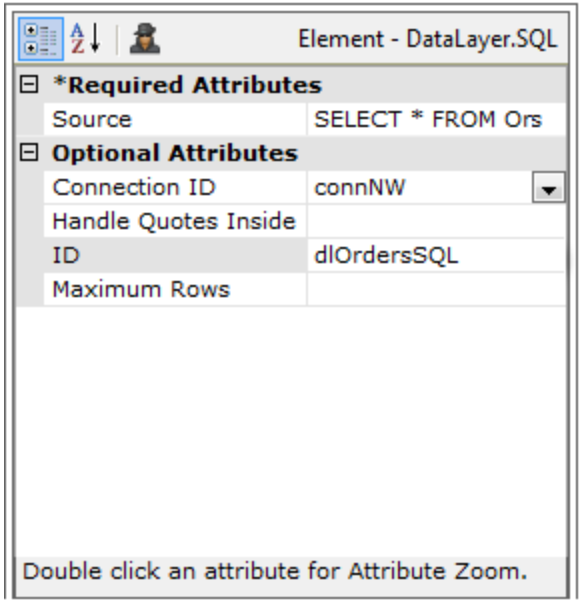


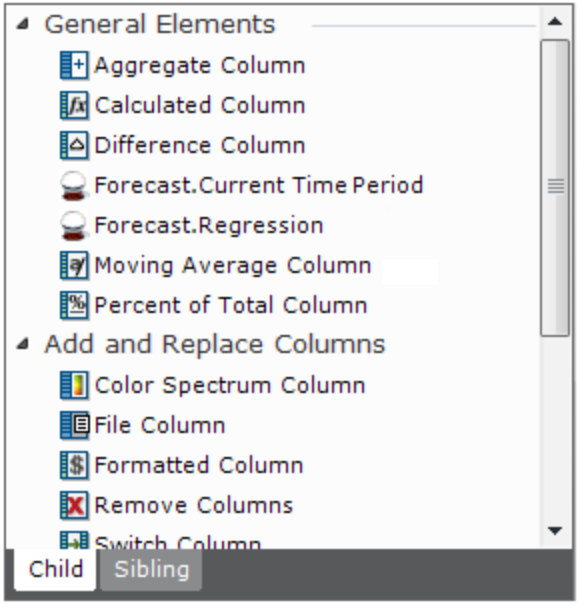
If you prefer to add elements to the Element Tree using the "right-click and context menu" method, you may not want to see the **Element Toolbox** panel at all; similarly, you may not use the **Test Parameters** panel. You can effectively hide these two by reducing their *heights*, as shown above. Just drag their separator bars downward completely.

The Test Parameters panel is displayed only if the application accepts parameters.





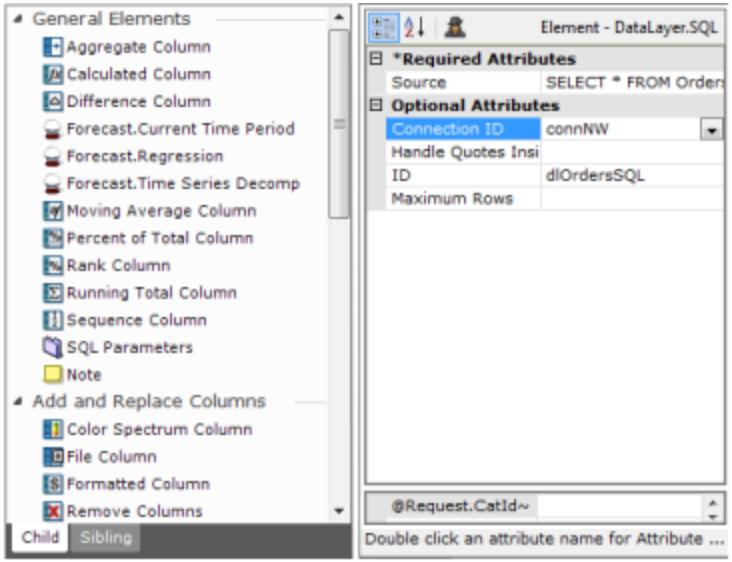




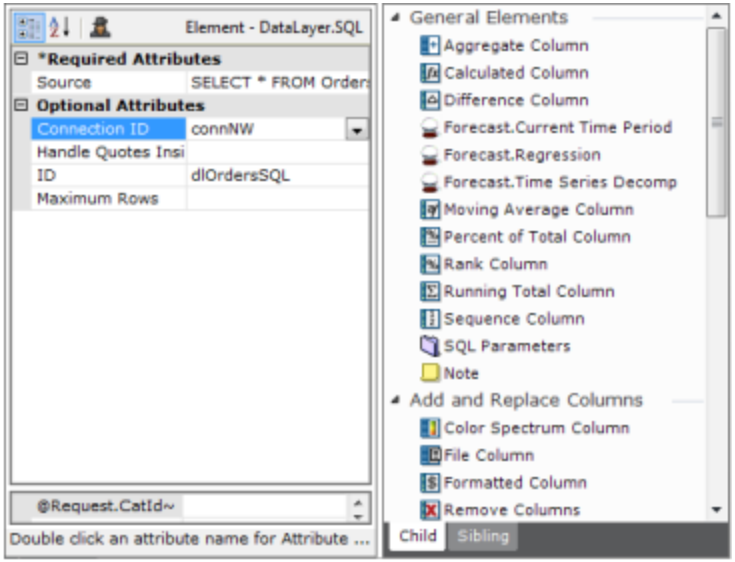
Elements then Attributes

Attributes then Elements

If desired, the **vertical** positions of the Element Toolbox and Attributes/Test Parameters panels combination can be swapped, as shown above, usingthe Tools → Elements and Attributesmain menu item.



Elements then Attributes



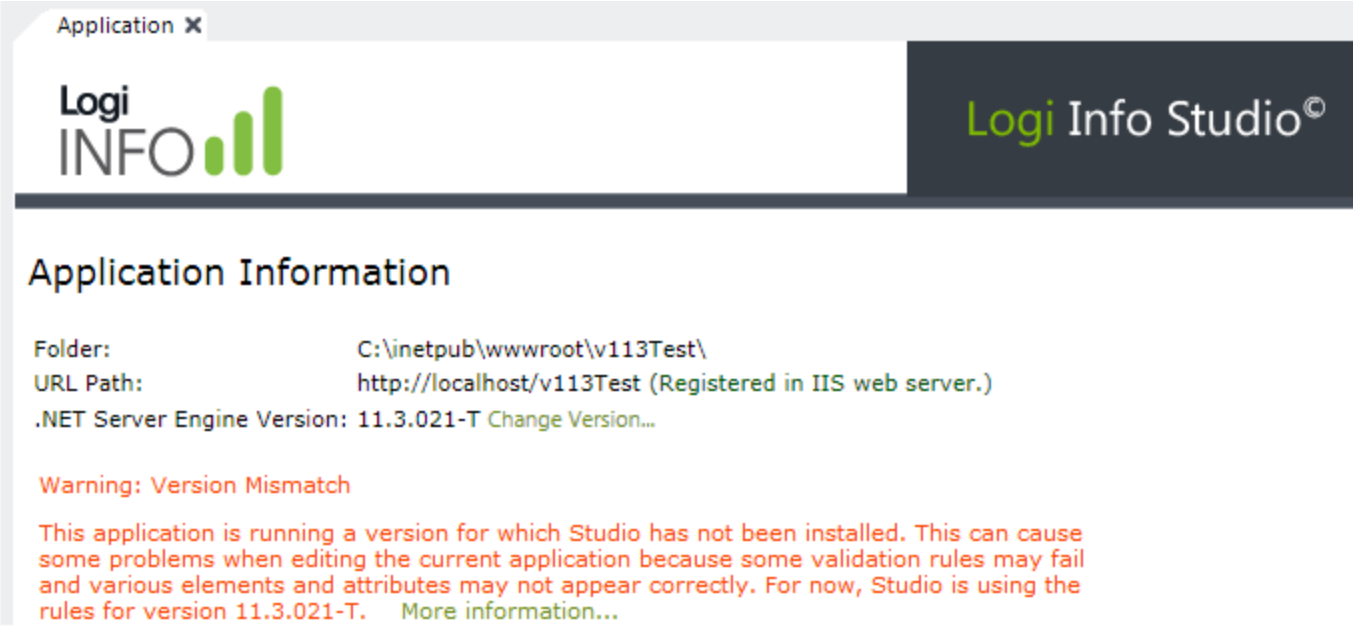
Attributes then Elements

The two panels can also be arranged side-by-side, as shown above.

Changing Application Versions

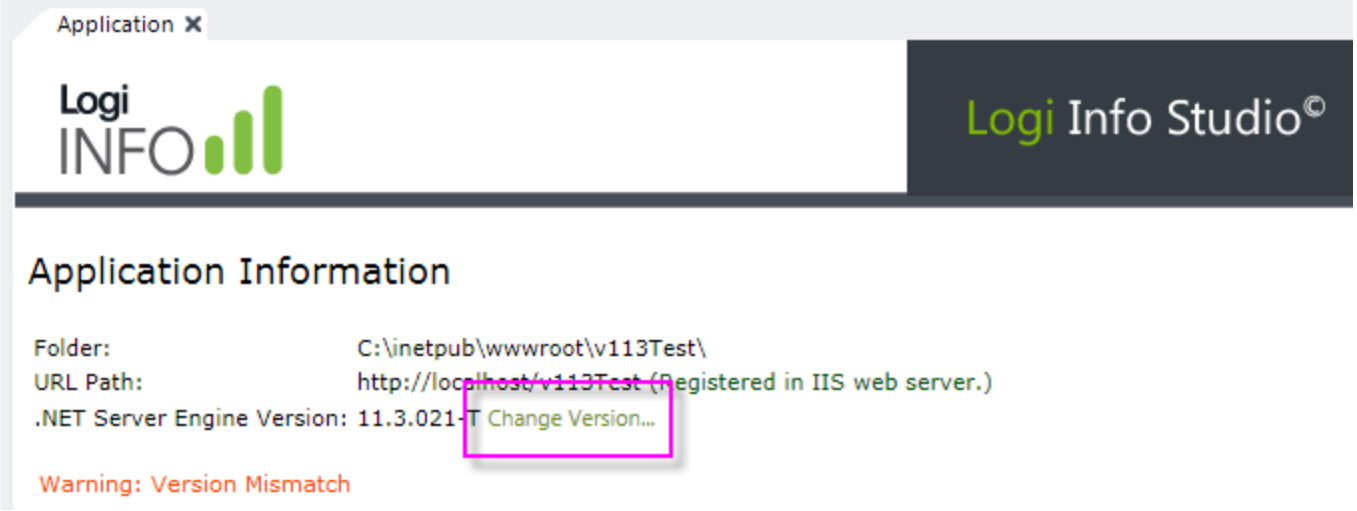
A Logi application includes files and settings installed on the web server and *version-specific* binary files included in each Logi application project folder. This allows each Logi application to maintain its own **version identity** and allows backward compatibility from new versions of Logi products.

Applications can be **upgraded** and **downgraded** to different versions and, when a new version of Logi Info is installed, applications must be upgraded before they'll have the benefits of the new version.



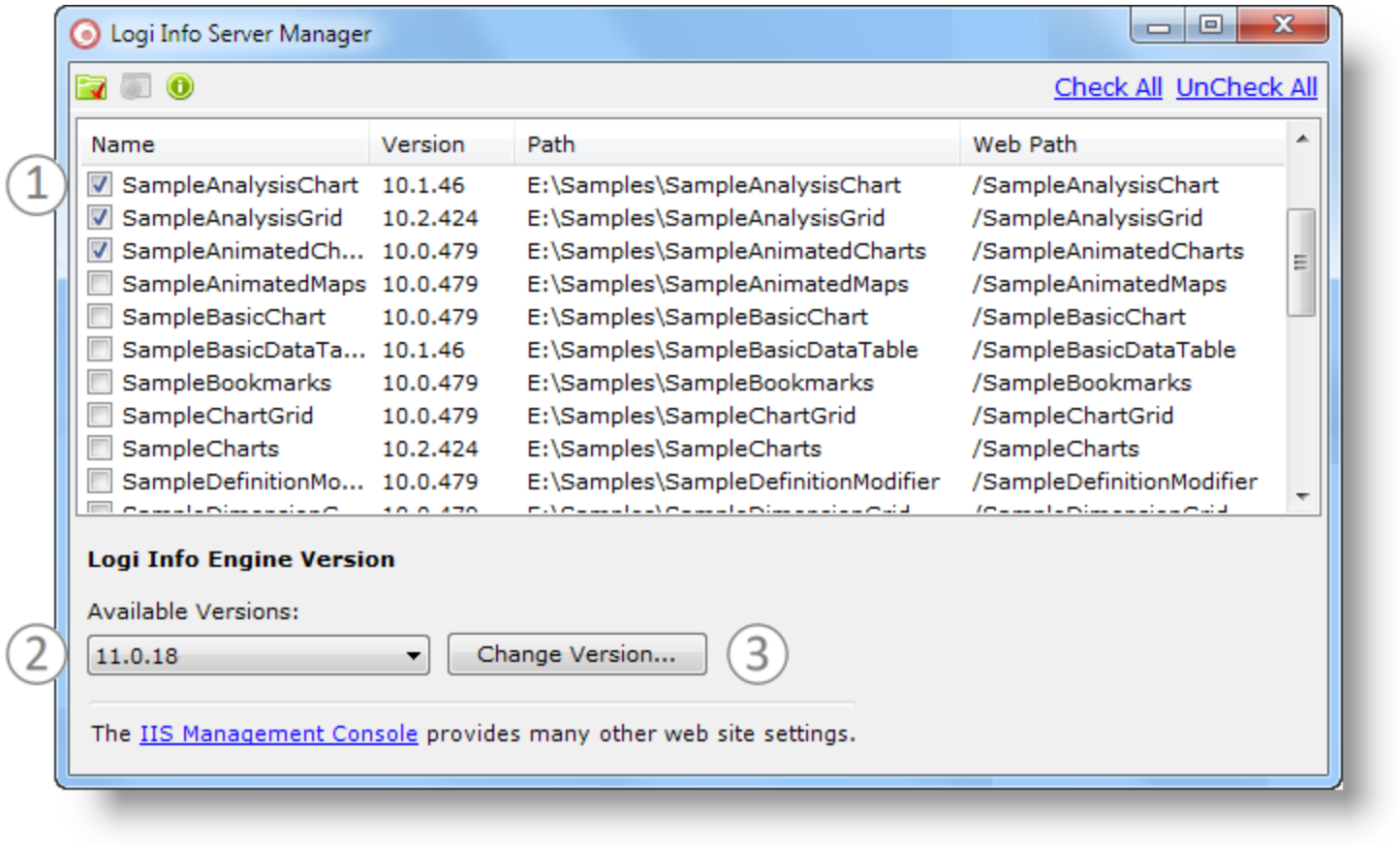
When you open an application, Studio displays a **warning** in its Application tab, as shown above, if the application's version does not match the version of Studio being used. Applications will continue to run correctly with this mismatch but, as the warning says, the rules governing element relationships and attributes may be incorrect when editing definitions.

For existing Logi .NET and Java applications, Studio provides a quick and easy way to change versions:



To change versions, click the **Change Version...** link, shown above. This will display a wizard with a list of the installed versions and, after you select one, will replace the application's version-specific binary files with the files of the selected version. The replacement may take several minutes. *This will not affect any definitions or support files you have created and you can revert the change*.

For .NET applications, the **Server Manager**, which can be launched from Studio's Tools menu, can also be used to change application versions. This is a tool that allows you to examine all of the .NET Logi applications on the local web server and manage them.



The **Server Manager** dialog box, shown above, displays a list of the applications installed on the local web server. Select one or more applications (1) and the desired product versions (2), and click Change Version... (3). The process usually takes less than a minute for each application selected.

Server Manager is intended for use *only* with .NET applications, using the IIS or Cassini web servers; if neither one of these servers is installed, Server Manager will prompt you to install them.

For additional important information about changing application versions, see *Logi Product Upgrades*.

Bookmark Storage

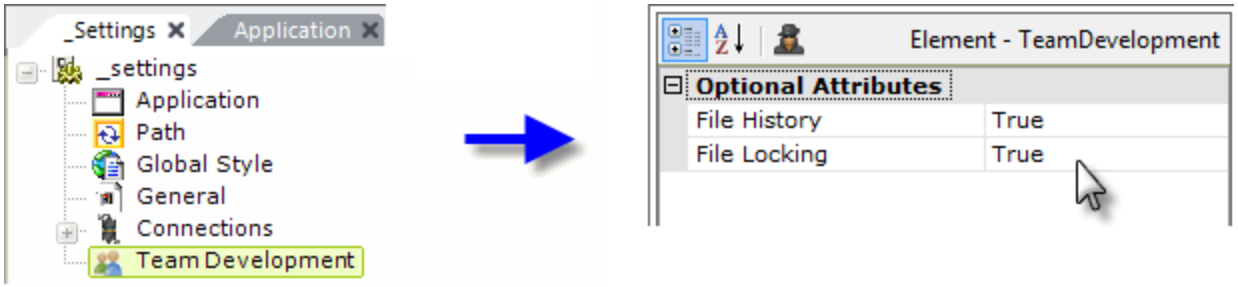
Bookmark, Gallery, and Save files are typically stored as XML files in the web server file system. However, this may not be useful or practical in some Logi application deployments, so it's possible to store them instead in a SQL database. Logi Studio includes a migration tool and a special **File To Database Mapping** element that make it easy to transfer existing bookmark files into a database, and then use them from there.

For more information about migrating these files, see "Storing Bookmark, Gallery, and Save Files in a Database" on page 328.

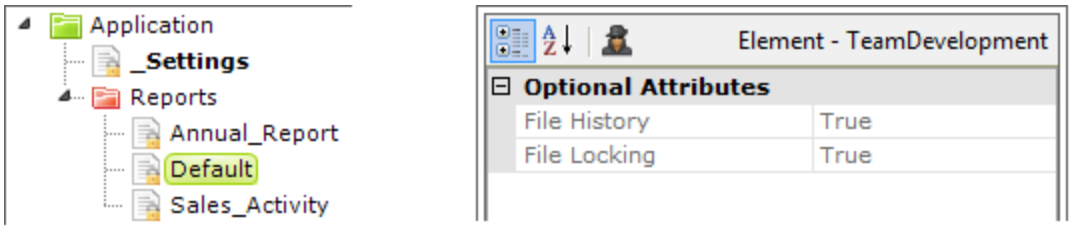
Team Development Features

Note: This feature is not supported in Info 12.7.

Logi Studio includes two features for use in a **team development** environment, where multiple developers are working on the same Logi application. **File Locking** prevents multiple developers from inadvertently working on the same file at the same time and blocking or overwriting each other's modifications. **File History** retains a copy of each file revision and current versions can be "rolled back" to earlier versions.



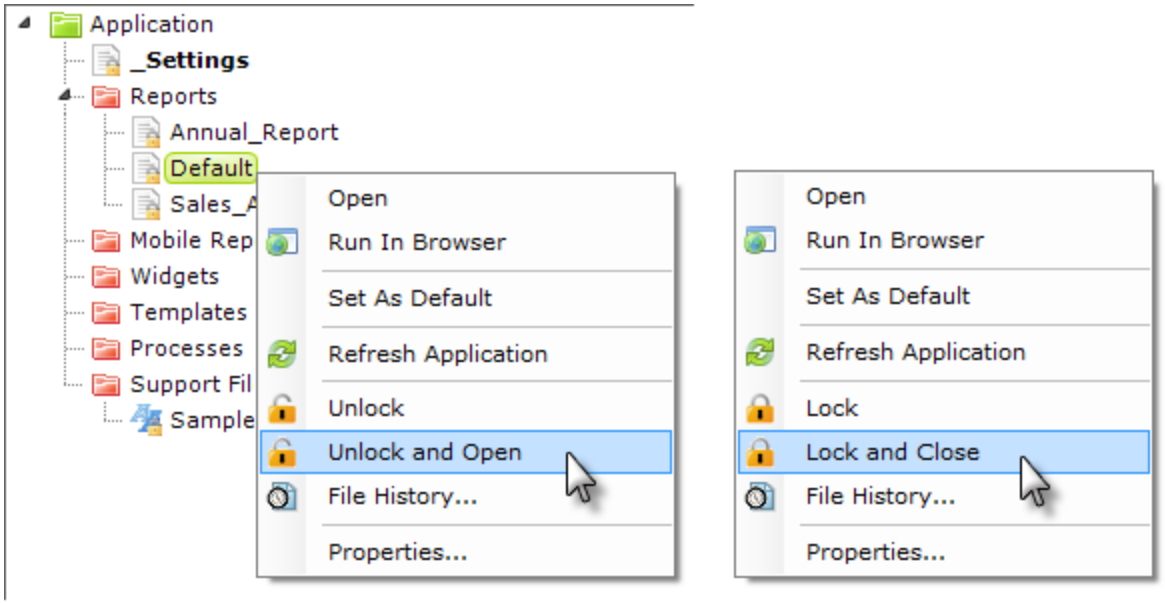
As shown above, these features are enabled by adding a **Team Development** element to the application's **_Settings** definition. The two features, File History and File Locking, are enabled by setting their respective attributes.



"Padlocked" icons

Editing disabled (gray text)

When you Save the _Settings definition, the Team Development features will become enabled. Definitions in the Application panel, including the _Settings definition, will be shown with a "padlocked" icon, indicating that the files have been locked. Hovering your mouse over the definitions will cause a "File Locked" tooltip to appear. Attributes and their values for elements in those definitions will be shown with gray text, indicating that they cannot be edited.



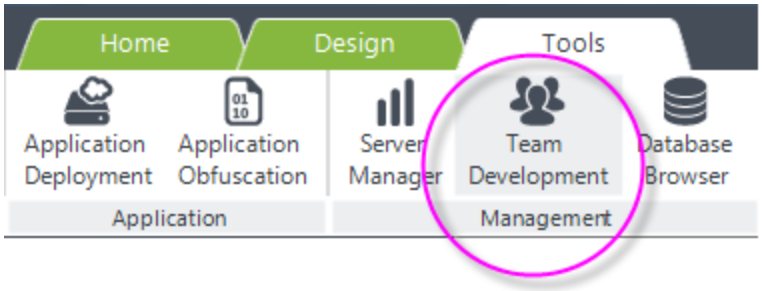
Individual files can be unlocked, opened, closed, or locked using the pop-up menu options, shown above, that are now available when a file is right-clicked. The file icons will change to reflect a file's locked or unlocked state. Locking applies to text-based **support files** as well as definition files.

Team Development uses a **source control system** approach, where to unlock a file is to "check it out" and to lock it is to "check it in". This means:

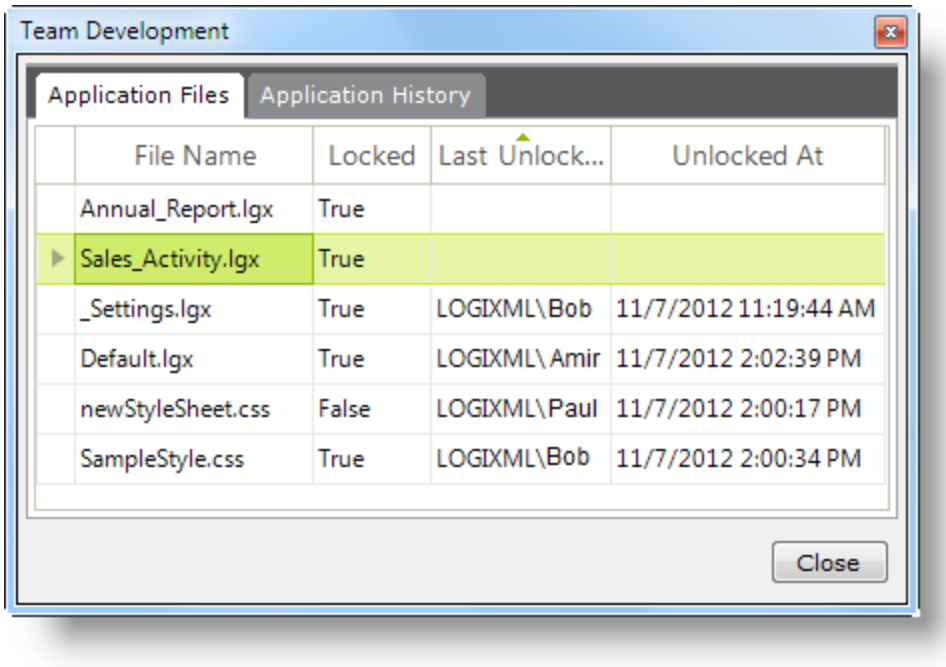
- When no one is working on any file, they are all locked (checked-in) and available to be unlocked (checked-out) by anyone.
- When you unlock (check-out) and open a file, you can edit it, but everyone else is prevented from working with it.
- When you save and/or re-lock (check-in) the file, then it's available to be unlocked by anyone again.

When a file has been unlocked by a user, other users attempting to unlock it will see an "in-use" icon next to the entry in Studio and a tooltip that indicates who's working with this file, when they hover their mouse over it.

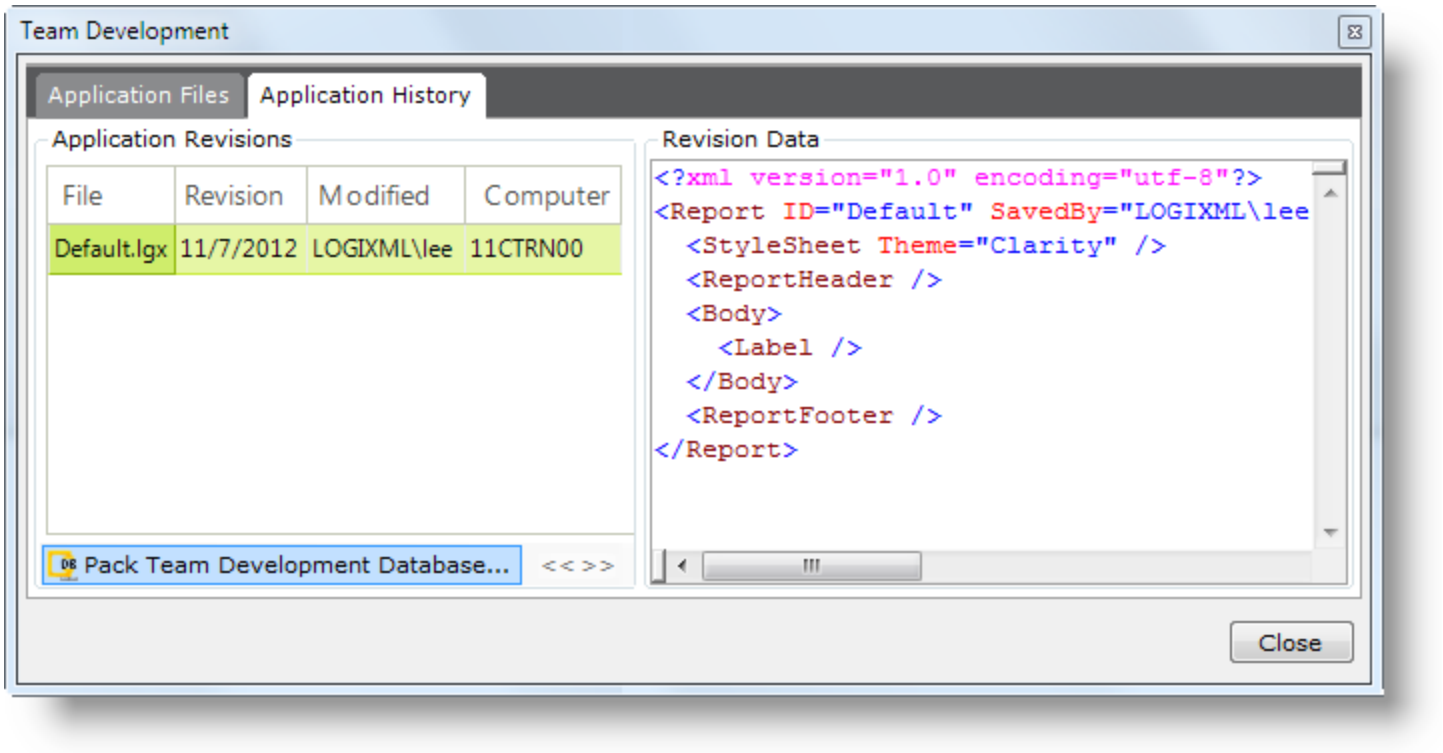
The initial locked state is *also* applied to the **_Settings** definition, and you must now unlock it if you want to modify it.



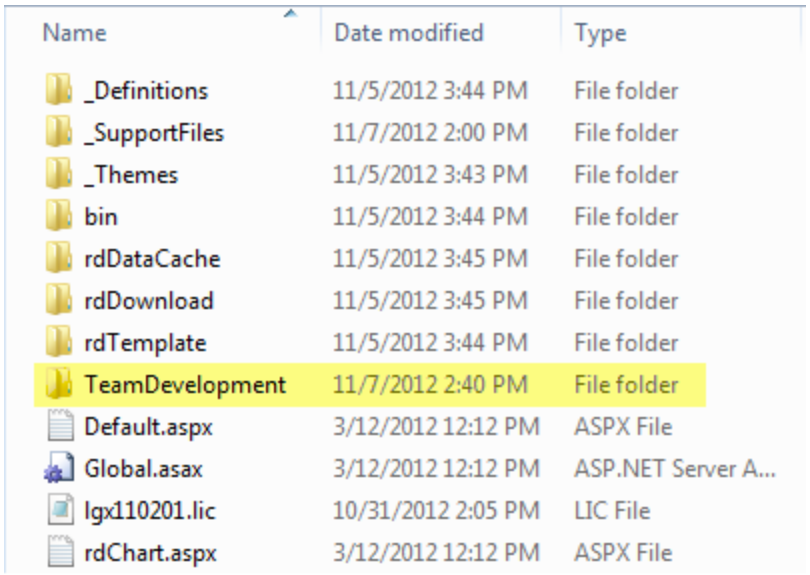
If they've been enabled in the _Settings definition, File Locking and File History information can be viewed by clicking the **Team Development** menu item, shown above, in the Tools tab.



When the toolbar icon is clicked, the **Team Development** dialog box opens, as shown above. **File Locking** details are displayed in the first tab, allowing development team members to determine which files are locked, who locked them, and when.



File History details are displayed in the second tab. Revision information and the actual revised data are presented. If a particular file is double-clicked, an individual file history dialog box is displayed and it includes a *Rollback...* button, which can be used to replace the current file contents with the selected version's contents.



The Team Development features use a **localdatabase** file located under the folder "TeamDevelopment", shown above, in the Logi application folder. The folder and database file are automatically created when the Team Development features are enabled

The **Pack Team Development Database** feature helps to maintain performance. Clicking the appropriate button in the Application History tab will present a prompt for confirmation. Clicking **Yes** will start a process that makes a backup of the Team Development database and then packs and shrinks it. All developers should close the application in Studio before a packing operation is started. You should pack the database whenever you feel that **File History** retrieval is taking a long time.

These features allow team developers to work together on an application, without getting in each other's way, and provide a useful audit trail of modifications.

System Configuration

A commonly-used approach for team development is to set up a centralized "development server" where the Logi application files can be accessed, using a shared network folder, by all members of the development team. Logi Info Server is installed on that server, along with a web server and all necessary supporting files. A Logi Info license is required for that server.

Developers who can "map" a local drive to the shared folder on the development server must be given *Read*, *Write*, *Modify*, and *Delete* file access permissions. Developers must also have Logi Info or, at the very least, Logi Studio installed and licensed on their own machines. They then "open" the Logi application on the development server and apply the Team Development settings as outlined above.

In the _Settings definition, the Path element's **Application Path** attribute should be set to the URL of the Logi app on the development server, rather than to the usual `http://localhost/yourApp`. Do *not* set the **Alternative Definition Folder** attribute, which is not relevant for this situation.

Working with Source Control

The choice of a source control system is highly dependent on the complexity of the source control functionality required. For smaller, simple applications, the use of the Team Development functionality within Logi Studio is often enough. Logi Team Development allows you to maintain version control, based on file saves, and provides file-locking/sharing capabilities for multiple developers.

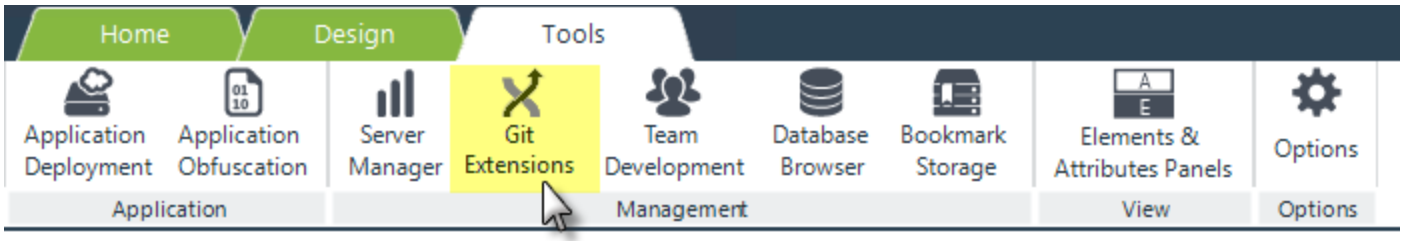
If you require more sophisticated source control, then a third-party product will be necessary. Popular products include Git, Microsoft TFS, and Subversion. The files that make up a Logi Info application are used directly from the file system; they're not stored in, or controlled by, a database or other constraining mechanisms, so third-party source control products can be used directly on them without concern.

To benefit from a third-party product, however, you should keep your Logi application's source code folders (`_Definitions`, `_SupportFiles`, `_Templates`, `_Scripts`, `_Plugins`), and all the root folder `.aspx` files under source control. Keeping the `bin`, `rdTemplate`, and other "rd*" system folders *outside* of source control will allow you to easily move between different versions of the Logi engine without interfering with your source control functionality.

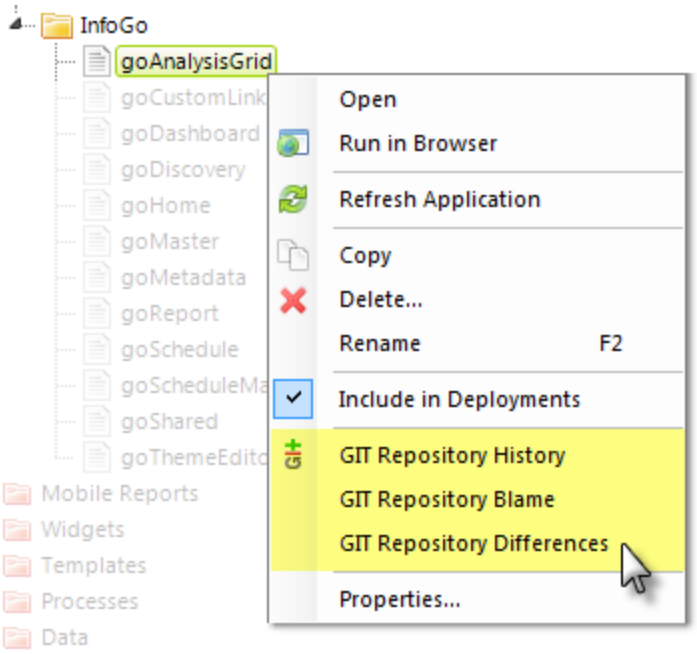
Basic Git Extensions Support

Git Extensions is a standalone UI tool for managing Git repositories. Logi Studio is now able to integrate Git Extensions, providing quick access to basic Git features when it's being used as the application's code repository.

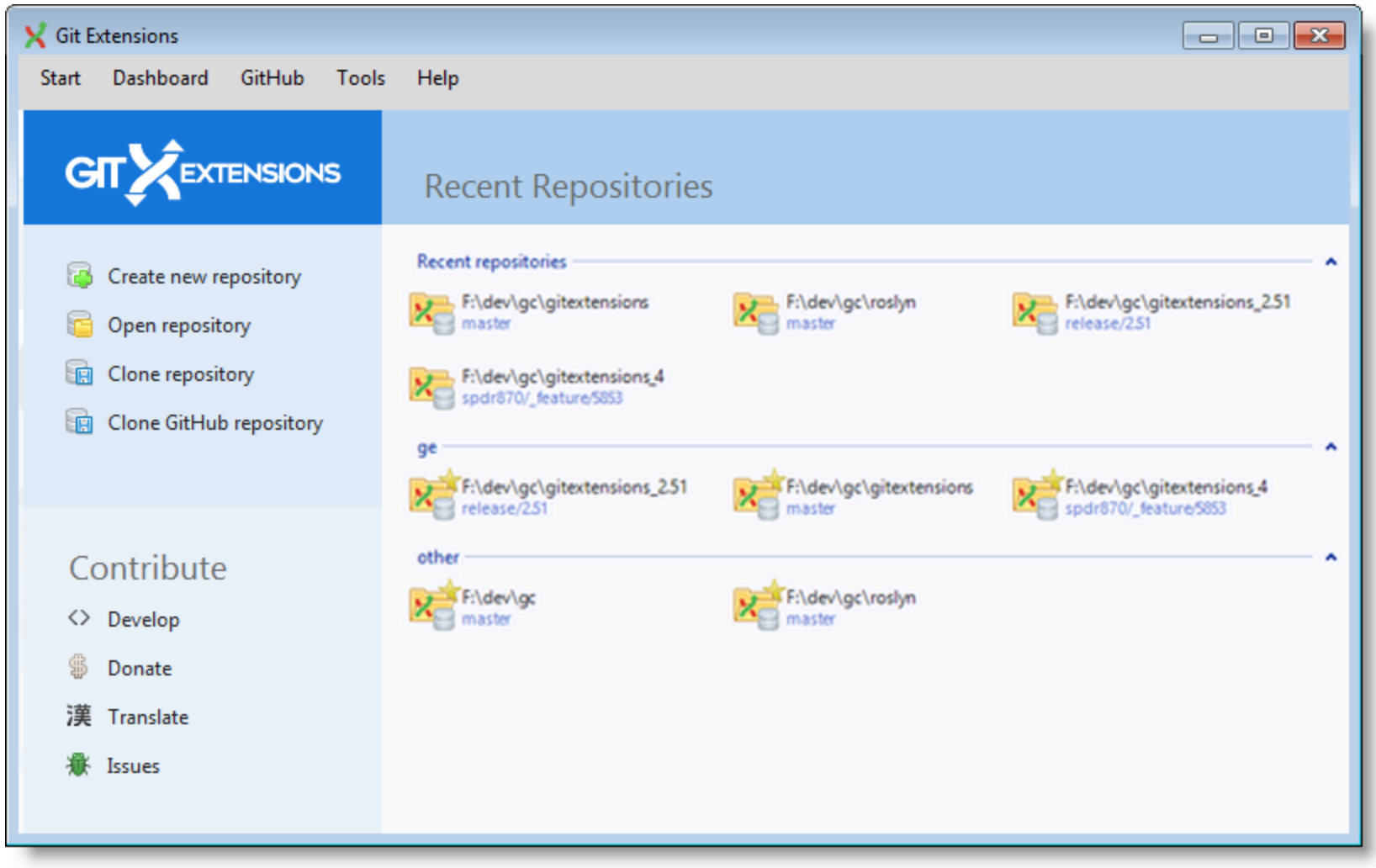
Logi Studio's Git Extensions integration is enabled when a Git Extensions installation is detected and when an application ancestor folder contains a .git file (indicating that the application files are actually being managed in Git).



Once enabled, Git features can be accessed using the icon on Studio's Tools ribbon menu, shown above,



or by right-clicking any file in the Application Panel, as shown above.



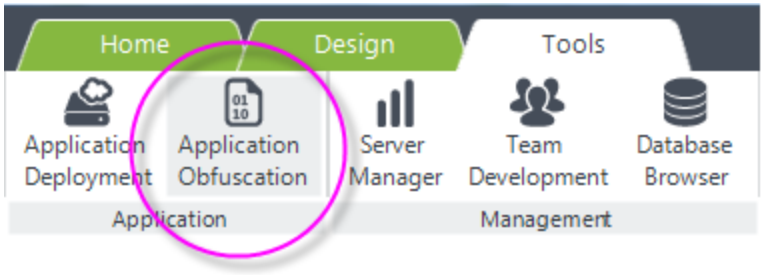
In either case, the Git Extensions UI shown above will appear so that you can manage your repositories and files.

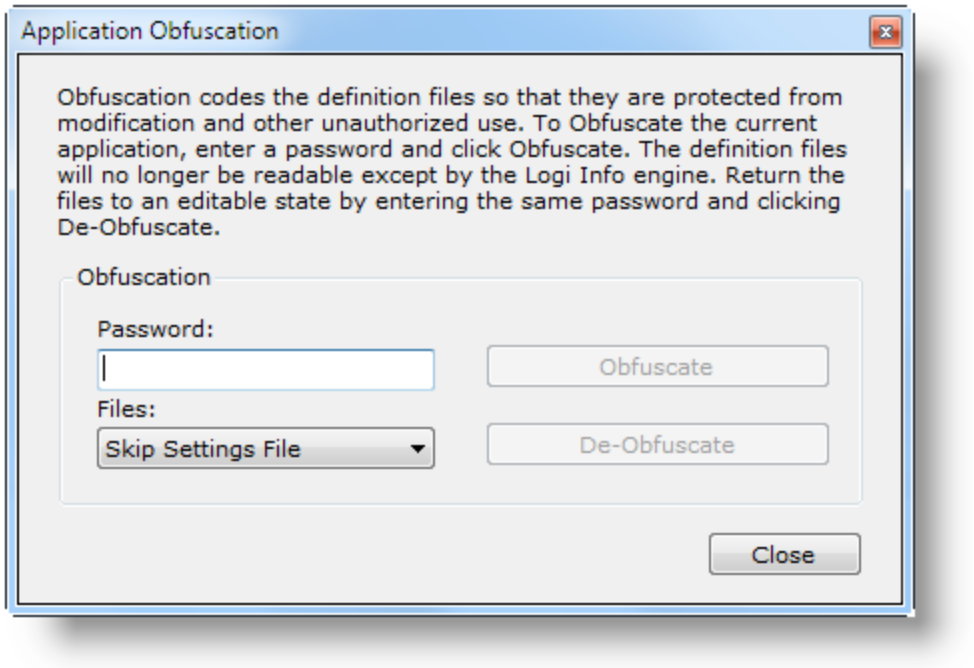
Obfuscating Definitions

Obfuscation is a common technique used by developers to **protect** their software products from theft, tampering, and reverse-engineering. The code obfuscation feature in Studio for .NET is a mechanism that allows Logi developers to make their _Settings, Report and Process definitions **unreadable** by humans. Definitions that have been obfuscated can be de-obfuscated, making them readable again. At runtime, both .NET and Java Logi applications automatically decode and use obfuscated files.

OEMs who embed Logi products within their own applications *should* use obfuscation to aid in securing their products before distribution, but there may be little need for obfuscation in an enterprise setting where all Logi report consumers are internal corporate employees.

The Obfuscation tool is available via the Studio **Tools** tab on the main menu:





Obfuscation is based on a *password*, which is entered in the tool, shown above. There are three file choices available in the process:

- **Skip Settings File** - All process and report definitions will be obfuscated but the _Settings.lgx file will not be included. This can be useful if certain settings, such as Connection Strings, need to be manually edited after deployment or if there is a

chance that debugging may be needed.

- **Settings File Only** - This is useful when settings need to be hidden, such as Connection String passwords, but there is little concern about securing other kinds of definitions. *In particular, OEMs with deployed solutions should use this option to shield their license key data from view.*
- **All Files** - Includes all .lgx files in the operation.

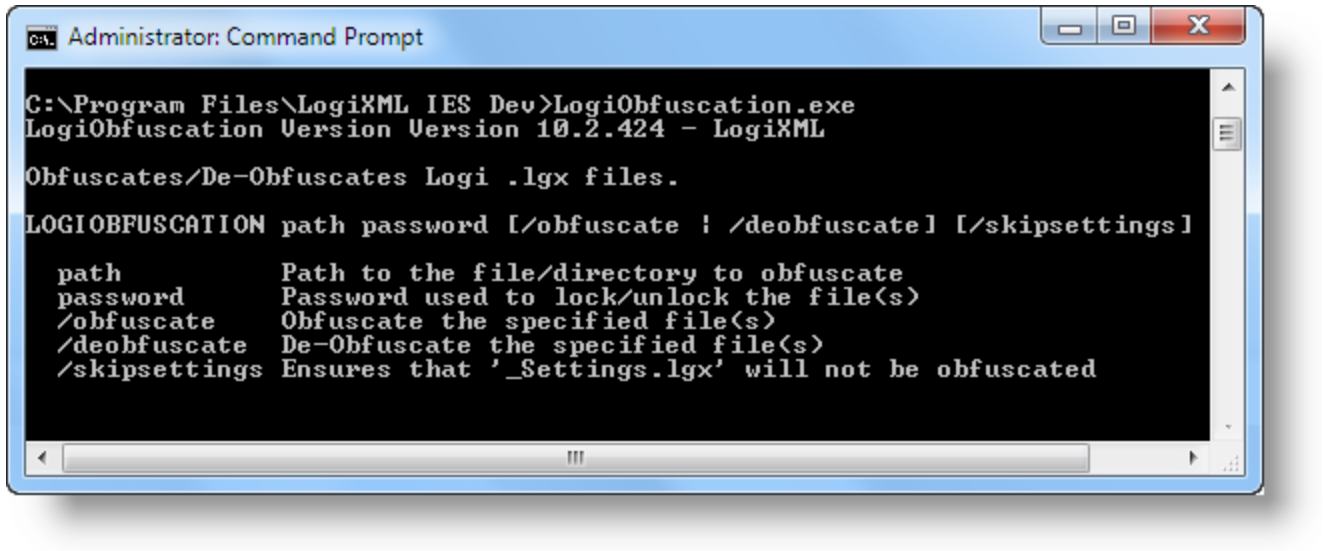
Other than the _Settings.lgx file, there is no way to select individual files within an application for obfuscation. In general, we do not recommend that you obfuscate files as a routine practice if you have no security concerns.

Application Obfuscation is compatible with Federal Information Processing Standards (FIPS) security in .NET and Java environments.

 When obfuscating files, record and store your password carefully. *Logi Analytics is not able to decipher passwords for the purpose of de-obfuscation if your password is lost or forgotten.*

Batch Obfuscation Tool

Developers may want to include obfuscation as part of an unattended build process and the Obfuscation Tool can be run as a console application for this purpose.



As shown above, if the default product installation is used, the tools can be found at:

```
C:\Program Files\LogiXML IES Dev\LogiObfuscation.exe
```

Running the tool without any arguments will display the help information shown above. These arguments are the same as the settings available through Studio's interface and are discussed in "Working with Source Control" on page 148.

Deleting a Logi Application

If you want to **delete** an entire Logi application, you can do so by using your file system tools, such as Windows Explorer, to delete the entire application folder.

First, in IIS Manager or other web server management tool, delete the virtual directory or web application entry for the app.

Then, delete the application folder. On a typical Windows server platform, for example, you would delete:

```
C:\inetpub\wwwroot\yourAppFolder
```

and all its subfolders and files.

Your web server may hold some application files open to improve performance and you may receive an "Access Denied" type error when you try the folder deletion. If this happens, try stopping and restarting the web server. For IIS, this can be done very easily by running `iisreset.exe` from the Start Menu → Run option or from a command line window. Once the web server is restarted, try deleting the application folder again.

Afterwards, in Studio, your application will still appear in the **Recent Applications** panel of the Welcome tab. Click its entry, as if to open it, and Studio will display a warning that it no longer exists and remove it from its list of recent applications.

Logi Studio Keyboard Shortcuts

The following details keyboard shortcuts for elements, definitions and general Logi Studio commands:

F1	Logi Help (DevNet)
F5	Open Application in browser
Ctrl + W	Close Window/Definition
Ctrl + C	Copy
Ctrl + V	Paste
Ctrl + S	Save
Ctrl + Shift + S	Save All
Ctrl + Z	Undo
Ctrl + Y	Redo
Ctrl + N	New Application

Shortcuts for Elements

F7	Push element up
F8	Push element down
Ctrl + -	Navigate backward
Ctrl + +	Navigate forward
Ctrl + R	Remark
Ctrl + F	Find & Replace Dialog (Defaults to find)
Ctrl + H	Find & Replace Dialog (Defaults to replace)

Shortcuts for Definitions

F2	Change definition name *
Ctrl + 0	Open Logi App by folder/directory
Ctrl + Tab	Switch to the next open definition

*You must have the definition highlighted in the definition browser

Logi JavaScript API for Chart Canvas Charts

The Logi JavaScript API for Chart Canvas Charts is a library that developers can use to work with Chart Canvas Charts.

The following topics provide information needed to work with the API:

About the API

The **Logi JavaScript API** allows you to interact with Chart Canvas Charts using JavaScript in order to customize them and/or refresh them periodically as needed. Data can be retrieved from the Info application server or directly from JSON sources and the visualization updated without redrawing the entire report page.

Scripts including the API functions can be executed by user interface actions, by using a timer, or by "listening" for data broadcast from an outside source. These are all discussed in the topics mentioned above.

Showing Initial Data

To show some initial data in a chart that will be updated automatically, use a datalayer under the chart or series as you normally would. This datalayer will run and provide data for the chart when the page is first loaded, or when its refreshed, but will not interfere with data delivered using this API.

Restrictions


Logi Info licenses the underlying code for Chart Canvas Charts from a 3rd-party. However, we only license the part of their code library that we need, not the entire library, and our license does not include the right to allow developers to independently access the library. Developers who access the library directly in their Logi Info applications using the Logi API do so at their own risk, as we cannot guarantee that the library and/or our implementation of it will not change in future Logi Info releases.

Getting the Chart Object

The API library is automatically included in your Logi Info v12+ application when you use Chart Canvas Charts; you do not need to take any special steps to include it in your definition. The first thing you need to do is get the chart object, using this function:

```
rdGetChartCanvasObject(chartId)
```

Here's an example that assumes you have a Chart Canvas Chart element with the ID "myChartCanvas1":

```
var myChartObject = rdGetChartCanvasObject('myChartCanvas1');
```

With the chart object in hand, you can proceed to manipulate the chart data.

Updating Chart Data

To request new data from the Info application server and replace the data, if any, in the chart, use this chart object method:

```
rdUpdateChartData([requestParams])
```

where `requestParams` is:

- An optional JavaScript object,
- Provided to the report definition as Request variables when the function is called,
- Where the property name is the Request variable name and the property value is Request variable value.

Here's an example using the chart object from "Getting the Chart Object" on the previous page:

```
var requestParams = {};
requestParams.OrderDate = "01/01/2015";
requestParams.ProductCategory = "Seafood";
myChartObject.rdUpdateChartData(requestParams);
```

In a hypothetical report definition using this script, the Chart Canvas Chart element would have a child datalayer that made use of the request variables by using tokens. For example, using DataLayer.SQL, they might be used in a query like this:

```
SELECT * FROM Orders WHERE OrderDate = '@Request.OrderDate~' AND Category = '@Request.ProductCategory~'
```

The query would return a result set in which the OrderDate column value = "01/01/2015" and the Category column value = "Seafood". The chart series would be updated with that result set. The same behavior applies whether the datalayer is a child of the Chart Canvas Chart element or a child of one of the Series elements.

Appending Chart Data

To request new data from the Info application server and append it to existing data, if any, in the chart, use this chart object method:

```
rdAppendChartData([requestParams], maxVisiblePoints)
```

where `requestParams` is:

- An optional JavaScript object,
- Provided to the report definition as Request variables when the function is called,
- Where the property name is the Request variable name and the property value is Request variable value.

and `maxVisiblePoints` is:

- A positive integer,
- The threshold of total chart data points. The chart will delete old data points if the number of total points is greater than this value.

Here's an example using the chart object from *"Updating Chart Data" on the previous page*:

```
var requestParams = {};
requestParams.OrderDate = "01/01/2015";
requestParams.ProductCategory = "Seafood";
myChartObject.rdUpdateChartData(requestParams, 100);
```

See "Updating Chart Data" on the previous page for an explanation of how the `requestParams` can be used as Request tokens in retrieving data.

Setting JSON Chart Data

To get the chart data directly from a JSON data source, without going to the application server, use this chart object method:

```
rdSetChartData(chartOptions, [updateType], [maxVisiblePoints])
```

where `chartOptions` is:

- A required JSON data object,
- Containing chart options, using the standard Logi Info server chart data structure.

and `updateType` is:

- An optional value,
- Either *'UpdateData'* (the default, which replaces the chart data) or *'AppendData'*, which appends the data.

and `maxVisiblePoints` is:

- A positive integer,
- Used only when `updateType` = *'AppendData',*
- The threshold of total chart data points. The chart will delete old data points if the number of total points is greater than this value.

Here's an example using the chart object from "Appending Chart Data" on the previous page:

```
var chartOptions = {
 chart: {},
 series: [{
```

```
id: 'mySeriesId',
 data: [29.9, 71.5, 106.4, 129.2, 144.0, 176.0, 135.6, 148.5, 216.4, 194.1, 95.6, 54.4]
}]
};
myChartObject.rdSetChartData(chartOptions, 'UpdateData');
```

Using a Timer

If you want to update your chart at regular intervals, you can using the functions previously discussed with the JavaScript `window.setInterval` function:

```
var myPieChart = rdGetChartCanvasObject('pieChart');
window.setInterval(function () {
 myPieChart.rdUpdateChartData();
}, 3000); The numeric value (3000 in our example) is the interval, which is expressed in milliseconds. A script like this would be
```

included using an **Include Script** element.

Using "Pushed" Data

You can also refresh charts using data that's "pushed" to the browser. This type of update is ideal for a use case in which many users receive very frequent data updates while avoiding web server performance degradation and negative network bandwidth impacts.

This technique relies on a 3rd party service, like Pusher.com, to transmit the data to the application. These services essentially broadcast data received from a (Logi or non-Logi) data source application to your Logi application in the browser, using a protocol like Web Sockets. You Logi application in the browser listens for these broadcasts and uses the Real-Time Chart API to update your charts.

Here's an example:

An external PHP application (not shown) generates data periodically and sends it to Pusher.com.

The Pusher.com API library is included in our Logi definition using an **Include Script File** element, with its Script File attribute set to `//js.pusher.com/2.2/pusher.min.js`.

The following JavaScript code is included in the definition using an **Include Script** element:

```
// init the pusher API channel and bind the updates.
var pusher = new Pusher('yourPusherID');
var channel = pusher.subscribe('live_data');

channel.bind('new_data', function(data) {
// when a message has been sent to the pusher API, it will enact this code block
var dataObj = data.feed;
```

```
// target and capture the line chart object
var chartCanvasObject = rdGetChartCanvasObject('RealtimeDataChart');

for($i = 0; $i < dataObj.length; $i++){
// load the data records to the data array
var myData = [];
myData.push({
x: new Date(dataObj[$i].datetime),
y: parseInt(dataObj[$i].value)
});

// create the new chartOptions object
var chartOptions = {
chart:{},
series: [{
id: 'RealtimeLine',
data: myData
}]
};

// update the line chart
if(chartCanvasObject != null){
// append the new data to the existing chart series
chartCanvasObject.rdSetChartData(chartOptions, 'AppendData', 100);
```

```
}
}
});
```

Whenever Pusher.com broadcasts new data, our Logi application will receive it and update the chart, without making a request to the Logi application web server or refreshing the web page.

Events: beforeCreateChart and afterCreateChart

These two events can be used to customize Chart Canvas Chart options and properties. As you can guess, one event fires just before the chart is created and one just after it's created. The events include these parameters:

Parameter	Description
id	The Chart Canvas Chart element ID.
options	The chart *options* object: the options structure for the chart.
chartCanvas	The Chart Canvas Chart container object.
chart	The *chart* object.

Here's an example that uses the *beforeCreateChart* event to cause the text "n/a" to appear on two charts when there are null data values. First, here's the Logi element source code (the JavaScript code is follows separately):

```
<Division ID="divNewAPI_Events" >
<ChartCanvas ID="verticalBarChart" >
<DataLayer Type="Static" ID="StaticDataLayer1" >
<StaticDataRow x="1" y="1" />
<StaticDataRow x="2" y="2" />
<StaticDataRow x="3" y="" />
<StaticDataRow x="4" y="4" />
```

```
<StaticDataRow x="5" y="" />
<StaticDataRow x="6" y="6" />
<StaticDataRow x="7" y="7" />
<StaticDataRow x="8" y="8" />
<StaticDataRow x="9" y="9" />
<StaticDataRow x="10" y="10" />
</DataLayer>
<Series Type="Bar" ChartYDataColumn="y" ChartXDataColumn="x" >
</Series>
</ChartCanvas>
<LineBreak />
<ChartCanvas ID="horizontalBarChart" ChartOrientation="SwapAxes" >
<DataLayer Type="Static" ID="StaticDataLayer2" >
<StaticDataRow x="1" y="1" />
<StaticDataRow x="2" y="2" />
<StaticDataRow x="3" y="" />
<StaticDataRow x="4" y="4" />
<StaticDataRow x="5" y="" />
<StaticDataRow x="6" y="6" />
<StaticDataRow x="7" y="7" />
<StaticDataRow x="8" y="8" />
<StaticDataRow x="9" y="9" />
<StaticDataRow x="10" y="10" />
</DataLayer>
<Series Type="Bar" ChartYDataColumn="y" ChartXDataColumn="x" >
```

```
</Series>
</ChartCanvas>
<IncludeScript IncludedScript="(see the JavaScript code that follows)" />
</Division>
```

And here's JavaScript code:

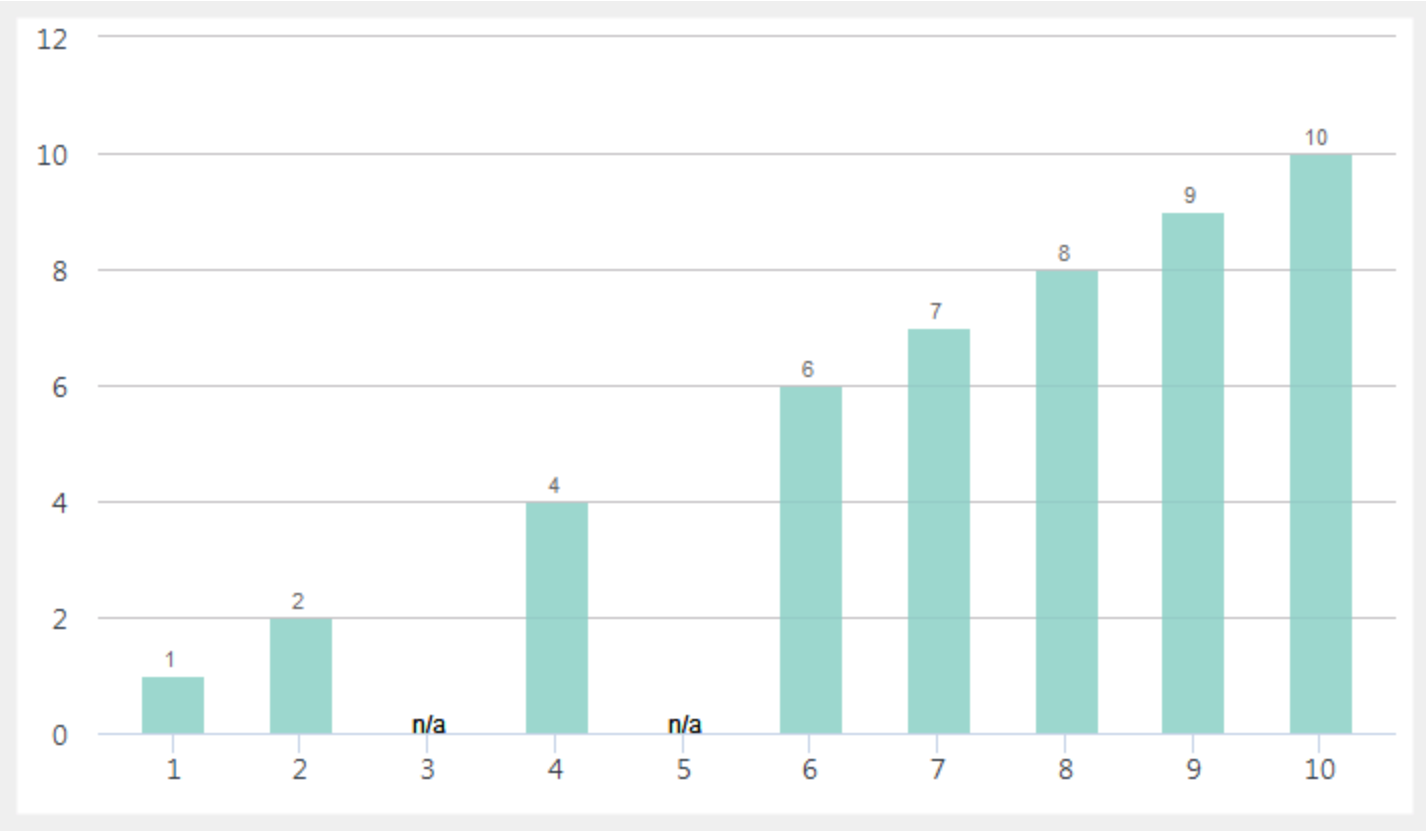
```
var verticalBarChartContainer = Y.one('#verticalBarChart');

verticalBarChartContainer.on('beforeCreateChart', function(e) {

setNAFormatter(e);

});

var horizontalBarChartContainer = Y.one('#horizontalBarChart');

horizontalBarChartContainer.on('beforeCreateChart', function(e) {

setNAFormatter(e);

});

function setNAFormatter (e) {

Highcharts.Point.prototype.dataLabelOnNull = true;
```

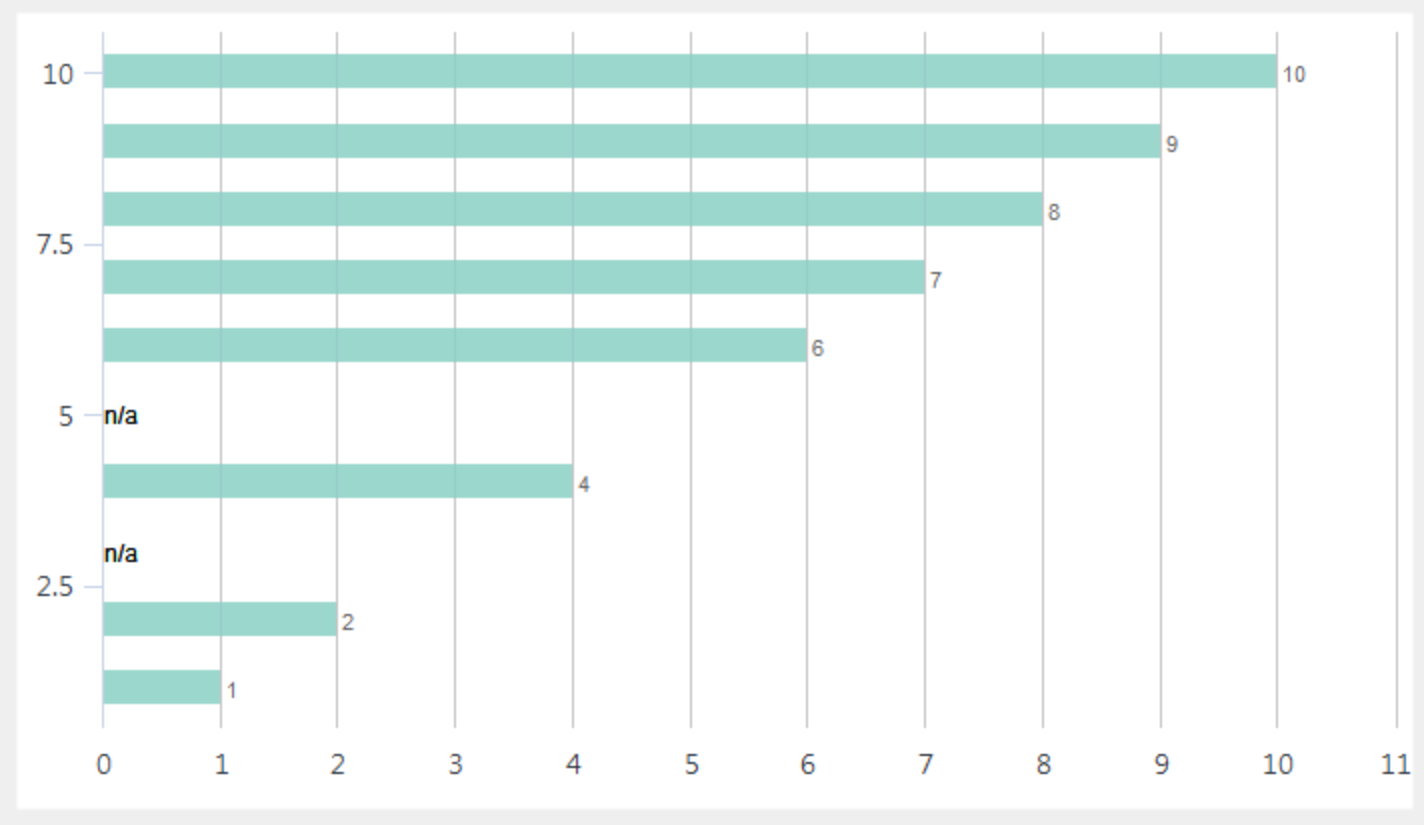
```
if (e.options.series == null || e.options.series.length == 0) {

return;

}

if (!e.options.plotOptions) {

e.options.plotOptions = {};

}

if (!e.options.plotOptions.column) {

e.options.plotOptions.column = {};

}

e.options.plotOptions.column.dataLabels = {

enabled: true,

formatter: function () {

var str;

if (this.y !== null) {
```

```
return this.y;

} else {

if (e.options.chart.inverted) {

var chart = this.series.chart,

offset = this.point.barX + 3;

chart.renderer.text('n/a', chart.plotLeft, chart.plotTop + chart.plotHeight - offset).add();

} else {

var chart = this.series.chart,

offset = this.point.barX + this.point.pointWidth / 2 - 8;

chart.renderer.text('n/a', chart.plotLeft + offset, chart.plotTop + chart.plotHeight).add();

}

}

return false;

},
```

```
crop: false,

overflow: 'none'

}

}
```

And the resulting charts look like this:





You can see, as shown above, that where there are no data values, the text "n/a" has been inserted.

Classic Chart Label Formatting

Logi Classic Chart **axis labels** can be formatted using several special-purpose elements and, in addition to these, the actual label text of *non-animated* charts can be formatted using a language called **CDML**.

The following topics discuss CDML and its use:

- CDML Reference
- CDML Examples

A number of Classic Charts have been deprecated; they are still supported and will work, but their elements are no longer available in Studio. See Classic Charts & Gauges for more information and use Chart Canvas Charts for all new development.

About CDML

The static Classic Charts in Logi Info are created using components from the *ChartDirector* charting library. The ChartDirector Markup Language (CDML) can be used with the charts in order to format chart labels by marking them up with HTML-like tags. CDML allows a single text string in a Logi attribute value to be rendered using multiple fonts, on multiple lines, and with different colors. For example, CDML can be used to add subscripted or superscripted characters to axis labels, such as "CO_2" and "Km^2", or to add a newline within a label, so it appears on two lines. Data used as labels can also be formatted. CDML provides powerful formatting features that can enhance your charts.

 Remember, CDML *does not apply* to Logi animated Classic Chart, Map, or Gauge elements, nor to Chart Canvas charts.

CDML Reference

In general, all **tags** in CDML are enclosed by `<*` and `*>`. Attributes within these tags then determine the styles of the text following the tags, within the same block. If you want to include `<*` in text without being interpreted as CDML tags, use `<<*` as an escape sequence.

Font Styles

You can change the *style* of the label text by using CDML tags. For example, the line:

`<*font=timesi.ttf,size=16,color=FF0000*>Hello <*font=arial.ttf,size=12,color=8000*>world!` in an chart's **Label Title** attribute will result in the following text being rendered:

The following table describes the supported font style attributes in CDML:

CDML Attribute	Description
font	Specifies the start of a new style section, and sets the font name. You may use this attribute without a value (that is, use "font" instead of "font=arial.ttf") to create a new style section without modifying the font name. Use `<*/font*>` to terminate a style section, which will restore the original font styles.
size	Specifies the font size.

CDML Attribute	Description
width	Specifies the font width. This attribute is used to set the font width to different values. If the width and height are the same, use the *size* attribute.
height	Specifies the font height. This attribute is used to set the font height to different values. If the width and height are the same, use the *size* attribute.
color	Specifies the font color, in hex format, e.g. *000000*.
bgColor	Specifies the font background color, in hex format, e.g. *000000*.
underline	Specifies the line width of the line used to underline the following characters. Set to *0* to disable underline.
sub	Specifies the following text as subscript.
super	Specifies the following text as superscript.
xoffset	Specifies a horizontal offset, in pixels, for the following text to be drawn from the original position.
yoffset	Specifies a vertical offset, in pixels, for the following text to be drawn from the original position.

 Unlike HTML tags, no double or single quotes are used within CDML tags. This is because CDML tags are often embedded as **string literals** in the parent page code and quotes, if used, may conflict with other string literal quotes. Therefore, quotes *must not* be used within CDML tags. Also, unlike HTML tags, CDML tags use the comma character (,) as a delimiter between multiple tag attributes. This is because certain attributes, such as a font file name, may contain embedded spaces.

Blocks and Lines

In CDML, a text string may contain multiple **blocks**. A block may contain multiple lines of text by separating them with the `<*br*>` tag. For example, the line:

`<*size=15*><*block*><*color=FF*>BLOCK<*br*>ONE<*/*> and <*block*><*color=FF00*>BLOCK<*br*>TWO<*/*>` in a chart's **Label Title** attribute will result in the following text being rendered:

The example above uses a line of text that contains two blocks separated by the word *and*. Each block in turn contains two lines. The blocks are defined using `<*block*>` as the starting tag and `<*/*>` as the ending tag.

When a block ends, font styles will be restored to the state they were in before entering the block.

Block Attributes

CDML supports **nested blocks**; that is, a block can contain other sub-blocks. Attributes are supported within the `<*block*>` tag to control the alignment and orientation of the sub-blocks. The following table describes the supported attributes inside a `<*block*>` tag:

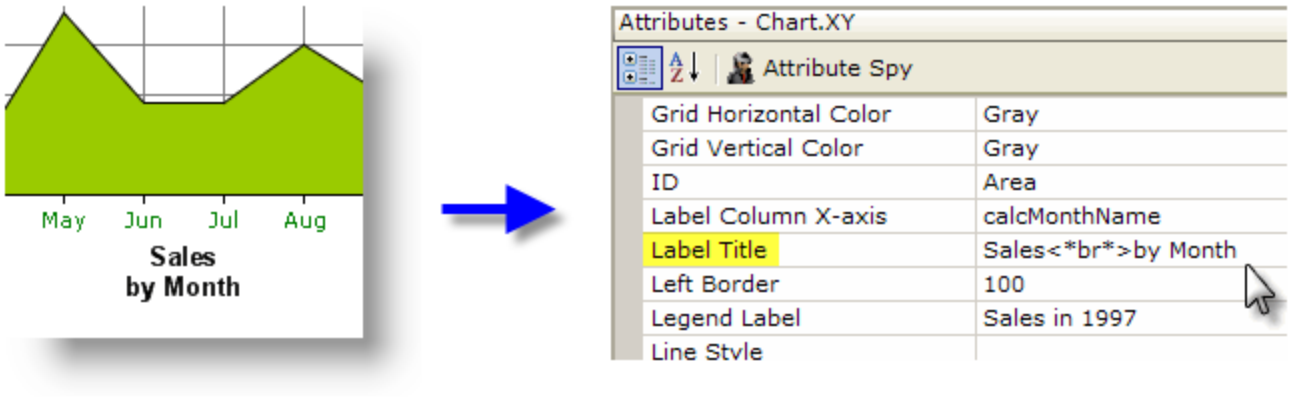
Block Attribute	Description
Block Attribute	**Description**
angle	Specifies the angle of rotation for the block, in degrees of counter-clockwise rotation.
bgColor	Specifies the block background color, in hex format, e.g. *000000*.
halign	Specifies the horizontal alignment of lines. This is for blocks that contain multiple lines. Supported values are *left*, *center* and *right*. The default value *left* means the left border of each line should align with the left border of the block. The value *center* means the horizontal center of each line should align with the horizontal center of the block. The value *right* means the right border of each line should align with the right border of the block.
height	Specifies the height of the block, in pixels. By default, the height is automatically determined to be the height necessary for the contents of the block. If the height attribute is specified, it will be used as the height of the block.
linespacing	Specifies the spacing between lines as a multiple of the default line spacing. For example, a line spacing of *2* means the line spacing is two times the default line spacing. The default line spacing is the line spacing as specified by the font used.
maxwidth	Specifies the maximum width of the block in pixels. If the content is wider than maximum width, it will be wrapped into multiple lines.
truncate	Specifies the maximum number of lines of the block. If the content requires more than the maximum number of lines, it will be truncated. In particular, if truncate is *1*, the content will be truncated if it exceeds the maximum width (as specified by maxwidth or width) without wrapping. The last few characters at the truncation

Block Attribute	Description
	point will be replaced with "...".
valign	Specifies the vertical alignment of sub-blocks. This is for blocks that contain sub-blocks. Supported values are *baseline*, *top*, *middle, absmiddle,* and *bottom*The value *baseline* means the baseline of sub-blocks should align with the baseline of the block. The baseline is the underline position of text. This is normal method of aligning text, and is the default in CDML. For blocks that are rotated, the baseline is the same as the bottom. The value *top* means the top line of sub-blocks should align with the top line of the block. The value *bottom* means the bottom line of sub-blocks should align with the bottom line of the block. The value *middle* means the middle line of sub-blocks should align with the middle line of the block. The middle line is the middle position between the top line and the baseline. The value *absmiddle* means the absolute middle line of sub-blocks should align with the absolute middle line of the block. The absolute middle line is the middle position between the top line and the bottom line.
width	Specifies the width of the block in pixels. By default, the width is automatically determined to be the width necessary for the contents of the block. If the width attribute is specified, it will be used as the width of the block. If the width is insufficient for the contents, the contents will be wrapped into multiple lines.

CDML Examples

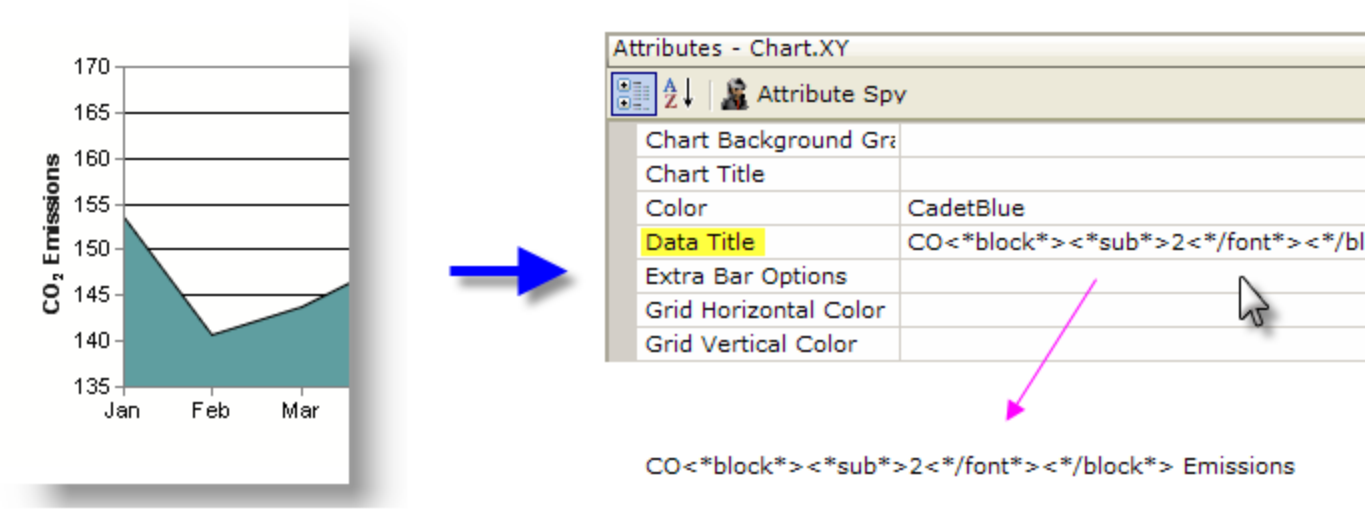
Here are some examples of CDML in action in Logi applications:

Creating a 2-Line Axis Title



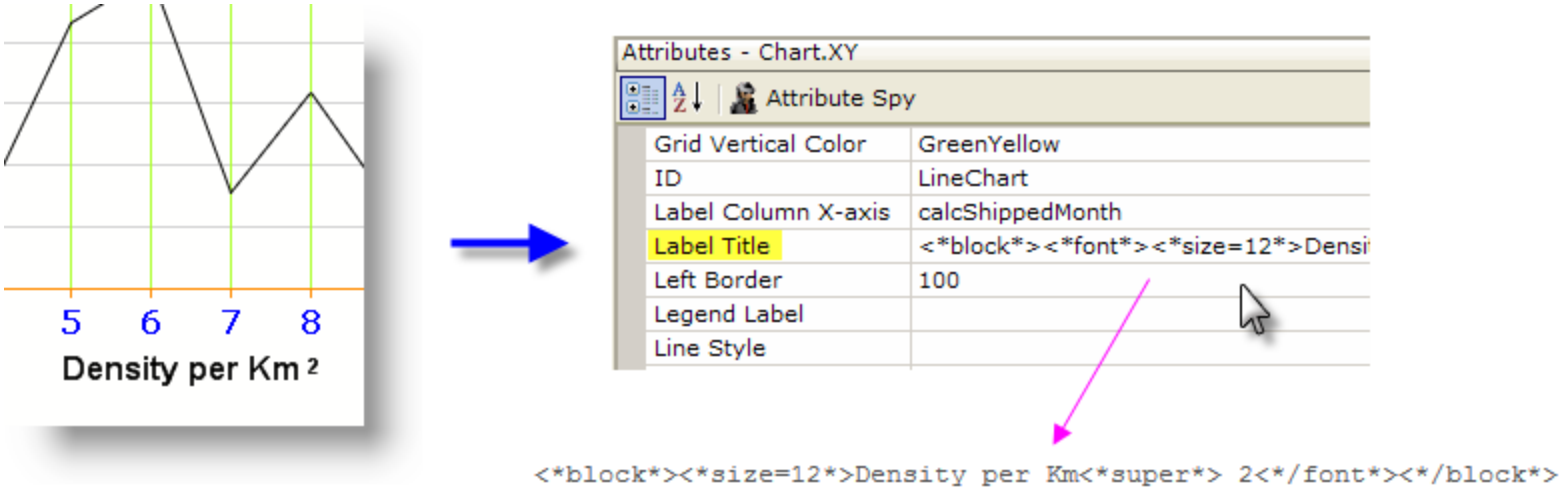
As shown above, a simple `<*br*>` tag is used in the **Label Title** attribute to produce a 2-line title.

Using Subscripts in an Axis Title



1. As shown above, the `<*block*>` and `<*/block*>` tags are used in the **Data Title** attribute to delimit the new font style.
2. The `<*sub*>` tag specifies the new style, i.e. subscript.
3. The `<*/font*>` tag ends the new style and returns to the original style.

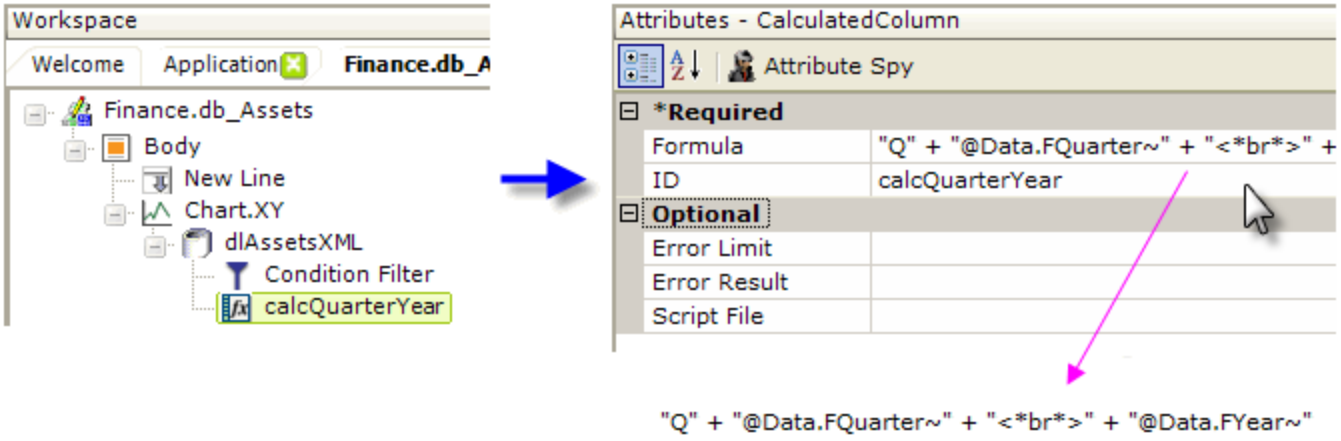
Setting Font Size and Adding Superscript in an Axis Title



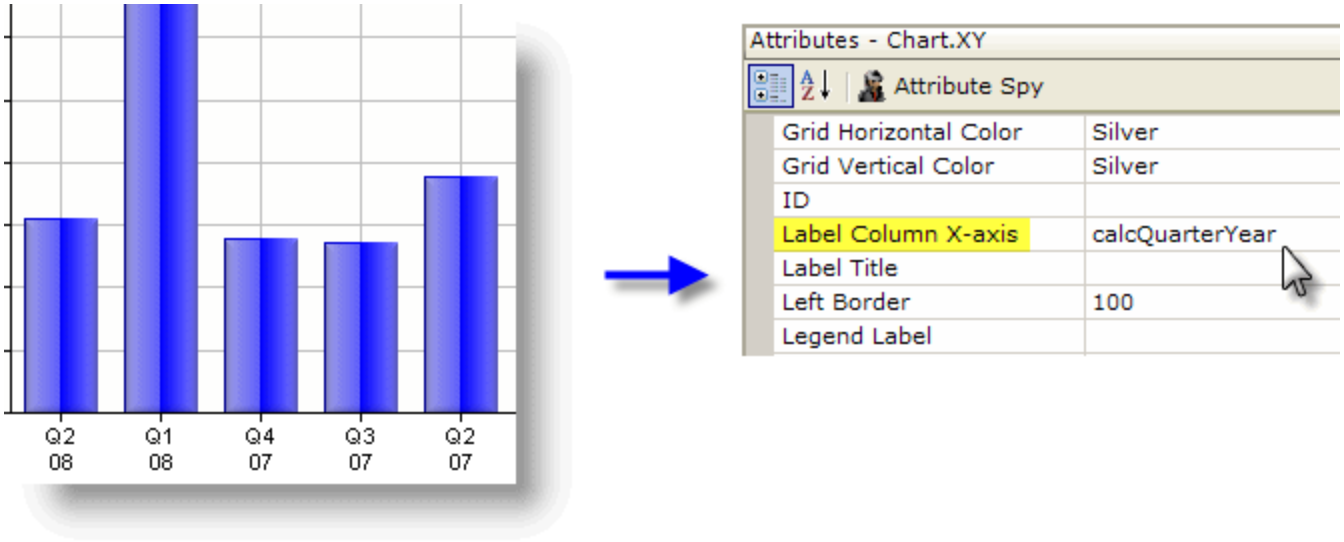
1. As shown above, the `<*block*>` and `<*/block*>` tags are used in the **Label Title** attribute to delimit the new font style.
2. The `<*size*>` and `<*super*>` tags are used to define the new font styles.
3. The `<*/font*>` tag ends the new style and returns to the original style.

Creating a 2-Line Data Label

To format data using CDML, it's necessary to first add a column to the datalayer to contain the formatted data.



1. As shown above, a **Calculated Column** is used to combine the data from two columns and add a `<*br*>` tag in between them to put the data on two lines.



2. The chart's **Label Column X-axis** attribute value is then set to the name of the calculated column, as shown above.

Alternately, instead of using a Calculated Column element, the results shown in the example above could also be achieved by **includingCDML** right in the **SQL query**. For example:

`SELECT FQuarter + '<*br*>' + RIGHT(FYear,2) AS calcQuarterYear FROM SomeSQLTable` would return a column that concatenated the values from two others with the CDML tag needed to insert the new line.

Export Chart Canvas Charts

Chart Canvas Charts can be exported in several ways.

- Exporting to .PDF Document
- Exporting to .PNG Image

Exporting to .PDF Document

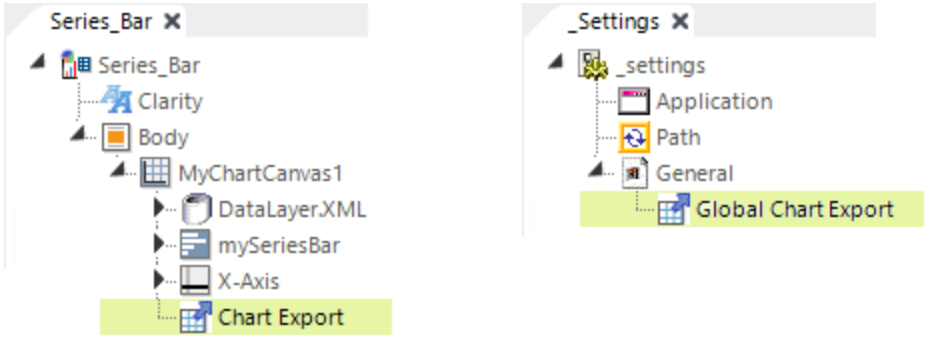
If you choose to export Chart Canvas charts as part of a complete report page, using Action.Export PDF and Target.PDF elements, the charts are exported as SVG objects rather than as images. This results in Chart Canvas charts exported to a PDF document having extremely high resolution - they can be zoomed or printed with high-quality at any resolution.

When exporting Chart Canvas charts to PDF, the **Chart Canvas** element *must* have Height and Width attribute values set.

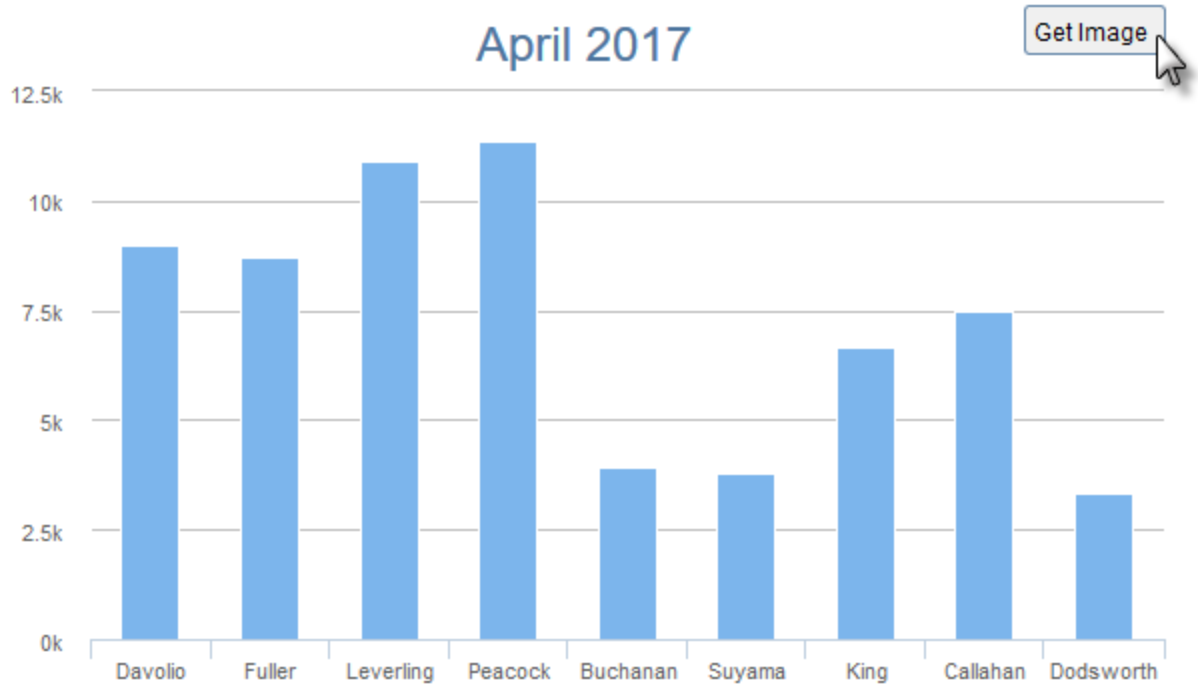
The attributes of the Action and Target elements can be configured to specify a variety of things that relate mostly to the overall report rather than to any charts they contain. During the process, however, a "hidden" copy of the report page is generated and used as the source for the PDF export engine. Ensure that this copy receives all of the parameters used to generate the visible page by using a Link Parameters element, a child of the Action element, and passing necessary values.

Exporting to .PNG Image

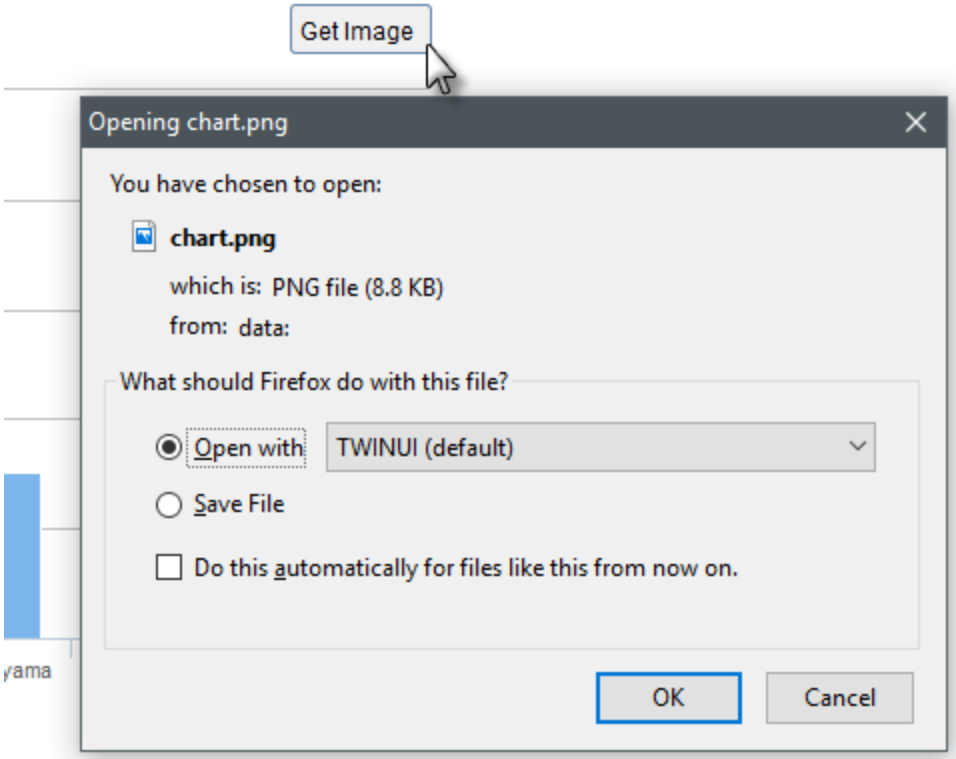
Chart Canvas charts can be configured to include an "export to .PNG image" feature.



This is done by adding either a **Chart Export** element as a child of a specific Chart Canvas Chart element, as shown above left, or by adding a **Global Chart Export** element as a child of the General element in the _Settings definition, as shown above right. Adding either Chart Export element enables the feature in the chart:



When enabled, hovering your mouse cursor over the upper right-hand corner of the chart will cause the **Get Image** button to appear. Click it to proceed.



You're essentially *downloading* the image, so the browser will behave in whatever manner it's been configured for when handling downloads. It may automatically save the image to its Downloads folder or, as shown above, you may be prompted to open (view) the image or select a location for saving it.

Attributes

Both of the Chart Export and Global Chart Export elements have an optional **Export Filename** attribute that allows you to specify a name for the downloaded .PNG image file and it accept tokens, so you can dynamically include time stamps, user names, etc. in it. If not specified, then the file is simply downloaded as "chart.png". The Chart Export element also allows you to optionally specify an **ID** and a **Security Right ID**. When using Logi Security, the latter allows you to control who can export a chart image.

Child Elements

Two child element are available that let you style the Get Image button. The **Chart Export Button Style** element lets you position, style, and customize the button, and the **Chart Export Button Hover Style** element lets you control the appearance of the button when the mouse cursor is hovered over it.

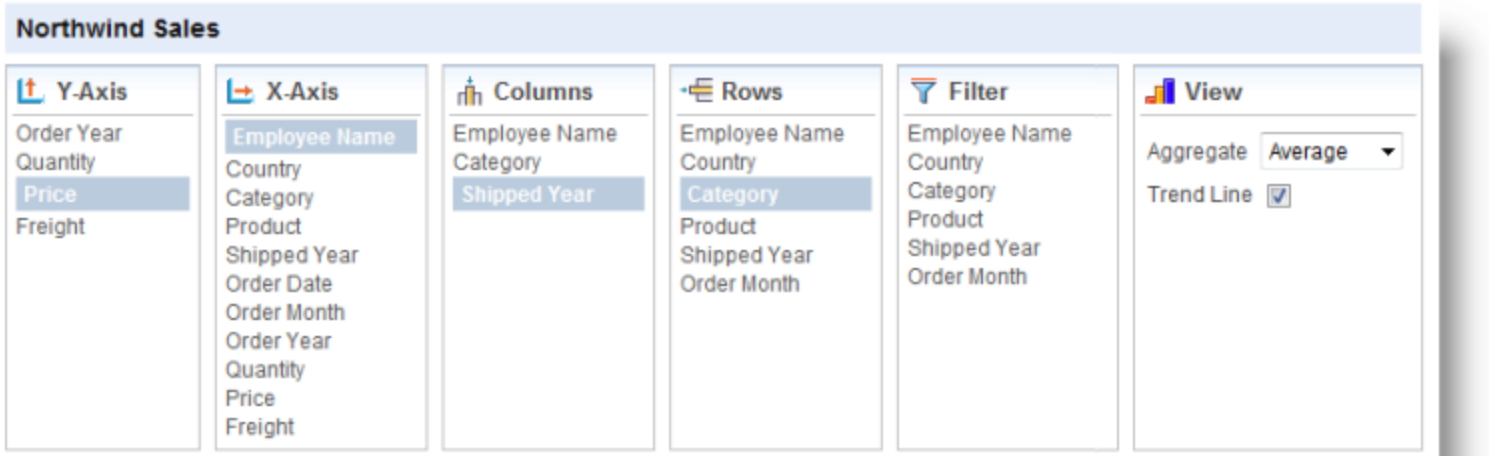
The Chart Grid

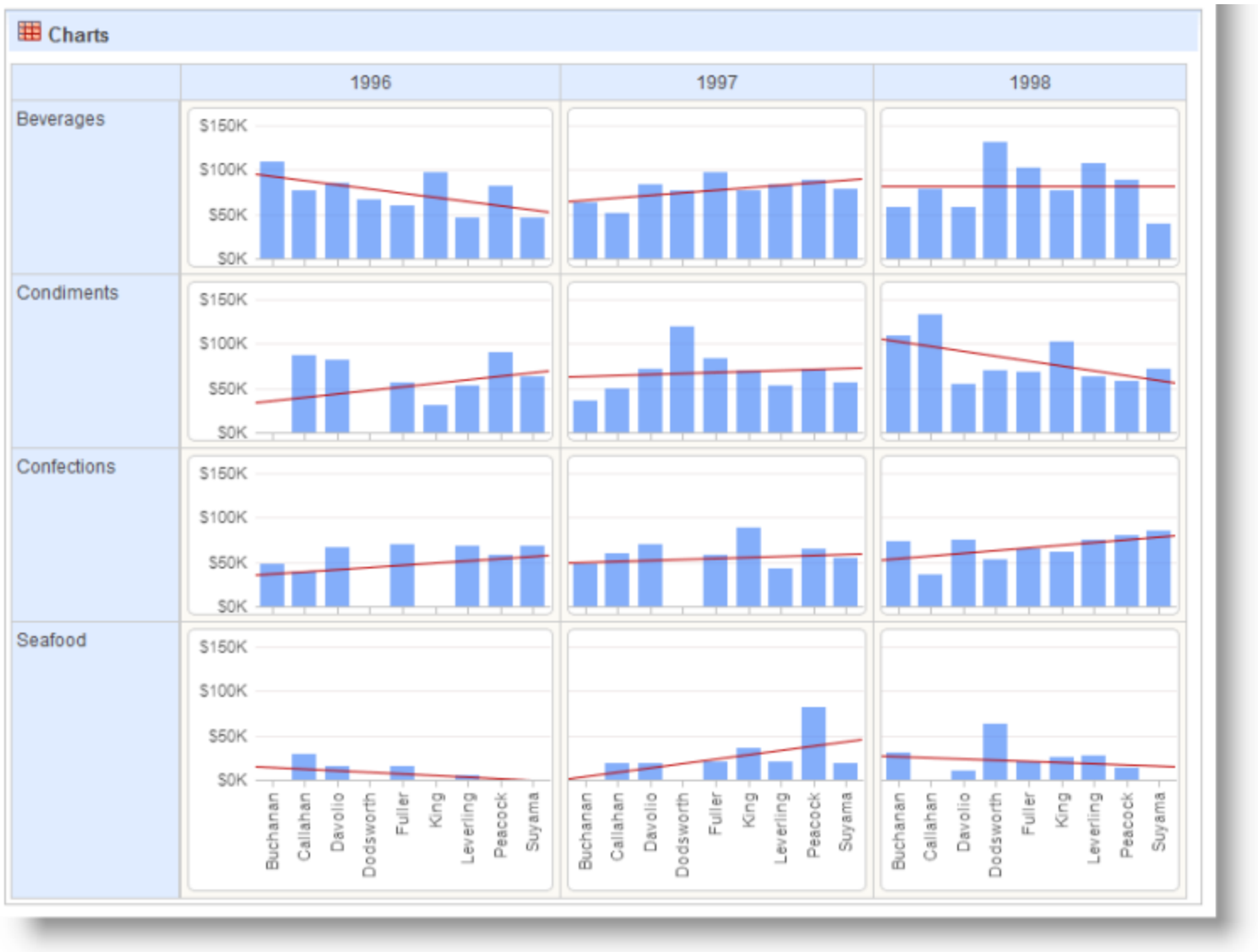
The Chart Grid is a tabular arrangement of charts, wherein each chart fills a table cell. It allows "at-a-glance" comparisons of complex data. As a "super element", the Chart Grid includes a control panel that allows users to customize the table and charts at runtime.

The following topics discuss the use of the Chart Grid:

About the Chart Grid

The **Chart Grid** displays a tabular arrangement of bar, line, or scatter charts, with each table cell containing a separate chart. All of the charts in the grid will be of the same type; it is not possible to mix chart types in different rows or columns. The Chart Grid is a "super-element", similar to the **Analysis Grid** and the **Dimension Grid** elements. At runtime, users can manipulate the data they see and the way in which it's presented using the Chart Grid's user interface controls. The report developer has complete control over which controls are available to a user and can present them selectively to different users.

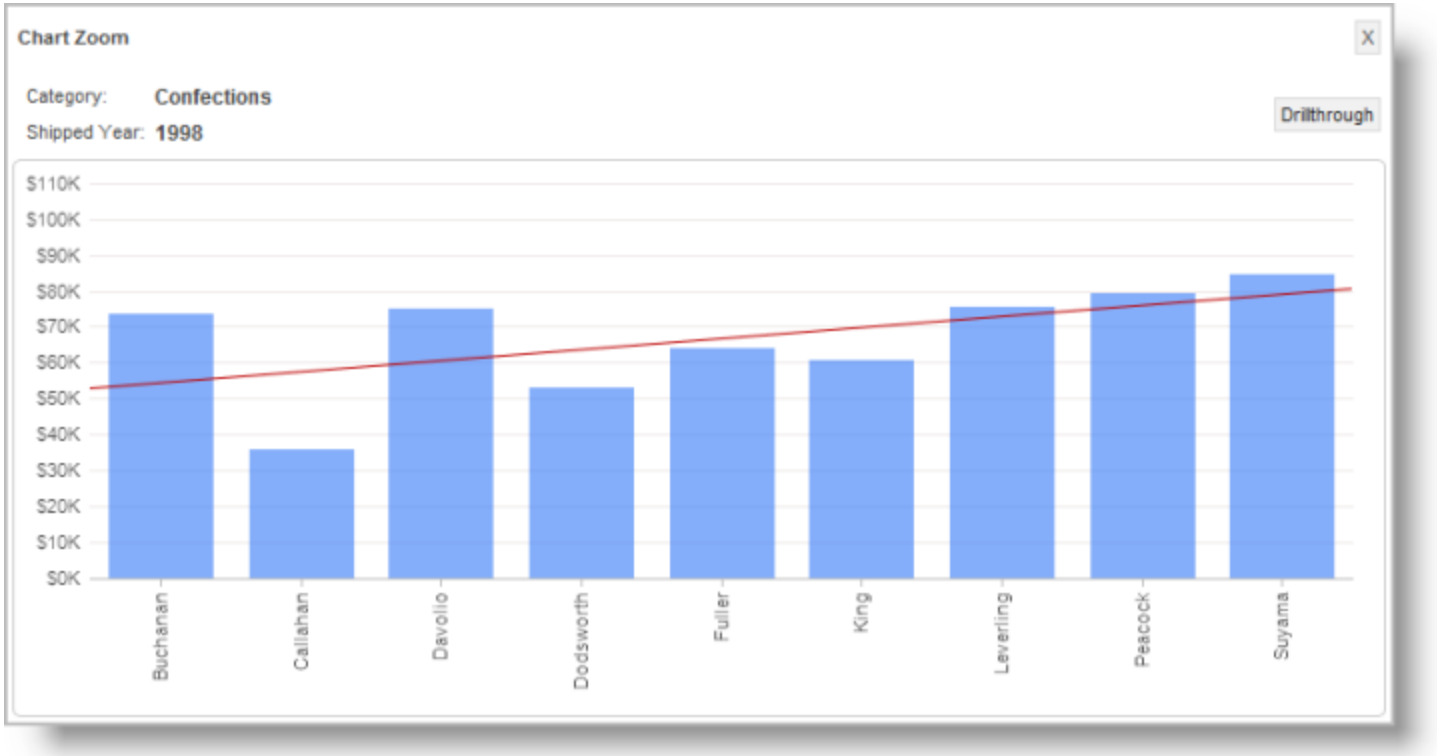

Northwind Sales
Y-Axis
Order Year
Quantity
Price
Freight
X-Axis
Employee Name
Country
Category
Product
Shipped Year
Order Date
Order Month
Order Year
Quantity
Price
Freight
Columns
Employee Name
Category
Shipped Year
Rows
Employee Name
Country
Category
Product
Shipped Year
Order Month
Filter
Employee Name
Country
Category
Product
Shipped Year
Order Month
View
Aggregate
Average
Trend Line



The example above shows a Chart Grid configured to show all of its option panels. Each chart resides in the grid at the intersection of a grid row and grid column. The data used to generate any chart is conditioned by filtering the complete dataset so that it only

includes data represented by the grid row and grid column intersection. Grid row titles and column headings show the actual data values used for this filtering. Chart Y-axis labels only appear in the first chart column and the X-axis labels only appear in the bottom row. The chart type generated is dependent on the X-axis data type selected: text-type columns produce bar charts and numeric- or date-type columns produce line or scatter charts (selectable by user). A technique for using dates with bar charts is explained later in this topic.

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



Clicking on an individual chart "zooms" the chart in a popup window, as shown above,

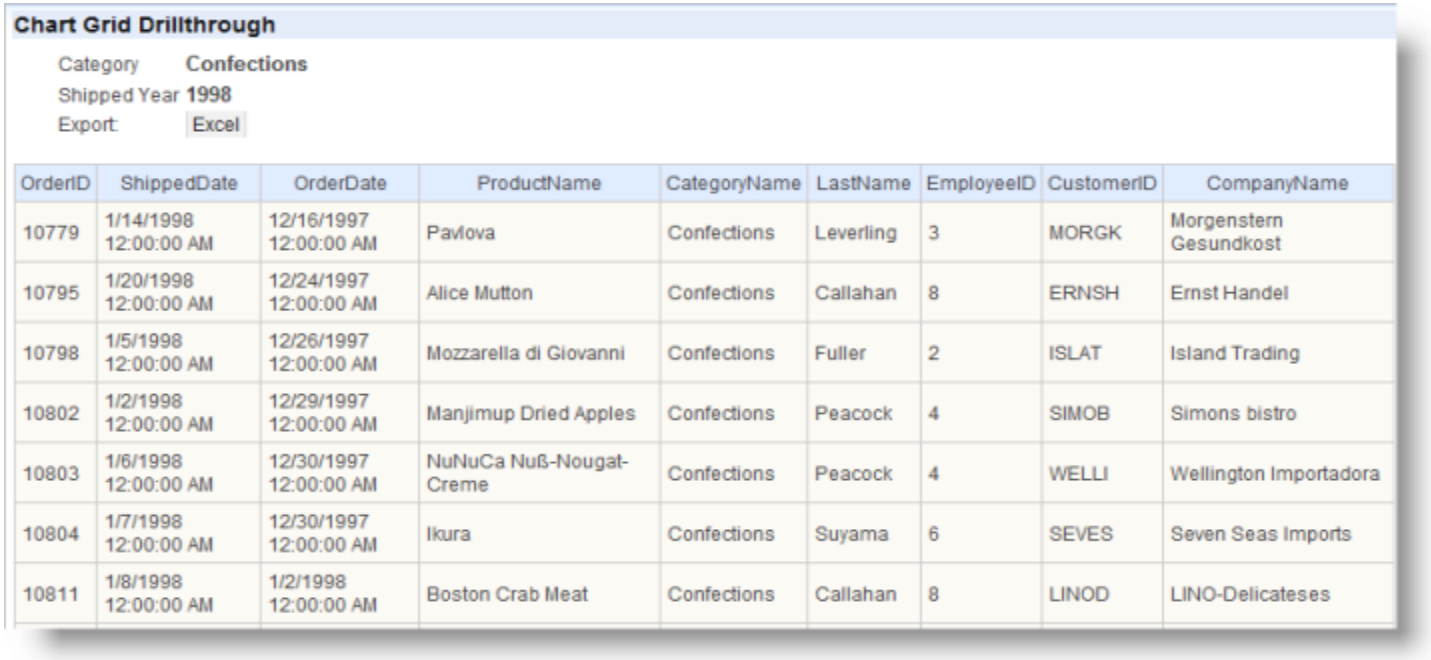


Chart Grid Drillthrough

Category **Confections**
Shipped Year **1998**
Export: Excel

OrderID	ShippedDate	OrderDate	ProductName	CategoryName	LastName	EmployeeID	CustomerID	CompanyName
10779	1/14/1998 12:00:00 AM	12/16/1997 12:00:00 AM	Pavlova	Confections	Leverling	3	MORGK	Morgenstern Gesundkost
10795	1/20/1998 12:00:00 AM	12/24/1997 12:00:00 AM	Alice Mutton	Confections	Callahan	8	ERNSH	Ernst Handel
10798	1/5/1998 12:00:00 AM	12/26/1997 12:00:00 AM	Mozzarella di Giovanni	Confections	Fuller	2	ISLAT	Island Trading
10802	1/2/1998 12:00:00 AM	12/29/1997 12:00:00 AM	Manjimup Dried Apples	Confections	Peacock	4	SIMOB	Simons bistro
10803	1/6/1998 12:00:00 AM	12/30/1997 12:00:00 AM	NuNuCa Nuß-Nougat-Creme	Confections	Peacock	4	WELLI	Wellington Importadora
10804	1/7/1998 12:00:00 AM	12/30/1997 12:00:00 AM	Ikura	Confections	Suyama	6	SEVES	Seven Seas Imports
10811	1/8/1998 12:00:00 AM	1/2/1998 12:00:00 AM	Boston Crab Meat	Confections	Callahan	8	LINOD	LINO-Delicateses

and clicking the zoomed chart's **Drillthrough** button displays a table, shown above, of the data underlying the chart. The zooming and drill-through functions are automatic and access to them can be controlled by the developer. An export button on the table allows the data to be exported to Excel. The Chart Grid settings selected by a user can be saved at the end of their session and restored during a later session. Logi Studio contains a Chart Grid wizard that will walk developers through creating and configuring the grid. For information about this wizard, see "The Chart Grid Wizard" on page 207.

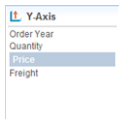
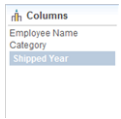
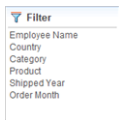
Chart Grid Element

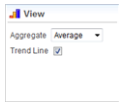
The **Chart Grid** element is the parent element of all other Chart Grid family elements. Its special attributes are:

Attribute	Description
ID	(Required) A unique element ID.
Caption	A title that will appear in the bar above the control panels.
Saved Chart Grid Folder	The Chart Grid settings made by a user at run time may be saved with by using the Procedure.Save Chart Grid element in a process task. The settings can later be reloaded by passing the saved setting filename in the rdCgLoadSaved parameter. The value of this attribute (Saved Chart Grid Folder) sets the name of the folder that contains the saved files.
Template Modifier File	The name of a custom template file that can be used to change language- and culture-specific Caption attributes in the grid.

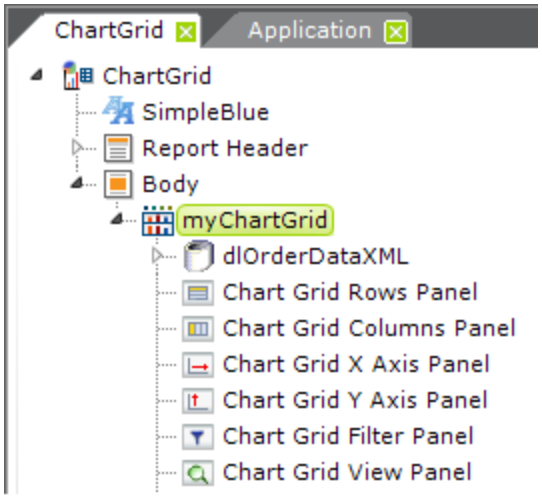
Chart Grid Settings Panel

The **Chart Grid Panel** elements determine what data and which options are available to the user at runtime. Let's examine the panels:

Panel	Description
	Links in this panel allow the user to select the data available in the datalayer that will be used for the charts' Y-axis values. Chart Y-axis labels only appear in the first chart column. The developer can limit the choices available here and can also designate an initial selection.
	Links in this panel allow the user to select the data available in the datalayer that will be used for the charts' X-axis values. Chart X-axis labels only appear beneath charts in the bottom table row. The developer can limit the choices available here and can also designate an initial selection.
	Links in this panel allow the user to select the data column that will be used for the grid columns. Its distinct values will appear as grid column header text.
	Links in this panel allow the user to select the data column that will be used for the grid rows. Its distinct values will appear as grid row title text.
	Links in this panel allow the user to filter the complete dataset by selecting values to be included from a popup list of data values.  This works by *inclusion*, not exclusion, so to have no filtering, *all* values must be selected.

Panel	Description
	This panel contains dynamic content. When X-axis chart data is text-type data, it's automatically grouped and the Y-axis values aggregated before a chart is generated. The user may select the aggregating function in this panel. Options include *Sum*, *Average*, *Count*, *Minimum*, and *Maximum*. When X-axis chart data is numeric- or date-type data, the Aggregate select list is replaced with a Chart Type select list, allowing users to select between Line or Scatter charts. This panel also allows the user to include a trend line on the charts. Columns with Null values are *excluded* by default. To include nulls, create the constant `rdCalculationsIncludeNulls` and set it to *True*.

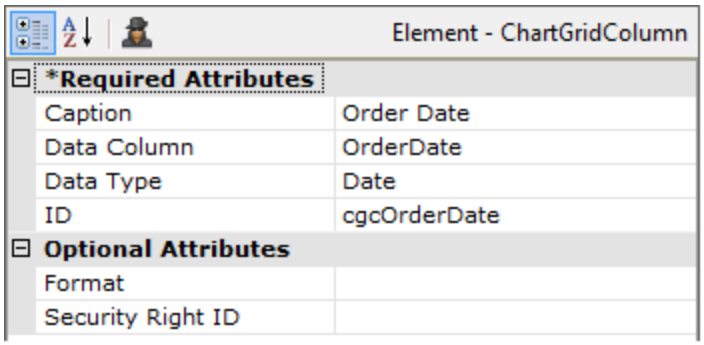
The **report definition** for a Chart Grid, with all its panel elements, looks like the example below:

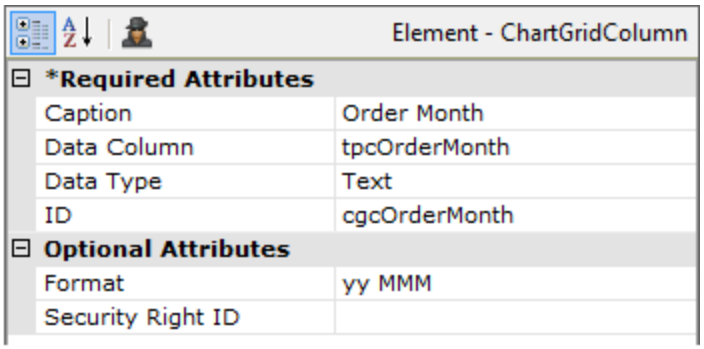


Regardless of the arrangement of the elements, the panels will always be displayed in the browser *above* the grid. Chart Grid Panel elements are *not* required. By default, if you leave them out, all panels will be displayed. To hide a panel, include its grid panel element and set its **Show Panel** attribute to *False*. This value can also be set using @Request and @Session tokens. Each Chart Grid Panel element has an **Allow ChartGrid Columns** attribute, which specifies one or more Chart Grid Columns to be displayed in this panel. If no columns are specified, then all columns of the matching data type are displayed; multiple columns can be enumerated using a comma-separated list of Chart Grid Column element IDs. Chart Grid Panel elements also have an **Initial ChartGrid Column ID** attribute, which specifies which Chart Grid Column will be selected when the grid is first displayed.

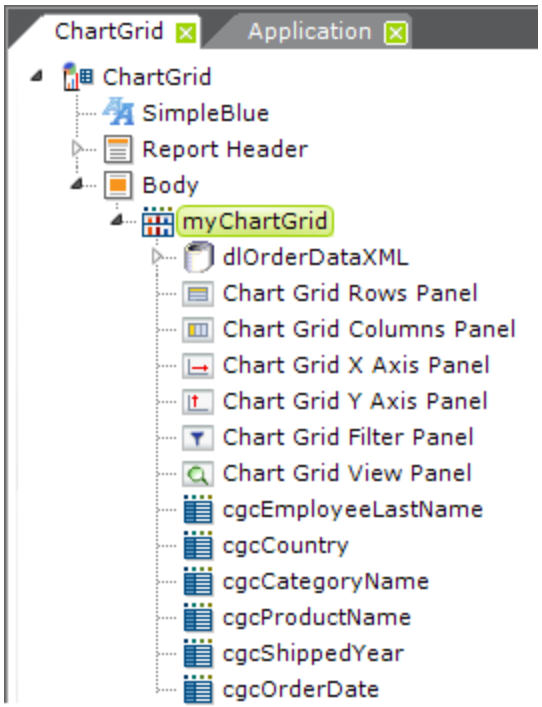
Configuring the Data

The Chart Grid requires only one datalayer element to retrieve data for it and it produces different views of the data for each chart automatically. **Chart Grid Column** elements are used to map datalayer columns to grid columns, defining which columns may be displayed and manipulated in the Chart Grid. The element's **Data Type** attribute effects how columns are presented in the grid. Text-type columns are available for use in the Columns, Rows, and X-axis panels, and produce Bar charts; number- and date-type columns are available for use in the X- and Y-axis panels, and produce Line or Scatter charts when selected for the X-axis.





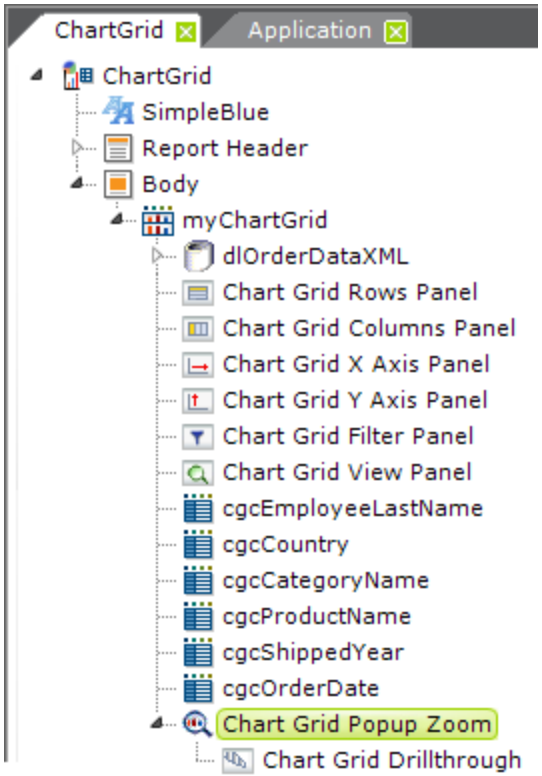
In the example above, the left image shows a typical date-type Chart Grid Column which, when used in the X-axis will produce a Line or Scatter Chart. However, if you wish to use a date in the X-axis of a Bar chart, you can do so by adding a **Time Period Column** element ("tpcOrderMonth") to your datalayer, naming it in a Chart Grid Column element as shown above, right, setting the data type to *Text,* and providing a format for the date.



So now our report definition includes Chart Grid Column elements for each data column we want to work with, as shown above, and we have a basic functional Chart Grid that can be run.

Chart Zoom and Drill-through

An optional feature, is the ability to click a chart to view it in a larger scale in a separate window, and from there to drill down into the detail data used to generate the chart.



The feature is included by adding a **Chart Grid Popup Zoom** element, as shown above, to the definition. Its sole attribute is for optional tooltip text. When the optional **Chart Grid Drillthrough** child element is included, a Drillthrough button appears in the zoomed chart. Clicking this button causes the detailed, exportable Data Table to be displayed. This element has an optional **Caption** attribute which allows you to specify a title for the Data Table.

Refresh, Reset, and Save Grid Settings

There are several possible actions that require links outside of the grid which you might want to make available to Chart Grid users:

For This Action	Do This
Refresh	To refresh the data in the grid, without affecting any other settings, call the report definition again and include this Link Param: `rdCgRefreshData = True` The parameter name is case-sensitive.
Reset	To clear all user settings and restore the grid to its default state, call the report definition again and include this Link Param: `rdCgReset = True` The parameter name is case-sensitive.
Save Settings	To save all user settings for future sessions, use an Action.Process element to call a process task. In the task use a **Procedure.Save Chart Grid** element to save the settings. The file name itself can incorporate a login name, a GUID, or another method of making it unique to the user and must end in ".xml". The file name can be preserved in a number of ways, including as a Cookie on the user's machine.
Restore Settings	Access the preserved file name then call the report definition again and pass the Link Param: rdCgLoadSaved = *<filename>* Once again, the parameter name is case-sensitive.

The following XML source code example demonstrates the use of these special parameters:

```
<Division ID="divControlMenu">
 <LineBreak LineCount="2" />
 <Label Caption="Reset" ID="lblReset">
 <Action Type="Report" ID="actRestart">
```

```
<Target Type="Report" ID="tgtRestart" />
<LinkParams rdCgReset="True" />
</Action>
</Label>
<Label Caption=" | " ID="bar" />
<Label Caption="Refresh Data" ID="lblRefreshData">
<Action Type="Report" ID="actRefreshData">
<Target Type="Report" ID="tgtRefresData" />
<LinkParams rdCgRefreshData="True" />
</Action>
</Label>
<Label Caption=" | " ID="bar" />
<Label Caption="Save" ID="lblSave">
<Action Type="Process" Process="MyTasks" ConfirmMessage="Save Grid?" TaskID="SaveCG" ID="actSave"/>
</Label>
<Label Caption=" | " ID="bar" />
<Label Caption="Restore" ID="lblRestore">
<Action Type="Report">
<Target Type="Report" />
<LinkParams rdCgLoadSaved="@Cookie.CgFilename~" />
</Action>
</Label>
<LineBreak LineCount="2" />
</Division>
```

When saving the grid settings in a Process task, using the **Procedure.Save Chart Grid** element, you need to provide a fully-qualified file path and file name (including an .xml file extension) in the **Filename** attribute. For example:

```
C:\inetpub\wwwroot\SampleChartGrid\SavedGridData\mySettings.xml
```

The @Function.AppPhysicalPath~, @Session, and @Constant tokens may be useful here. However, when restoring saved settings, two separate values are used so that you do not have to expose the file path in your query string:

1. Set the Chart Grid element's **Saved Chart Grid Folder** attribute to the fully-qualified file path for the *folder* that contains the stored settings file. For example: `C:\inetpub\wwwroot\SampleChartGrid\SavedGridData`. The @Function.AppPhysicalPath~ and @Constant tokens may be useful here.
2. In the Process task that restores the saved settings, call the report definition that contains the Chart Grid and pass the link parameter **rdCgLoadSaved** mentioned earlier, and set its value to the filename, with .xml extension, only. Do not include any path information here. For example: `mySettings.xml`

The Chart Grid will automatically combine the attribute value and the request variable to obtain the fully-qualified path and filename for the saved settings file.

The Chart Grid Wizard

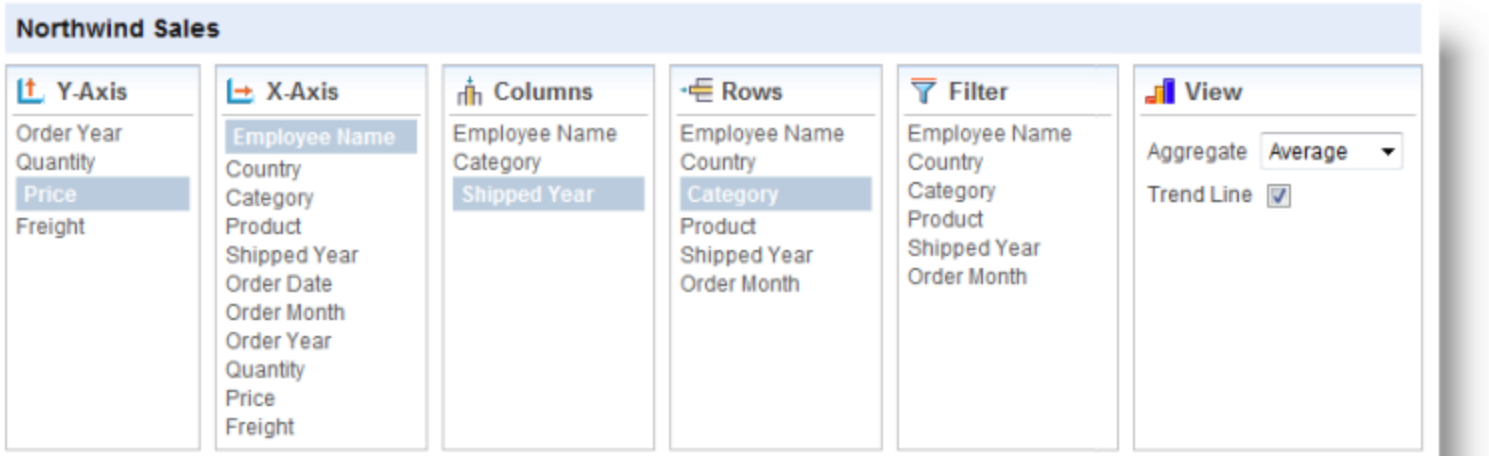
The Chart Grid is a tabular arrangement of charts, wherein each chart fills a table cell. As a "super element", the Chart Grid includes an interface that allows users to customize the table and charts at runtime.

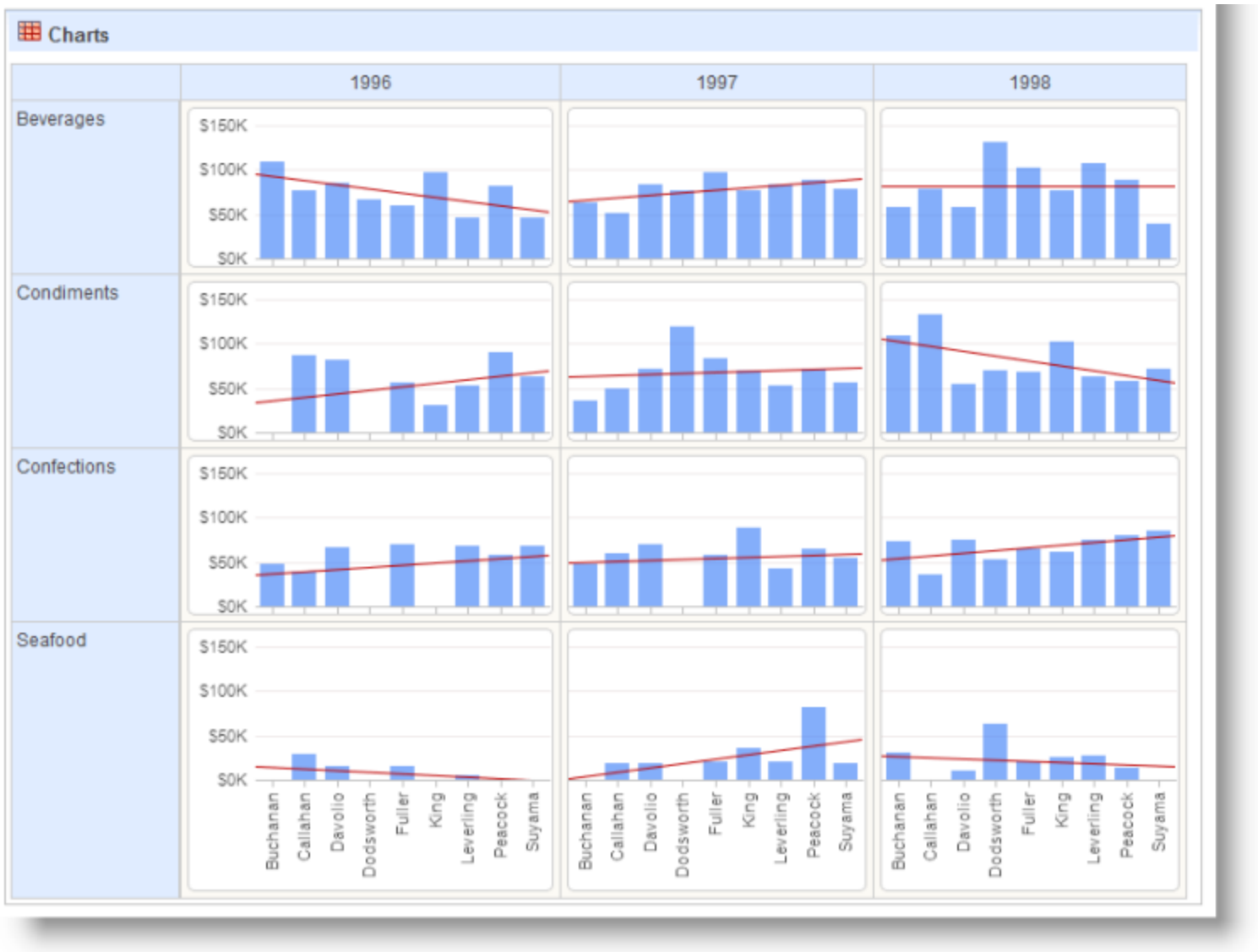
The following topics describe the Chart Grid Wizard in Logi Studio, which can be used to quickly create a Chart Grid:

- Preparing the Chart Grid Wizard
- Using the Chart Grid Wizard

About the Chart Grid

For information about the capabilities and features of the Chart Grid, see "The Chart Grid" on page 192.The **Chart Grid** displays a tabular arrangement of bar, line, or scatter charts, with each table cell containing a separate chart. All of the charts in the grid will be of the same type; it is not possible to mix chart types in different rows or columns. The Chart Grid is a "super-element", similar to the **Analysis Grid** and the **Chart Grid** elements. At runtime, users can manipulate the data they see and the way in which it's presented using the Chart Grid's user interface controls. The report developer has complete control over which controls are available to a user and can present them selectively to different users.


Northwind Sales
Y-Axis
Order Year
Quantity
Price
Freight
X-Axis
Employee Name
Country
Category
Product
Shipped Year
Order Date
Order Month
Order Year
Quantity
Price
Freight
Columns
Employee Name
Category
Shipped Year
Rows
Employee Name
Country
Category
Product
Shipped Year
Order Month
Filter
Employee Name
Country
Category
Product
Shipped Year
Order Month
View
Aggregate
Average
Trend Line



The example above shows a Chart Grid configured to show all of its option panels.

Preparing the Chart Grid Wizard

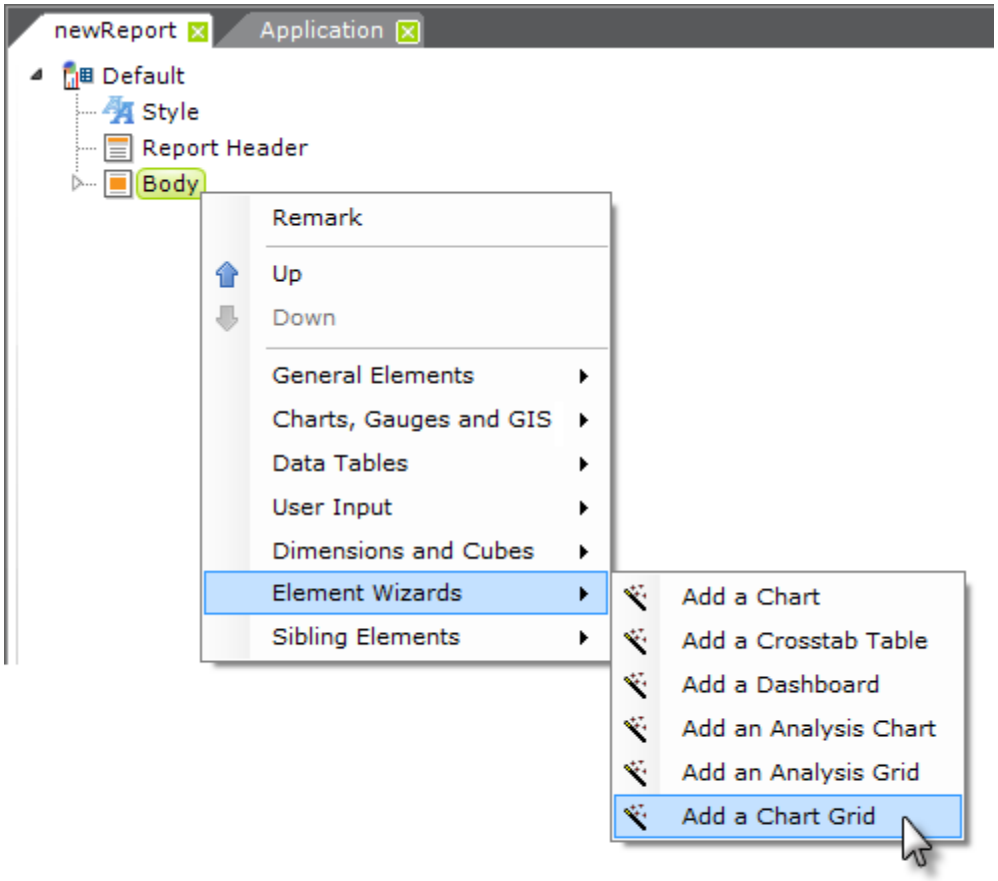
Prior to running the Chart Grid wizard:

- Become familiar with the data you'll be working with. Identify the data columns that you want to use, what aggregations you need, and how you might want to *filter* the data.
- Ensure that you have a valid **Connection** configured in _settings and can connect to your datasource.
- If you're using a SQL data source, you may care to experiment a bit in the Query Builder to develop the query that you'll need.

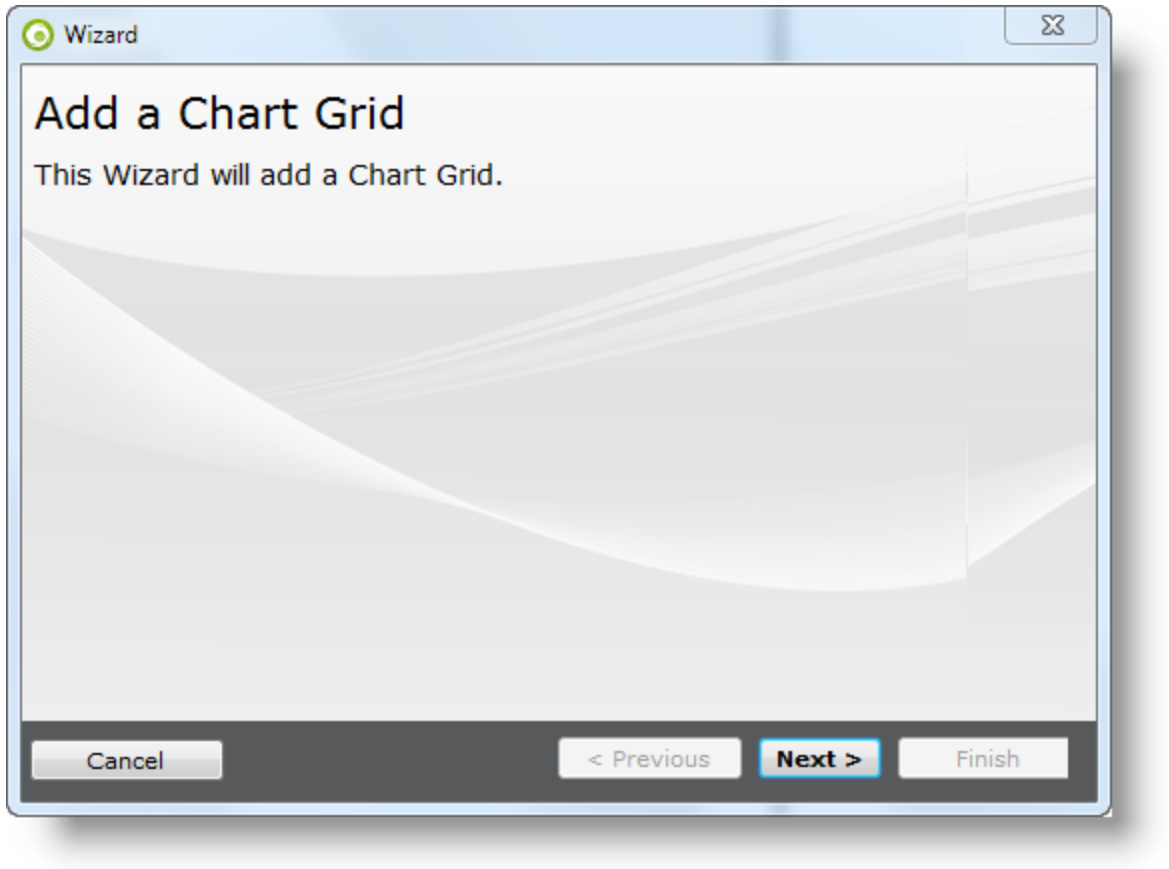
Now, launch Studio and open your report definition.

Using the Chart Grid Wizard

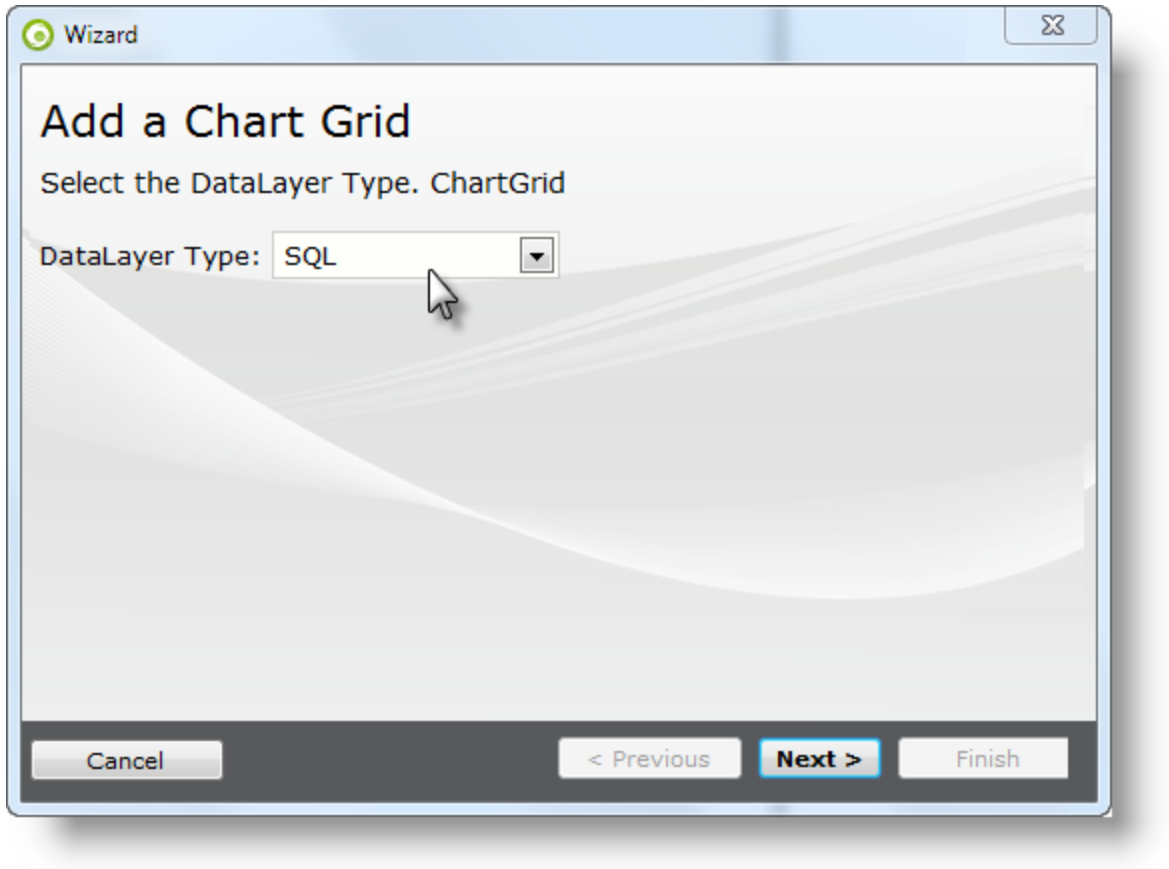
The following examples walk you through using the Chart Grid wizard.



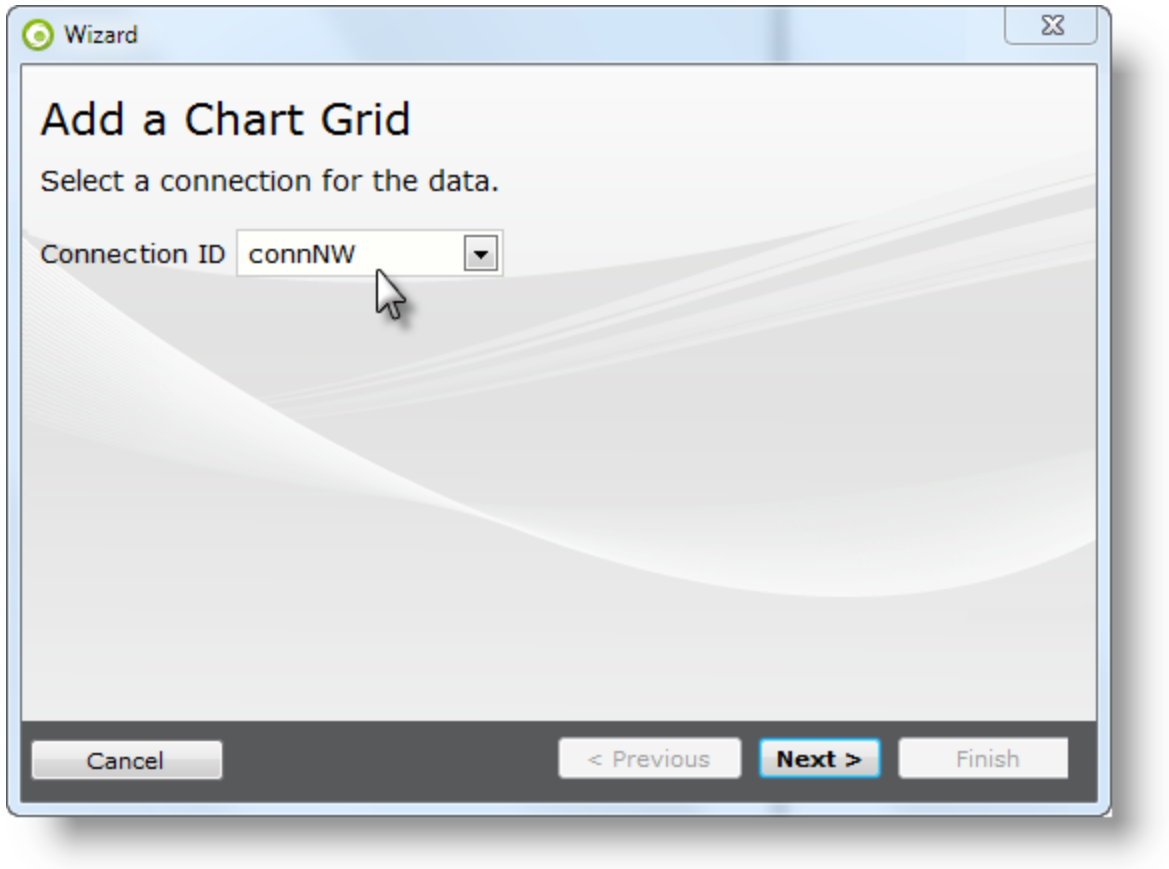
1. Start the wizard by right-clicking the grid's parent and selecting **Element Wizards→Add a Chart Grid**, as shown above.



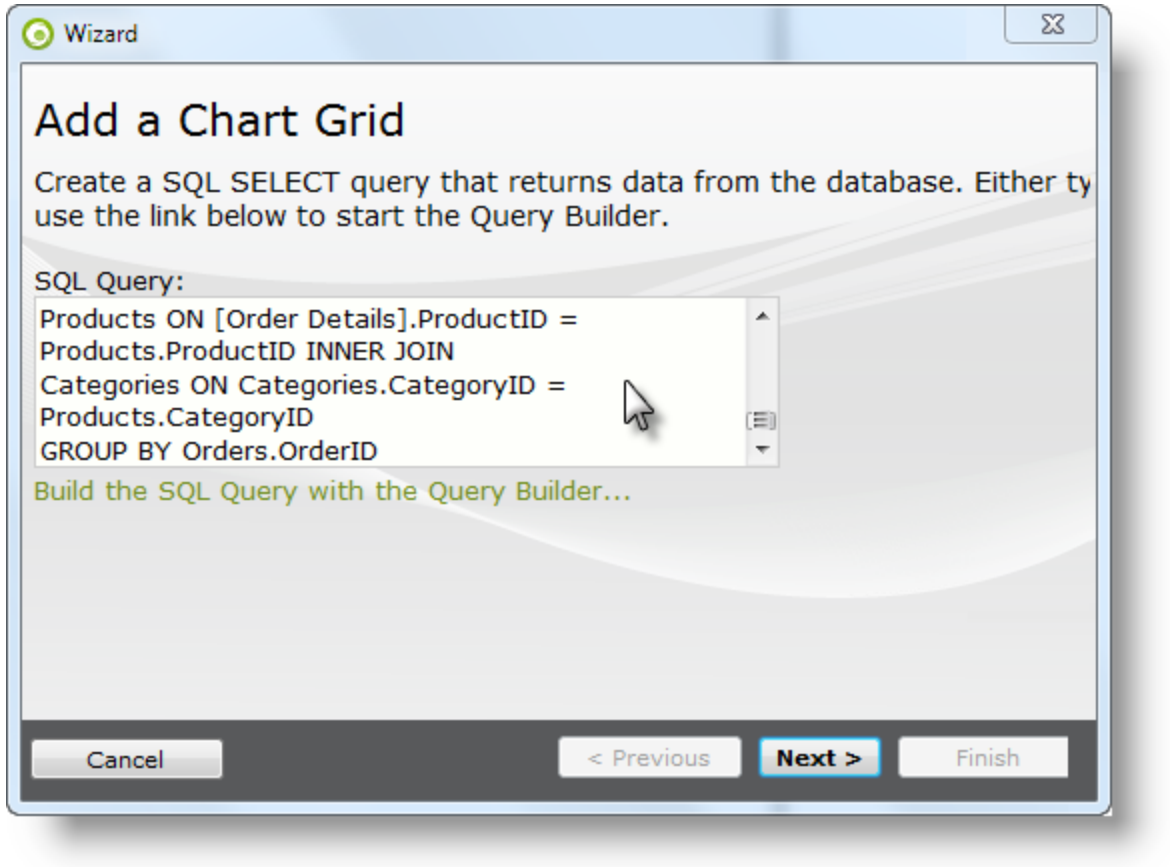
2. **Add aChart Grid Wizard** - Click **Next** to start the wizard.



3. **Datalayer Type** - Select the appropriate datalayer type for your datasource. Click **Next**.



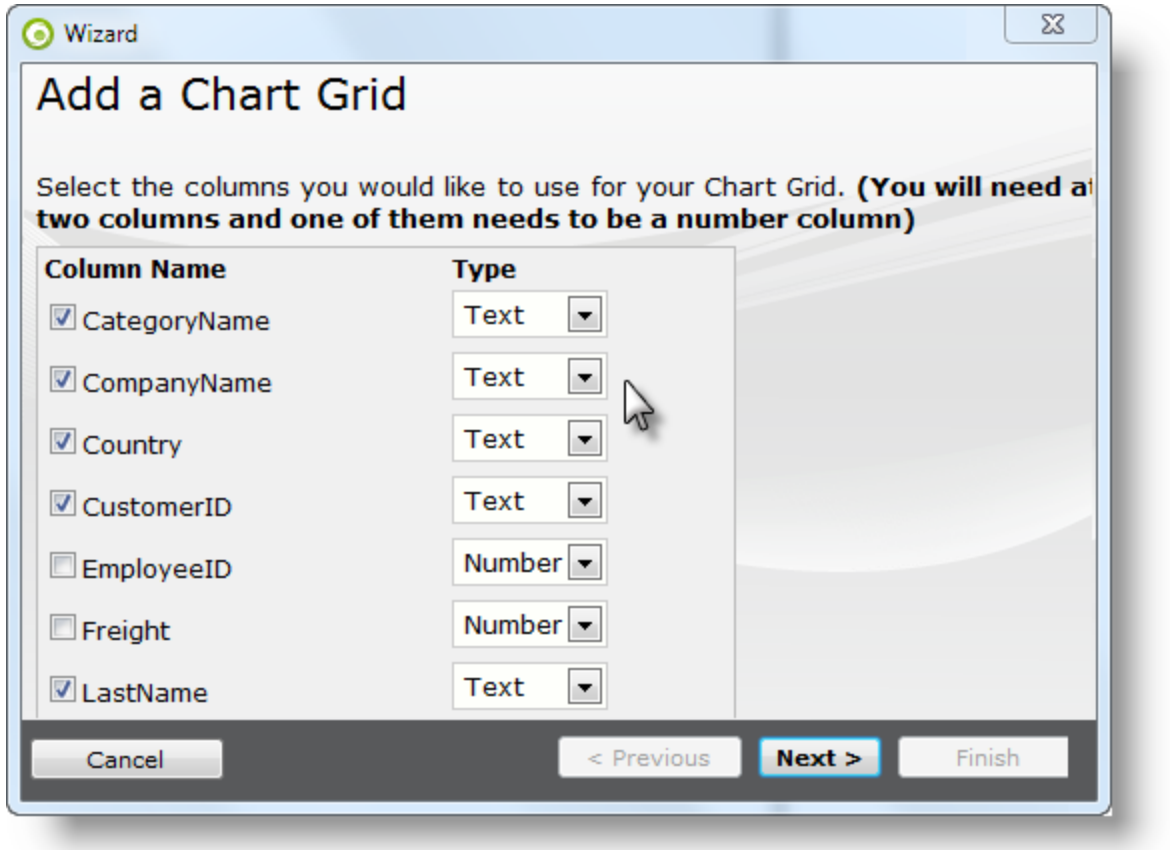
4. **Connection ID** - Select the desired connection from the list of connections defined in the _Settings definition. Click **Next**.



4. **SQL Query** - If you 're using a SQL datasource, enter the query (or use the query wizard to build one) and click **Next**. This example uses the following T-SQL query:

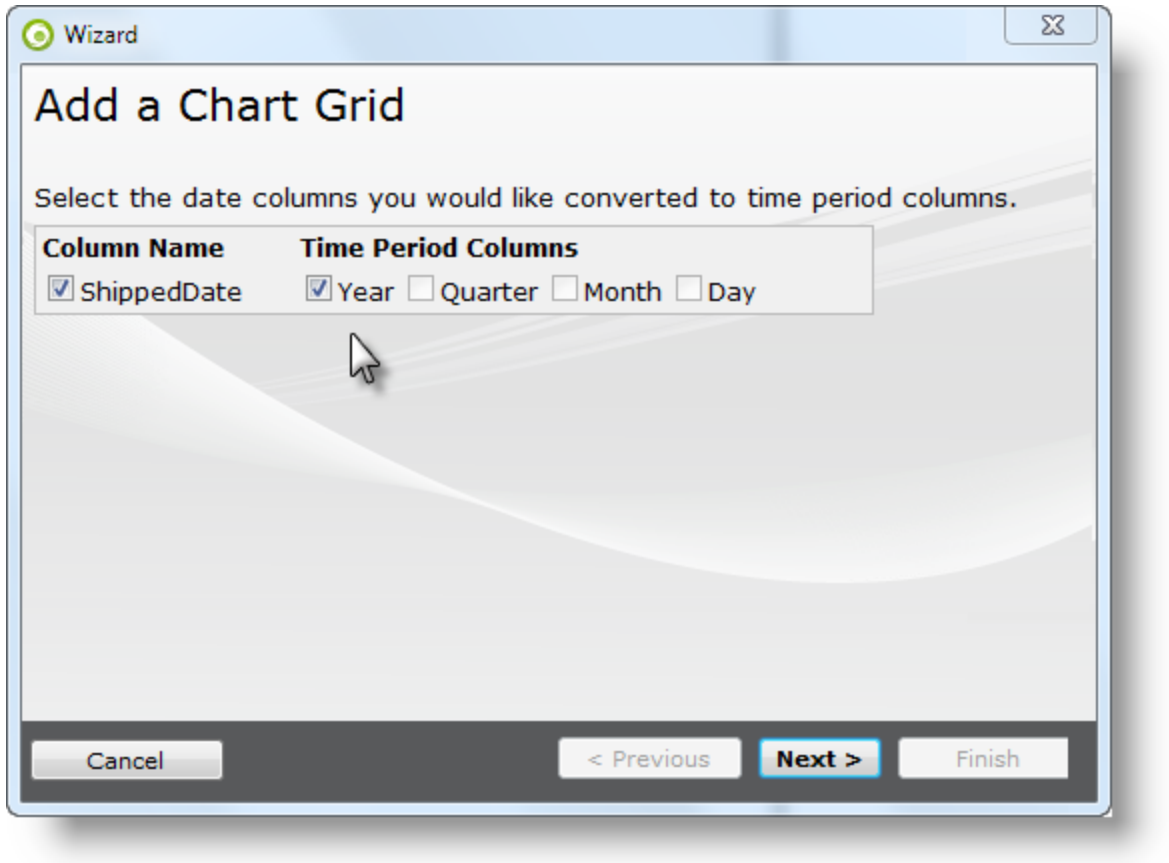
```
SELECT Orders.OrderID, MIN(Orders.Freight) AS Freight, MIN(Orders.ShippedDate) AS ShippedDate, MIN(Order-
s.OrderDate) AS OrderDate, SUM([Order Details].UnitPrice) AS sumUnitPrice, SUM([Order Details].Quantity) AS
sumQuantity, MIN(RTRIM(Products.ProductName)) AS ProductName, MIN(Categories.CategoryName) AS CategoryName, MIN
(RTRIM(Employees.LastName)) AS LastName, MIN(Employees.EmployeeID) AS EmployeeID, MIN(Orders.CustomerID) AS Cus-
tomerID, MIN(Customers.CompanyName) AS CompanyName, MIN(RTRIM(Customers.Country)) AS Country
FROM Orders INNER JOIN
```

```
[Order Details] ON [Order Details].OrderID = Orders.OrderID AND
Orders.OrderID = [Order Details].OrderID INNER JOIN
Employees ON Employees.EmployeeID = Orders.EmployeeID INNER JOIN
Customers ON Customers.CustomerID = Orders.CustomerID INNER JOIN
Products ON [Order Details].ProductID = Products.ProductID INNER JOIN
Categories ON Categories.CategoryID = Products.CategoryID
GROUP BY Orders.OrderID
```

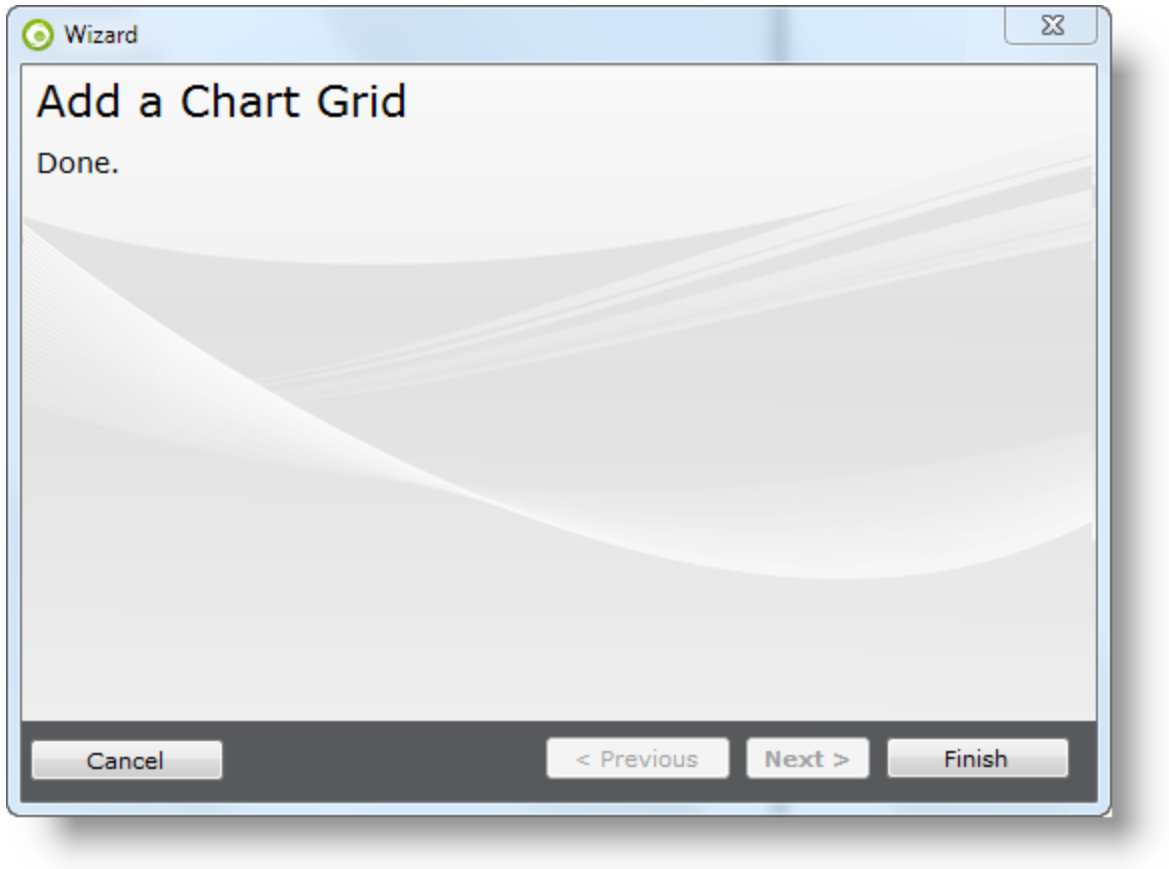


5. **Select Columns** - Select at least two data columns, one of which must be a numeric data type. Set the data type indicator for each selected column, then click **Next**. In this example, the columns selected are:

    - CategoryName
    - CompanyName
    - Country
    - CustomerID
    - LastName
    - ShippedDate
    - sumQuantity
    - sumUnitPrice

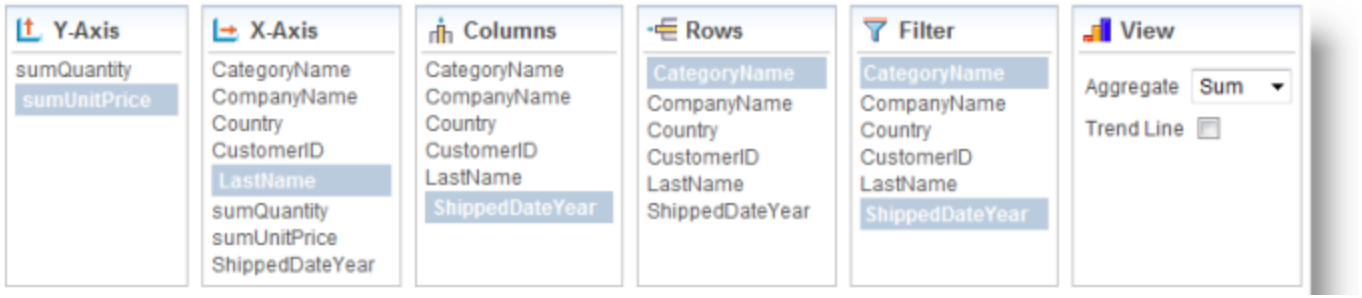


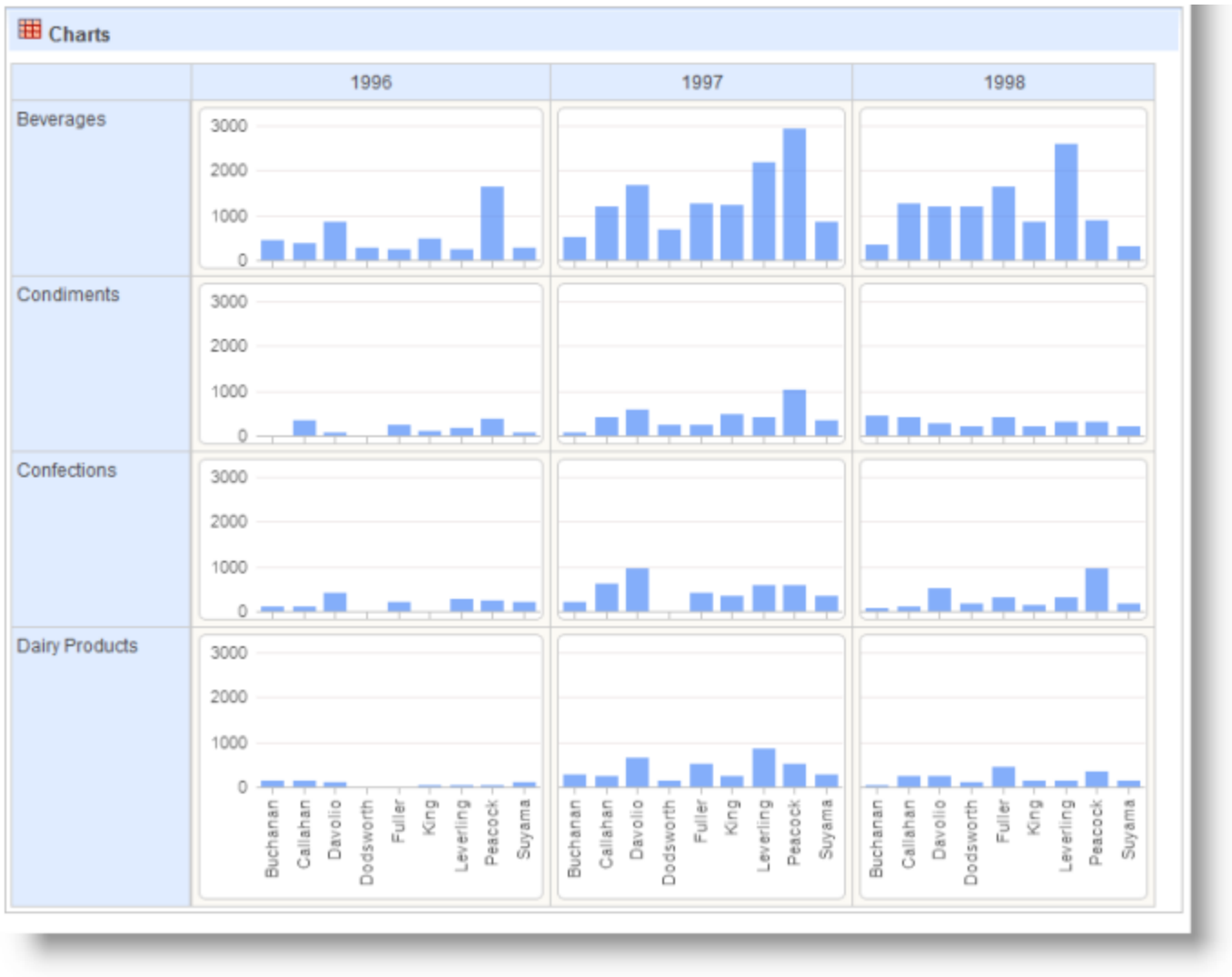
6. **Process Date Columns** - If your data includes date type data columns, the wizard will give you the option of including a Time Period Column for the Day, Month, Quarter, or Year represented in the data in each of them. This is necessary if you want to use date data in the X-axis labels of the charts. In this example, the Year portion of the ShippedDate column is used. Click **Next** to continue.



7. **Done** - Click **Finish** to close the wizard.

As you've completed these steps, you may have noticed that the wizard has added all the appropriate elements to your report definition. Save the definition and run it.


Y-Axis
sumQuantity
sumUnitPrice
X-Axis
CategoryName
CompanyName
Country
CustomerID
LastName
sumQuantity
sumUnitPrice
ShippedDateYear
Columns
CategoryName
CompanyName
Country
CustomerID
LastName
ShippedDateYear
Rows
CategoryName
CompanyName
Country
CustomerID
LastName
ShippedDateYear
Filter
CategoryName
CompanyName
Country
CustomerID
LastName
ShippedDateYear
View
Aggregate
Sum
Trend Line



When it appears, select Y-Axis, X-Axis, Column, and Row options, as shown above, and view your new Chart Grid. The example shown above has also had two filters applied to it, on the CategoryName (limits grid to four rows) and the ShippedDateYear (to remove null data). Finally, you should examine the elements inserted into the definition by the wizard to review formatting, fine-

tune captions, etc. Remember that Chart Grid configurations are automatically saved with session variables, which can lead to frustration if you're trying to make a lot of changes during development. You may need to reset these variables to make changes occur; to do this, see "Refresh, Reset, and Save Grid Settings" on page 204.

Dimension Grid Wizard

Logi Info includes a new technology called Logi **XOLAP** (pronounced "zo lap") that brings the analytical power of data cubes to your relational data and other diverse data sources such as web services, XML, and files.

The following topics describe the Dimension Grid Wizard in Logi Studio, which can be used to build Logi XOLAP data cubes:

- Preparing to Run the Dimension Grid Wizard
- Using the Wizard
- Sorting Dimensions by Dates

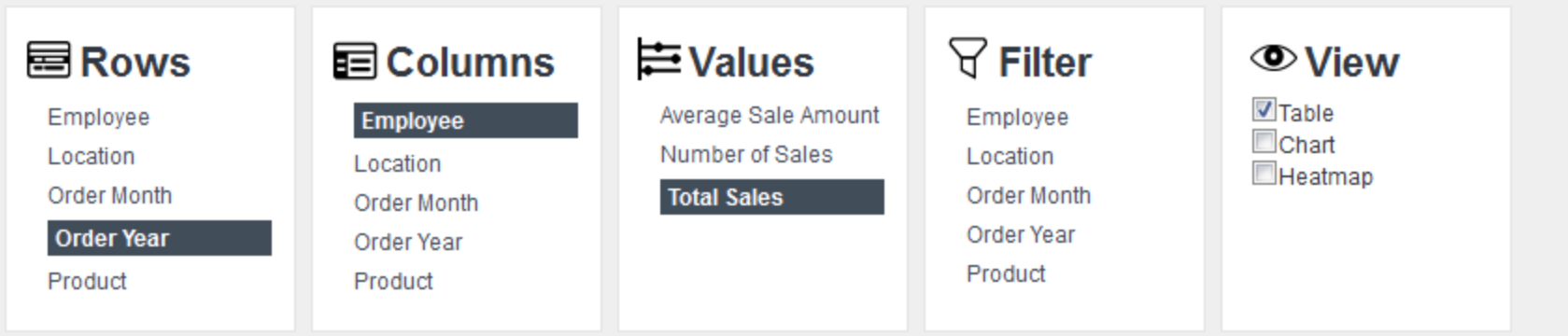
Logi XOLAP has been deprecated; the related elements are still supported, for now, and will work but they're no longer available in Logi Studio.

About Logi XOLAP

If you have not done so already, see *Logi XOLAP*for important basic information before proceeding here.

The two major parts of the Logi XOLAP technology are the **Dimension Grid** element, which is the user interface into the data, and its child element, the **Xolap Cube**, which retrieves the data, constructs the data cube, and connects it to the Dimension Grid. The Xolap Cube uses a number of child elements, including one or more datalayers, to retrieve the desired data.

The Dimension Grid is a "super-element", similar to the **Analysis Grid** and the **OLAP Grid** elements. At runtime, users can manipulate the data they see and the way in which it's presented using the Dimension Grid's user interface controls. The report developer has complete control over which controls are available to a user and can present them **selectively** to different users.



Table

Order Year	Employee All Employees	Buchanan	Callahan	Davolio	Dodsworth	Fuller	King	Leverling
	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales
All Years	$1,265,793.06	$68,792.29	$126,862.29	$192,107.60	$77,308.07	$166,537.75	$124,568.23	$202,812.84
2012	$208,083.98	$18,383.92	$22,240.12	$35,764.51	$9,894.52	$21,757.06	$15,232.16	$18,223.96
2013	$617,085.20	$30,716.47	$56,032.62	$93,148.09	$26,310.39	$70,444.14	$60,471.19	$108,026.14
2014	$440,623.88	$19,691.90	$48,589.55	$63,195.00	$41,103.16	$74,336.55	$48,864.88	$76,562.74


Chart
Chart Style: Horizontal Stacked Bar
Click the chart to drill down.
Employee | All Employees
Product | All Product
Seafood
Produce
Meat/Poultry
Grains/Cereals
Dairy Products
Confections
Condiments
Beverages
0k
50k
100k
150k
200k
250k
300k
Total Sales
Davolio
Peacock
Suyama
Callahan
King
Leverling
Buchanan
Dodsworth
Fuller

The example above shows a Dimension Grid configured to show a Data Table and a bar chart. The Data Table behaves like a Crosstab Table, and can pivot data.

This topic discusses the Logi Info Studio **Dimension Grid Wizard** that will create all of the elements necessary to work with Logi XOLAP. If you are interested instead in information about building cubes manually, using individual elements, see Dimension Grid.

The Dimension Grid wizard incorporates several **other wizards** which help you create Xolap Cubes, Xolap Dimensions, and Xolap Measures. Each of these wizards can be runindividually from their context menu or the Element Toolbox when they're selected.

Preparing to Run the Dimension Grid Wizard

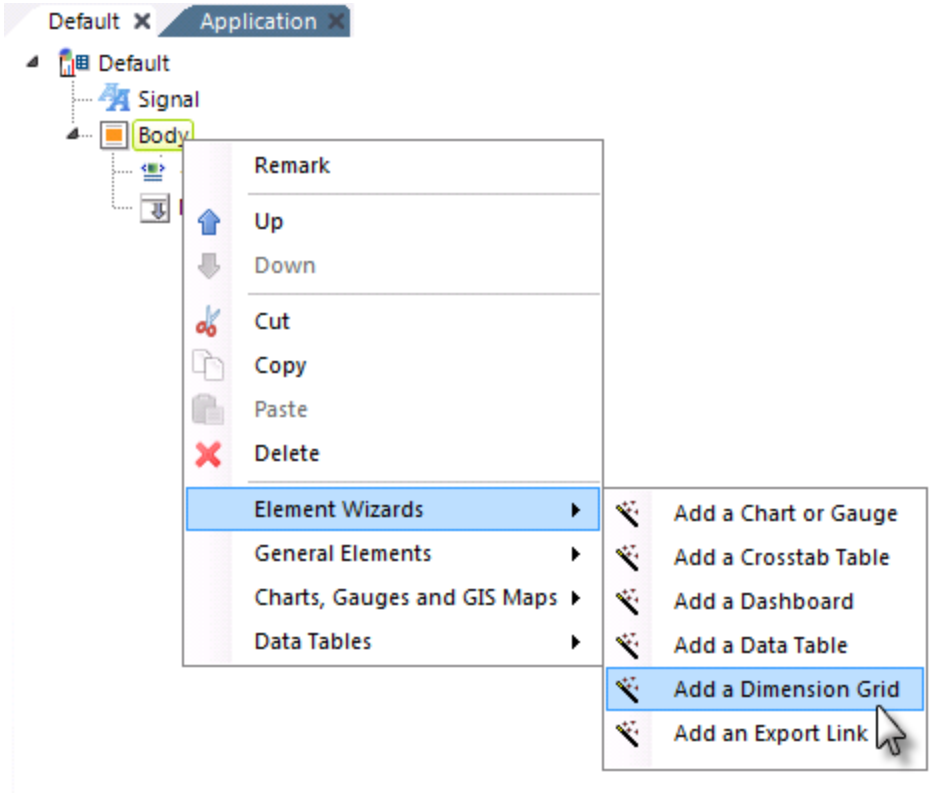
Prior to running the Dimension Grid wizard:

- Become familiar with the data you'll be working with. Identify the data columns that will become *dimensions* and *measures*, what *values* (aggregations) you need, and how you might want to *filter* the data.
- Ensure that you have a valid **Connection** configured in _settings and can connect to your datasource.
- If you're using a SQL data source, you may care to experiment a bit in the Query Builder to develop the query that you'll need.
- You should also keep an eye on the number of result rows your query is likely to return. While Logi XOLAP can process tens of thousands of data rows into an XML cube very quickly, millions of result rows may produce unacceptable delays.

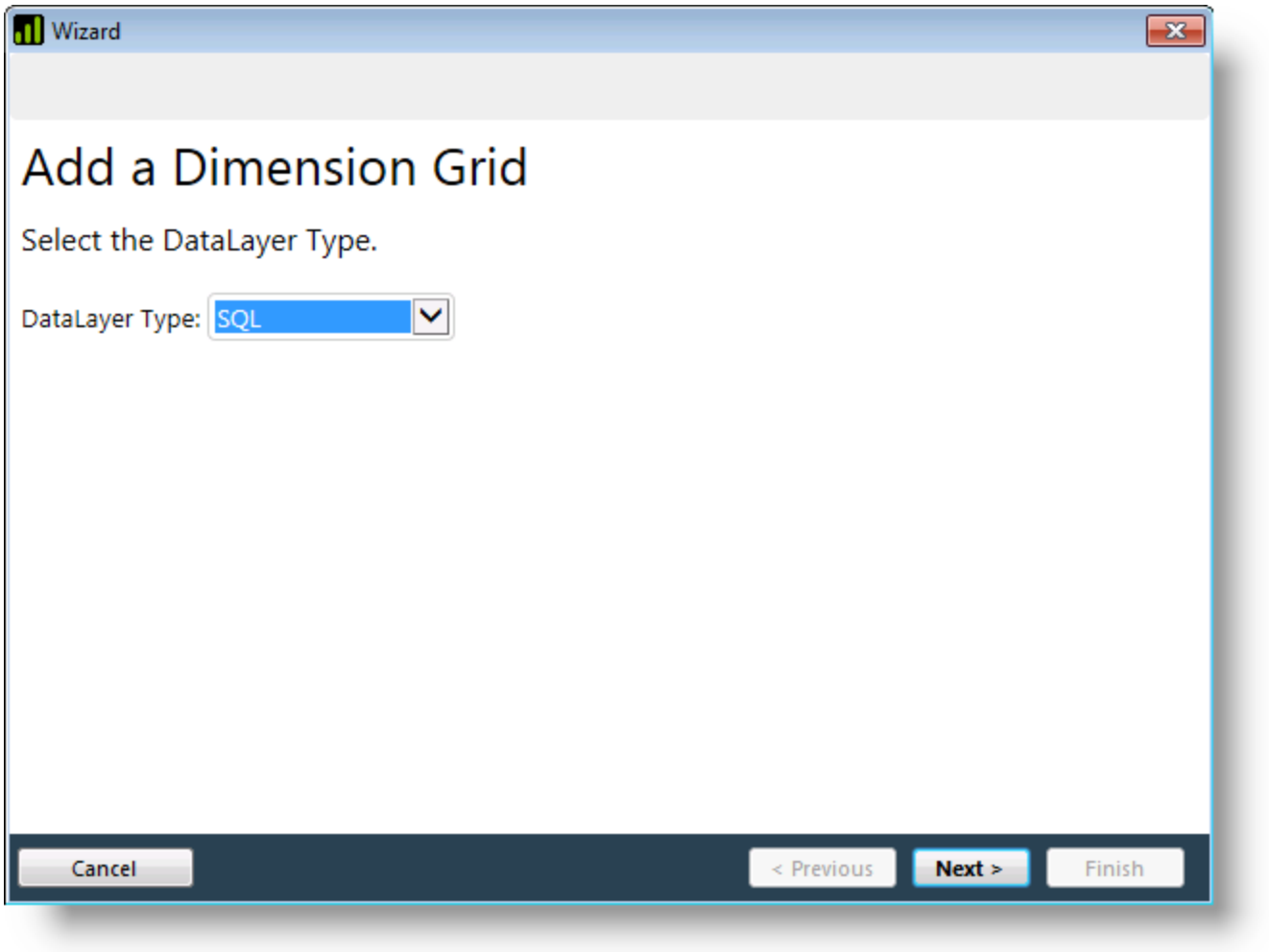
Now, launch Studio and open your report definition.

Using the Dimension Grid Wizard

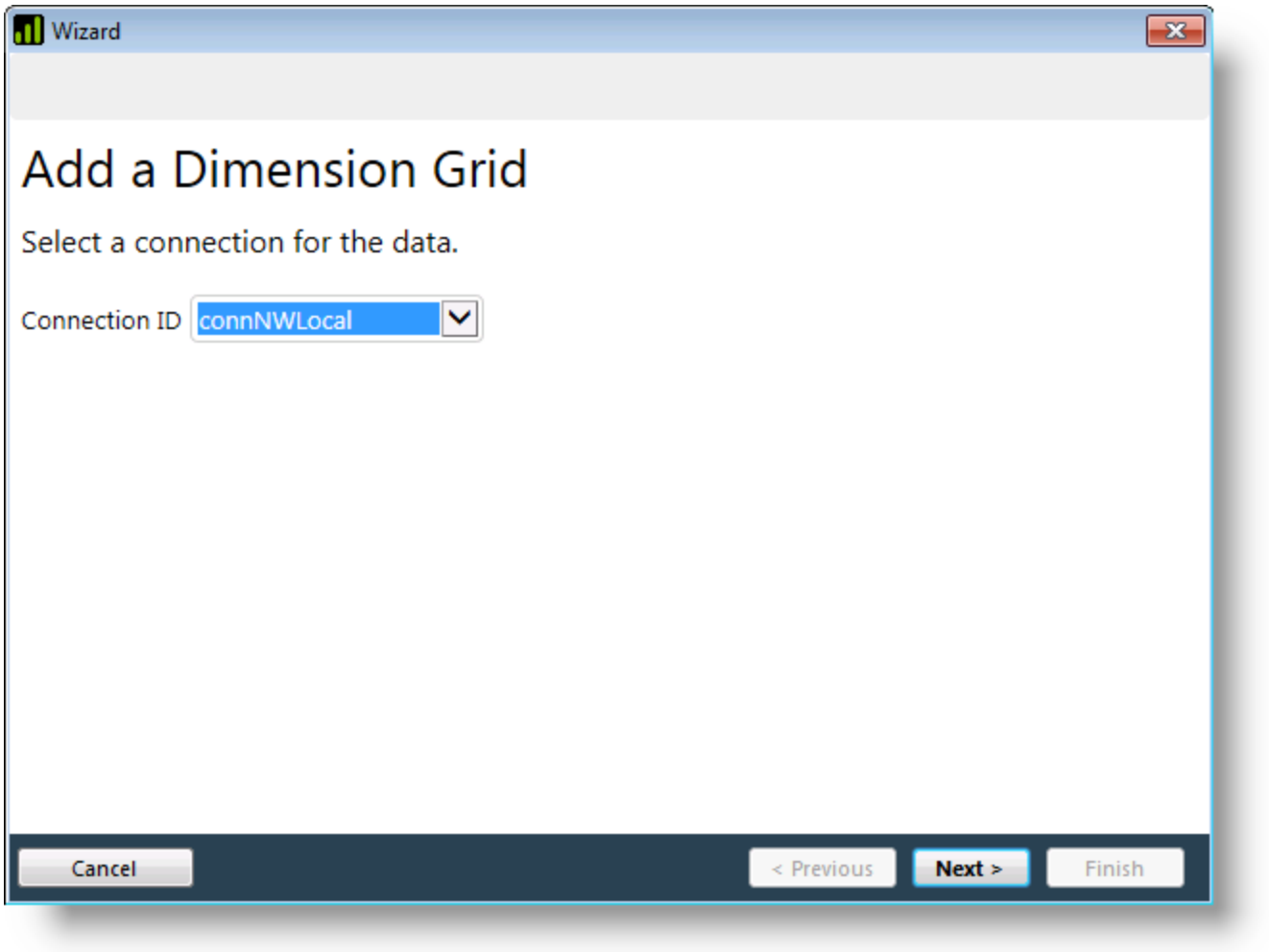
The following examples walk you through using the Dimension Grid wizard.



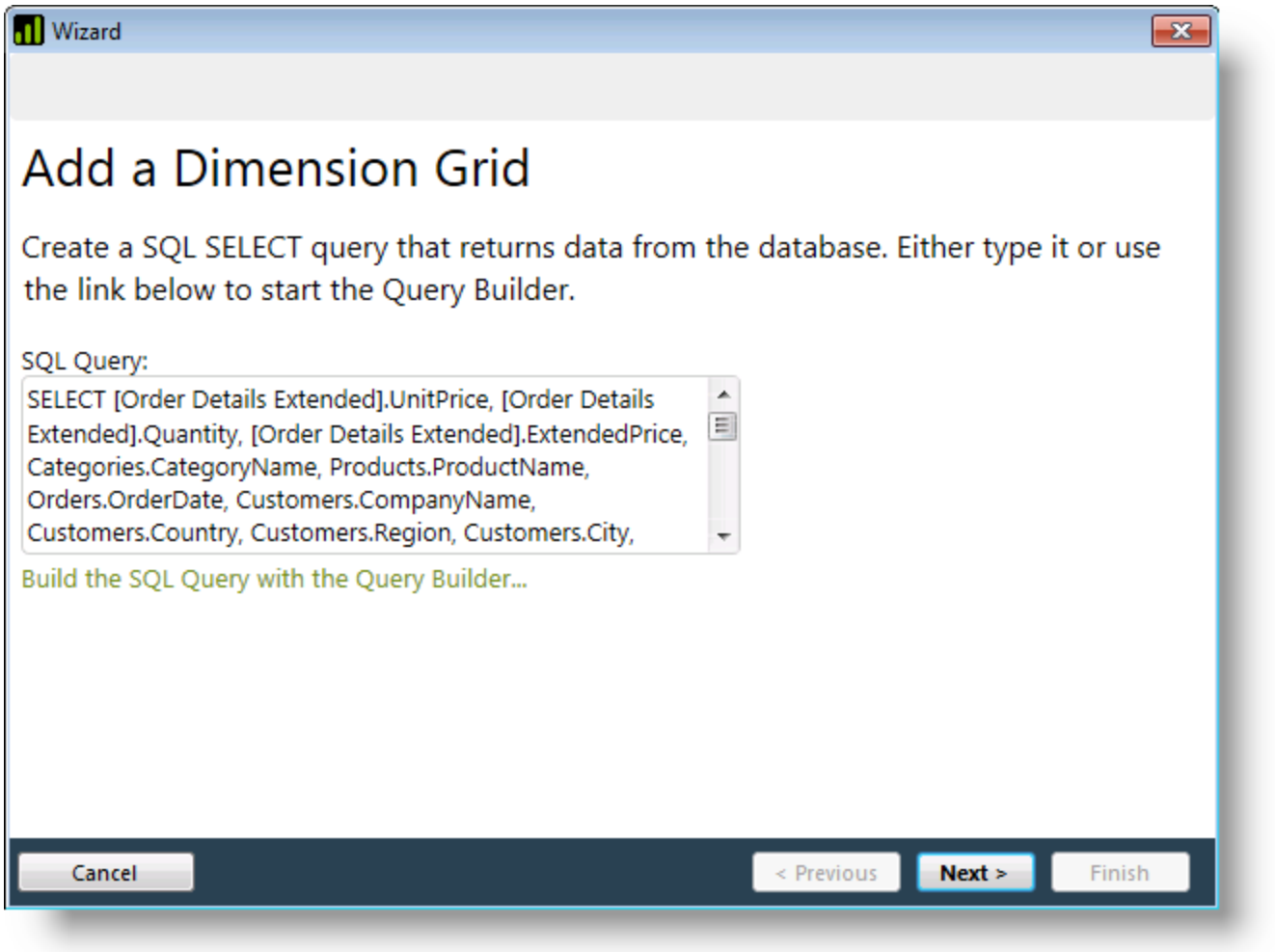
1. Start the wizard by right-clicking the grid's parent and selecting **Element Wizards→Add a Dimension Grid**, as shown above.



2. **Datalayer Type** - Select the appropriate datalayer type for your datasource. Click **Next**.

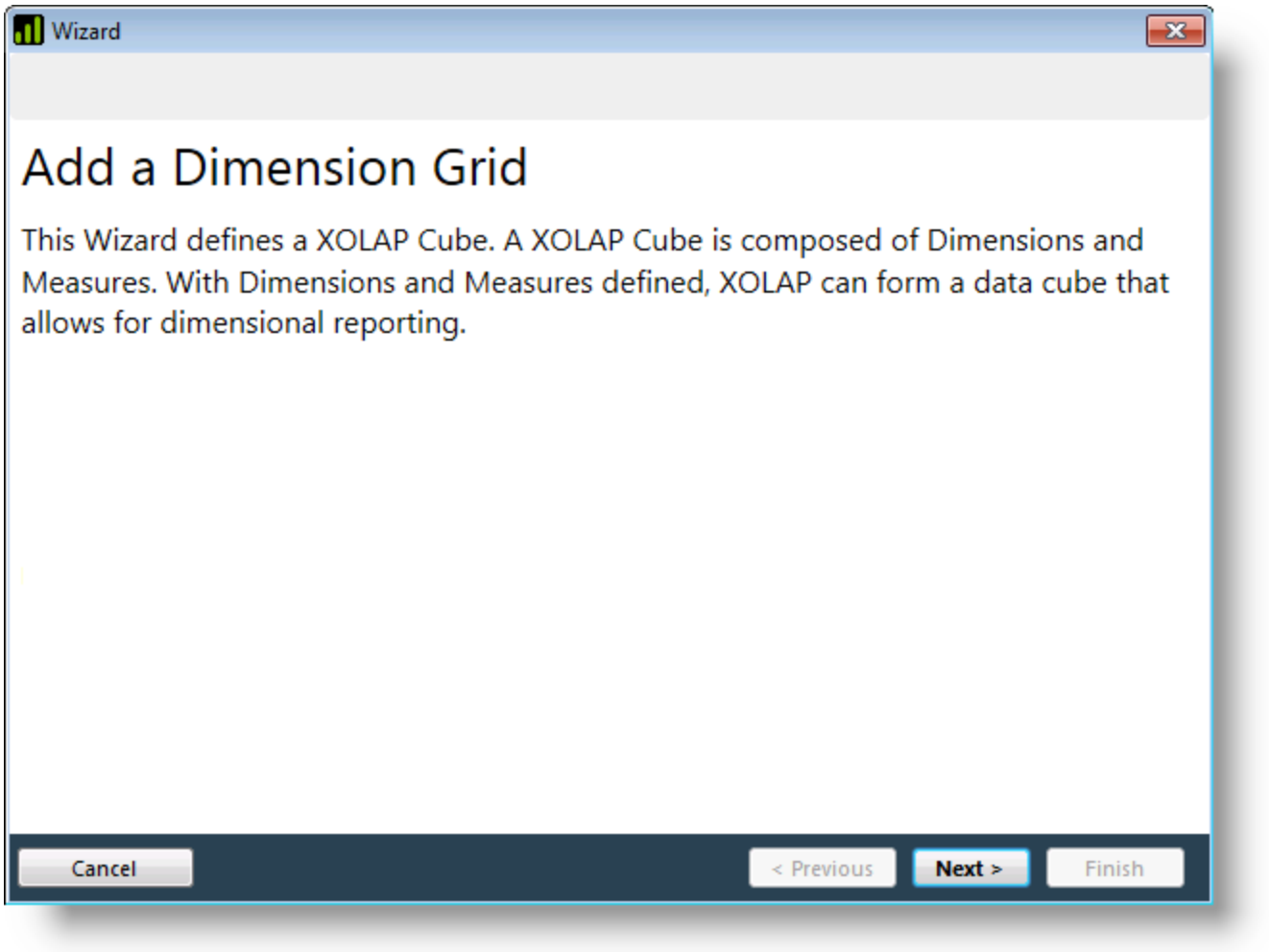


3. **Connection ID** - Select the desired connection from the list of connections defined in the _Settings definition. Click **Next.**

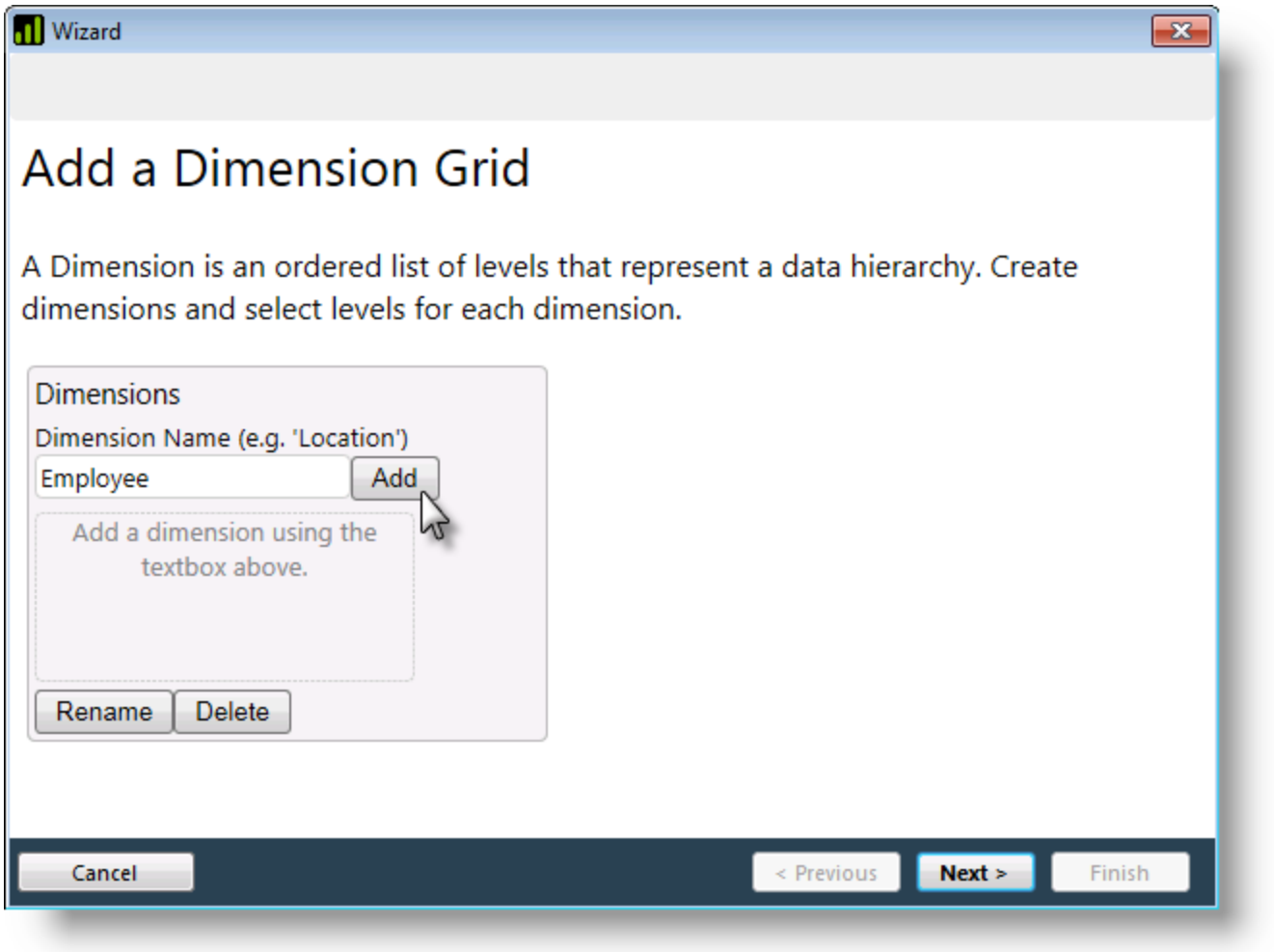


4. **SQL Query** - If you 're using a SQL datasource, enter the query (or use the query wizard to build one) and click **Next**. This example uses the following T-SQL query:

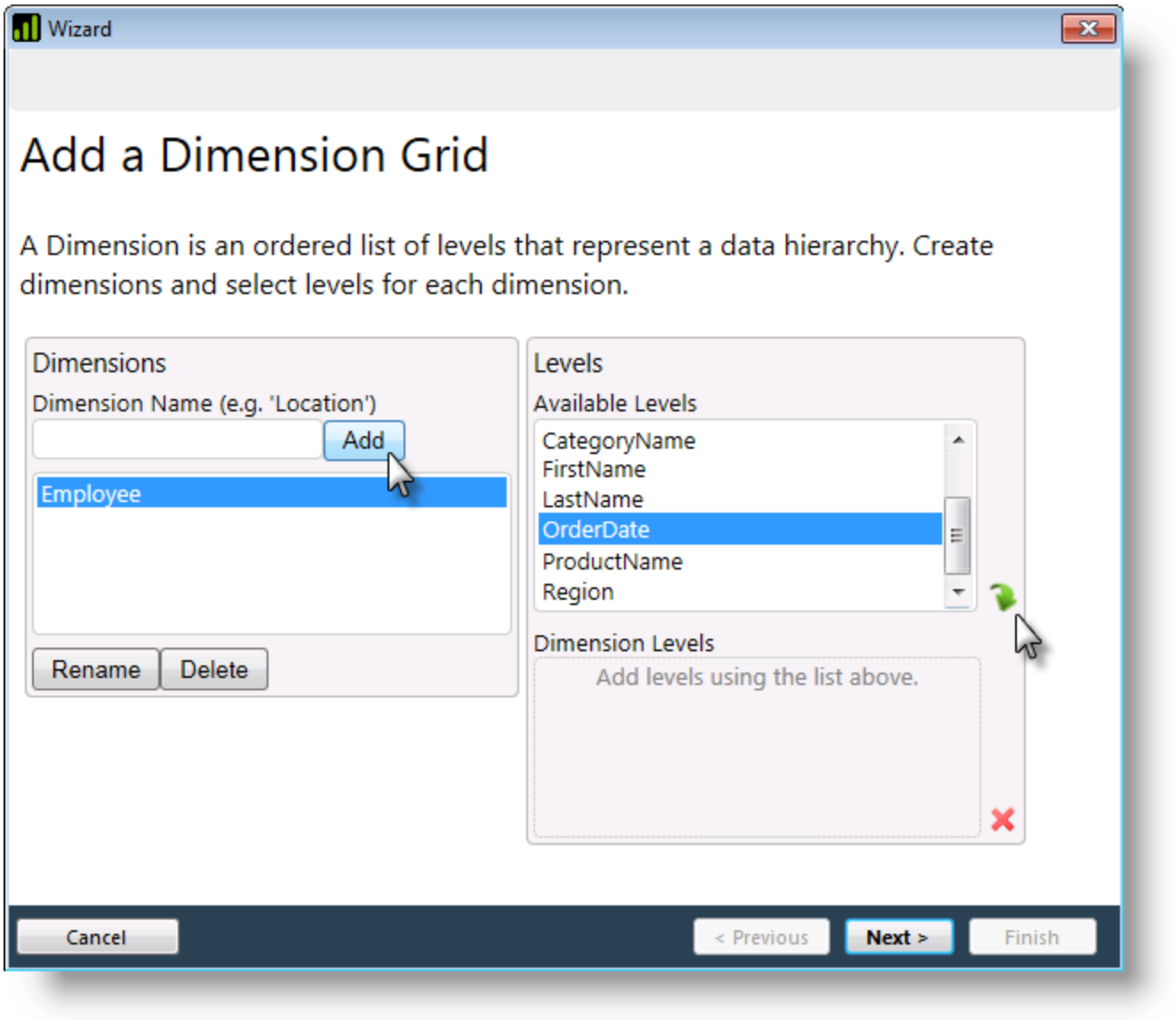
```
SELECT [Order Details Extended].UnitPrice, [Order Details
 Extended].Quantity, [Order Details Extended].ExtendedPrice,
 Categories.CategoryName, Products.ProductName, Orders.OrderDate,
 Customers.CompanyName, Customers.Country, Customers.Region,
 Customers.City, Employees.LastName, Employees.FirstName
            FROM [Order Details Extended]
            INNER JOIN Orders ON [Order Details Extended].OrderID =
 Orders.OrderID
            INNER JOIN Products ON [Order Details Extended].ProductID =
 Products.ProductID
            INNER JOIN Categories ON Categories.CategoryID = Products.CategoryID

            INNER JOIN Customers ON Customers.CustomerID = Orders.CustomerID

            INNER JOIN Employees ON Employees.EmployeeID =
 Orders.EmployeeID
```



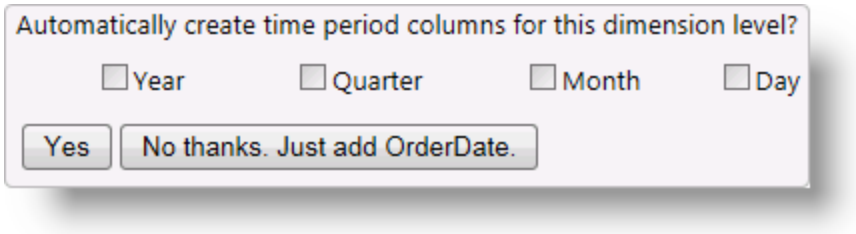
5. **Xolap Cube Wizard** - Click **Next** to start the cube wizard.



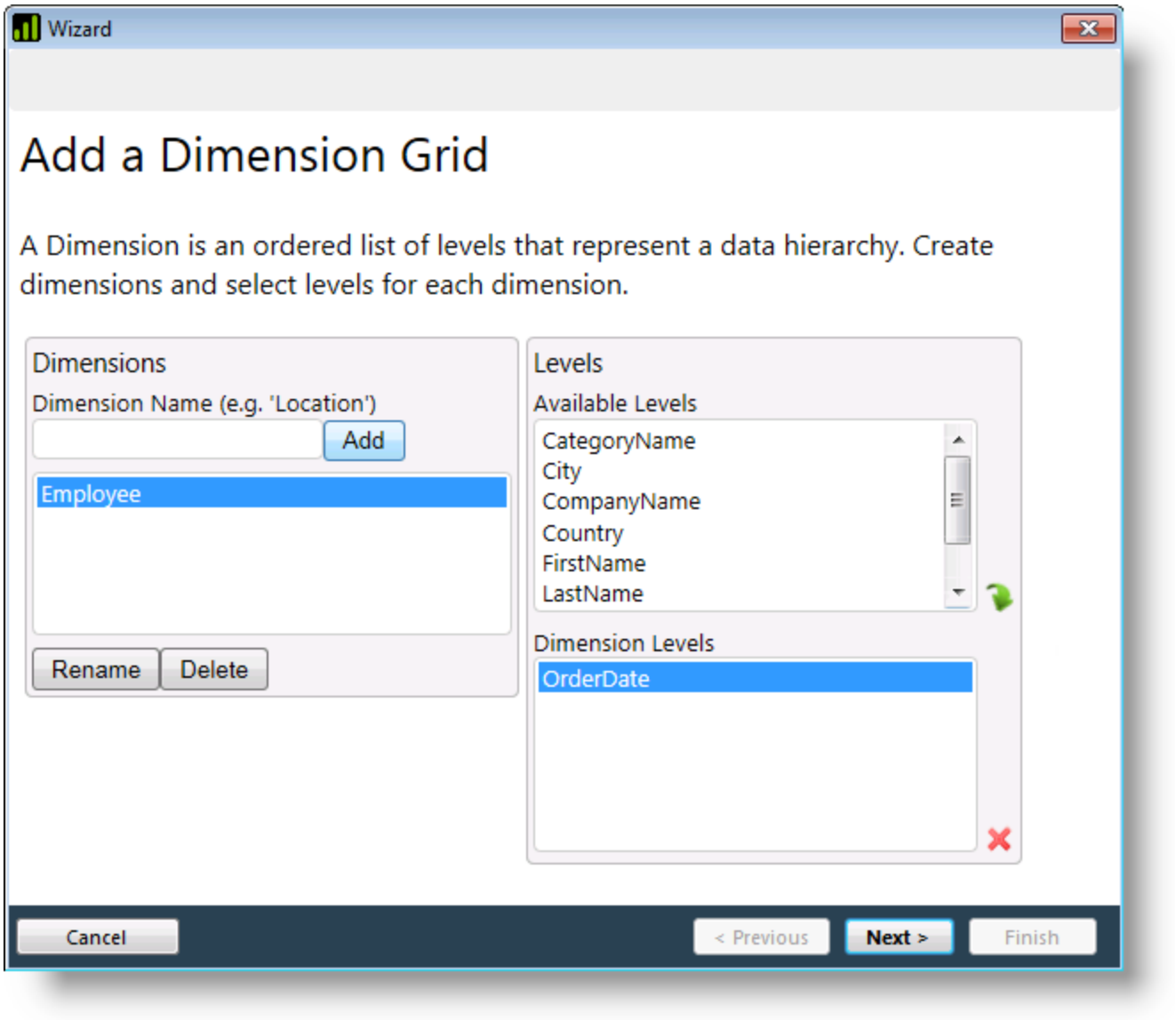
6. **Create Dimensions** - Add dimensions by entering the arbitrary Dimension Name in the text box and clicking **Add**.



7. **Add Levels** - Select a data column for a hierarchical level from the list and click the green arrow icon, as shown above.

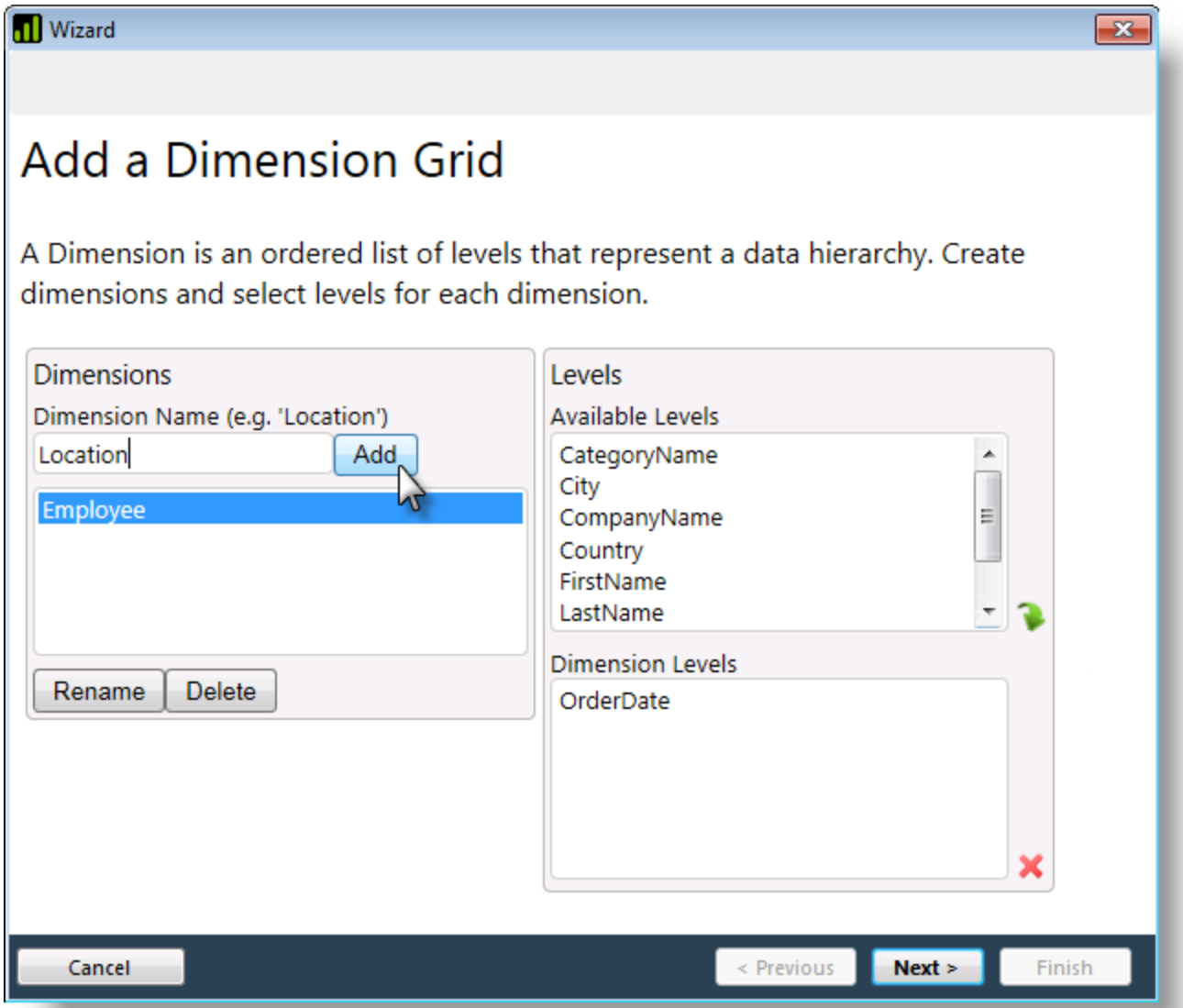


If you select a Date type column, the wizard will prompt you, as shown above, to add time period columns to the data that parse out the Year, Quarter, Month, or Day values, as shown above.

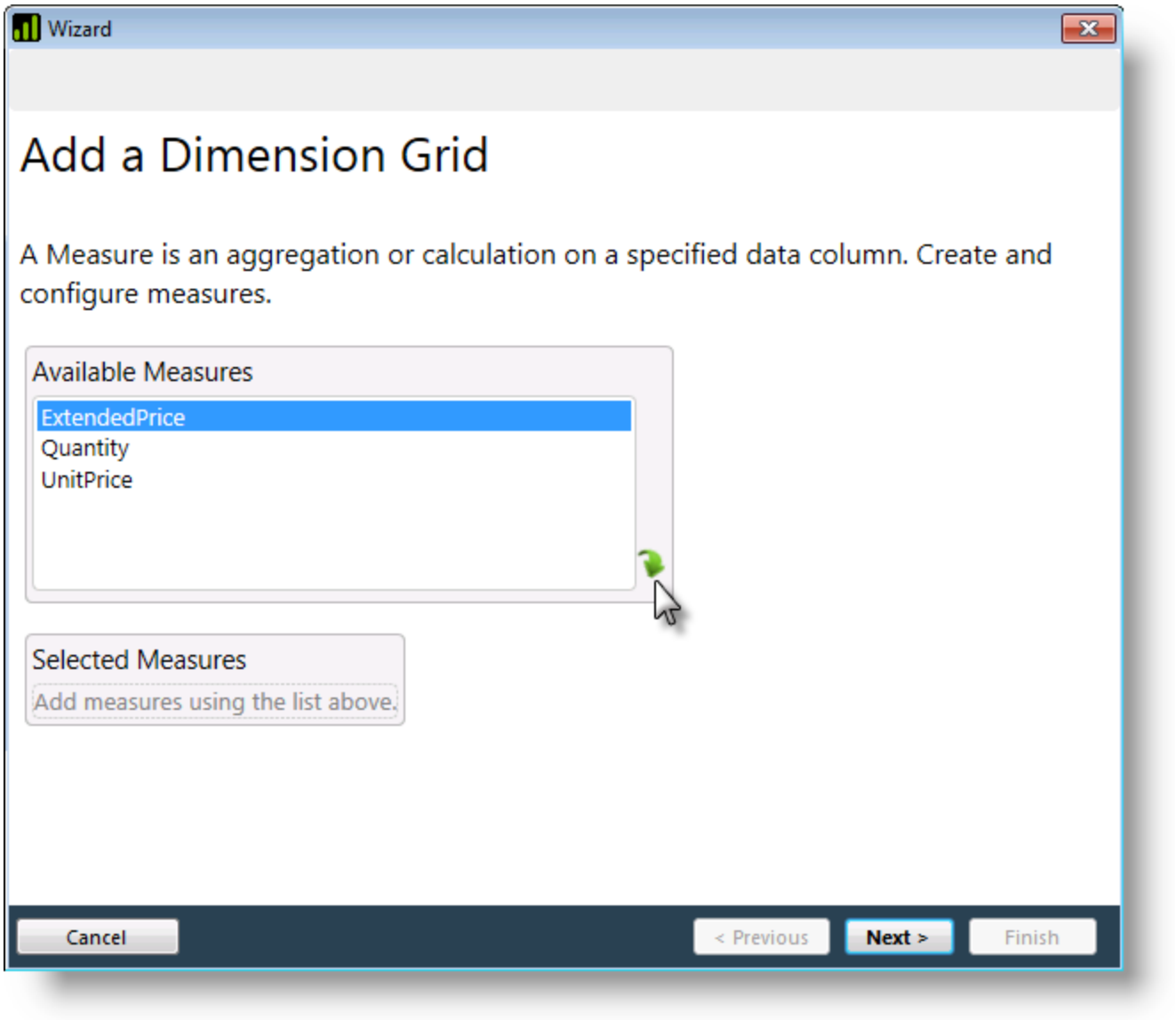


The column you selected will be added to the lower list of Dimension Levels. **Repeat** to add as many levels as desired. To

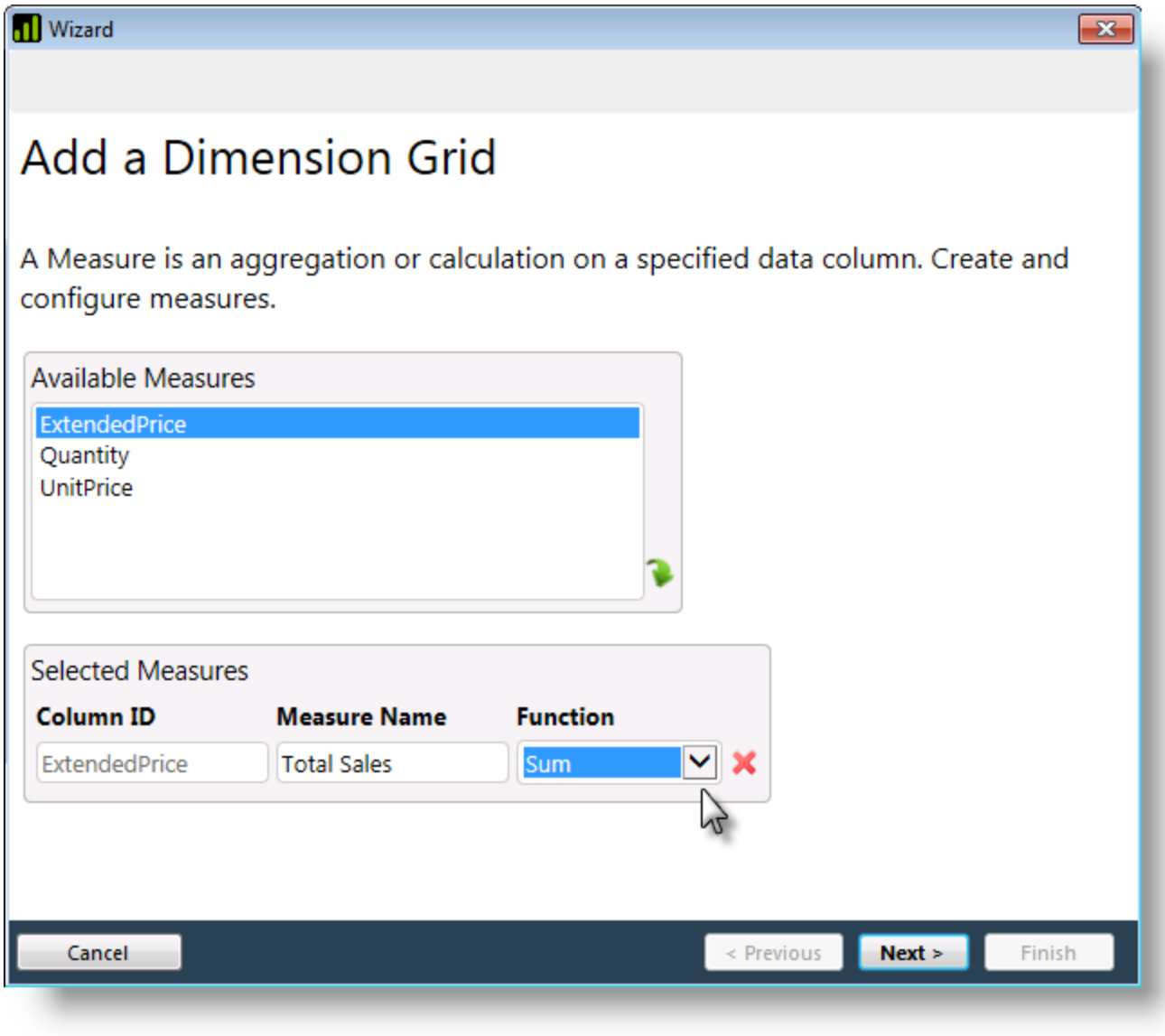
remove a level, select it and click the red X icon. Refer to the end of the topic if you're creating levels using Date-type columns.


Wizard
Add a Dimension Grid
A Dimension is an ordered list of levels that represent a data hierarchy. Create dimensions and select levels for each dimension.
Dimensions
Dimension Name (e.g. 'Location')
Location
Add
Employee
Rename
Delete
Levels
Available Levels
CategoryName
City
CompanyName
Country
FirstName
LastName
Dimension Levels
OrderDate
Cancel
< Previous
Next >
Finish

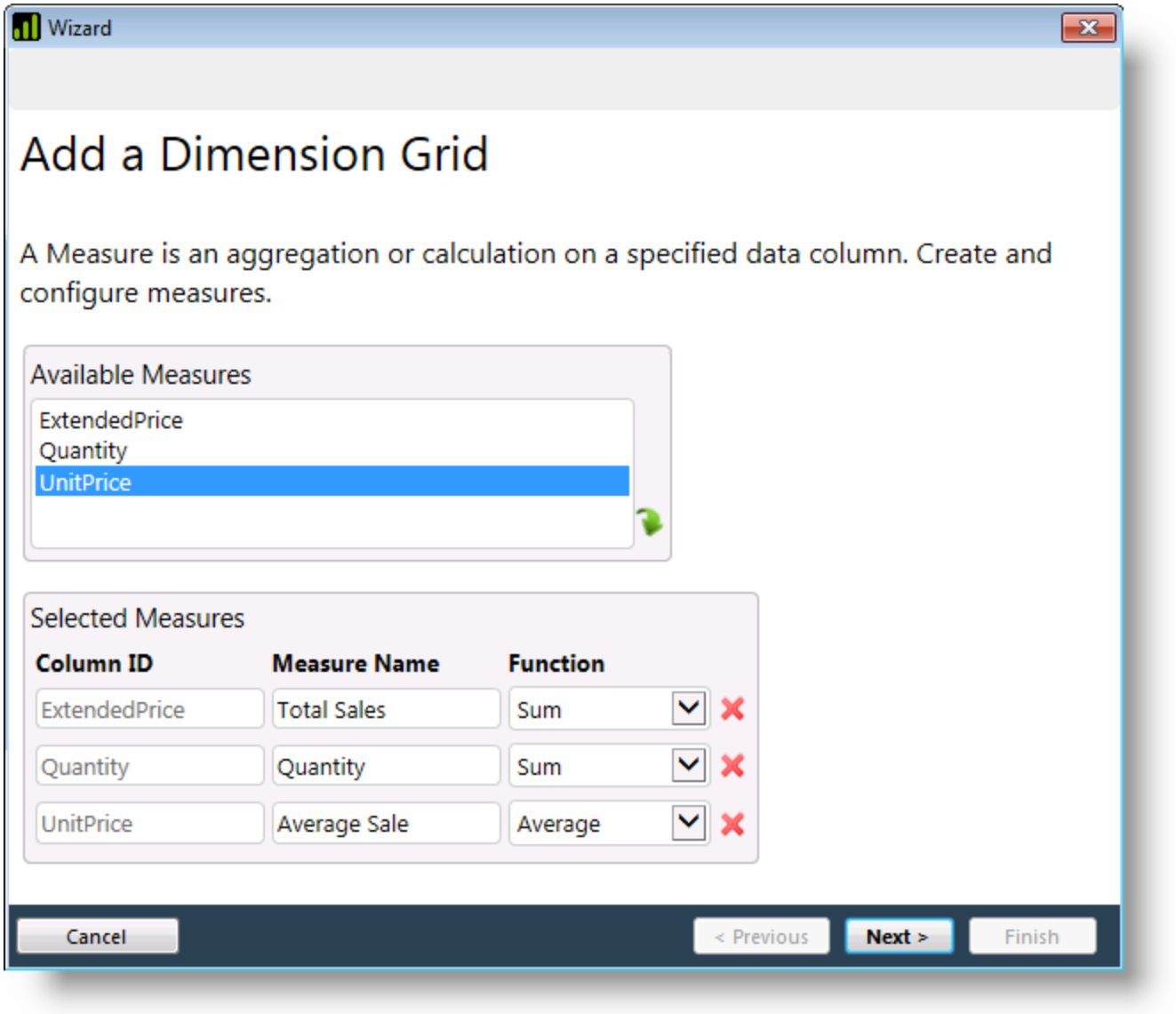
8. **Add More Dimensions and Levels** - Repeat the process of adding dimensions and levels until all dimensions have been configured. Use the Rename and Delete buttons to affect dimensions you've already configured. When all dimensions and levels have been configured, click **Next**.



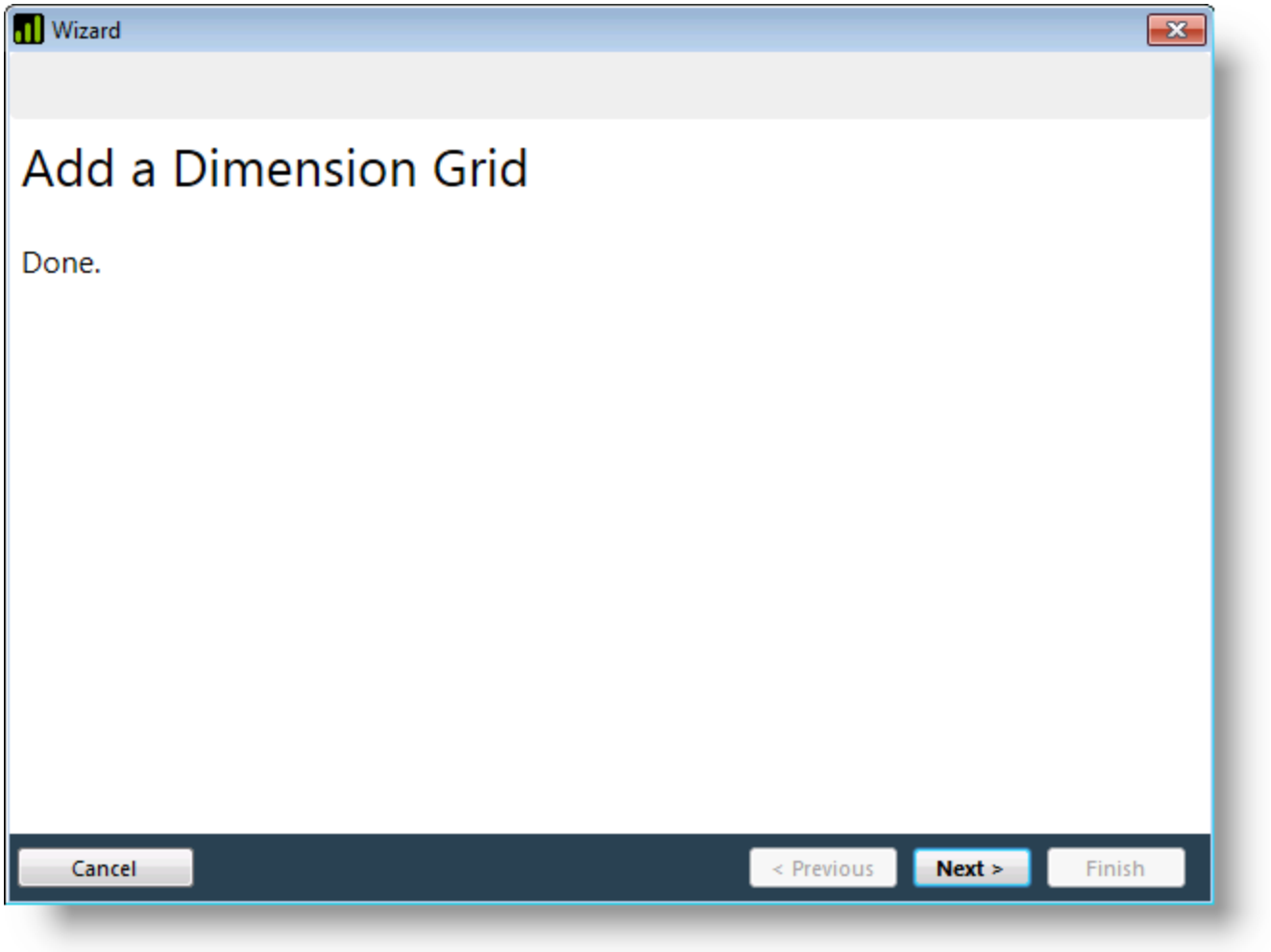
9. **Create Measures** - Select a data column to be aggregated from the list and click the green arrow icon, as shown above.


Wizard
Add a Dimension Grid
A Measure is an aggregation or calculation on a specified data column. Create and configure measures.
Available Measures
ExtendedPrice
Quantity
UnitPrice
Selected Measures
Column ID
Measure Name
Function
ExtendedPrice
Total Sales
Sum
Cancel
< Previous
Next >
Finish

The selected measure will appear below. Enter an arbitrary **Measure Name** and select the aggregation or summarization function to be applied to it. To remove a measure, click the adjacent red X icon.

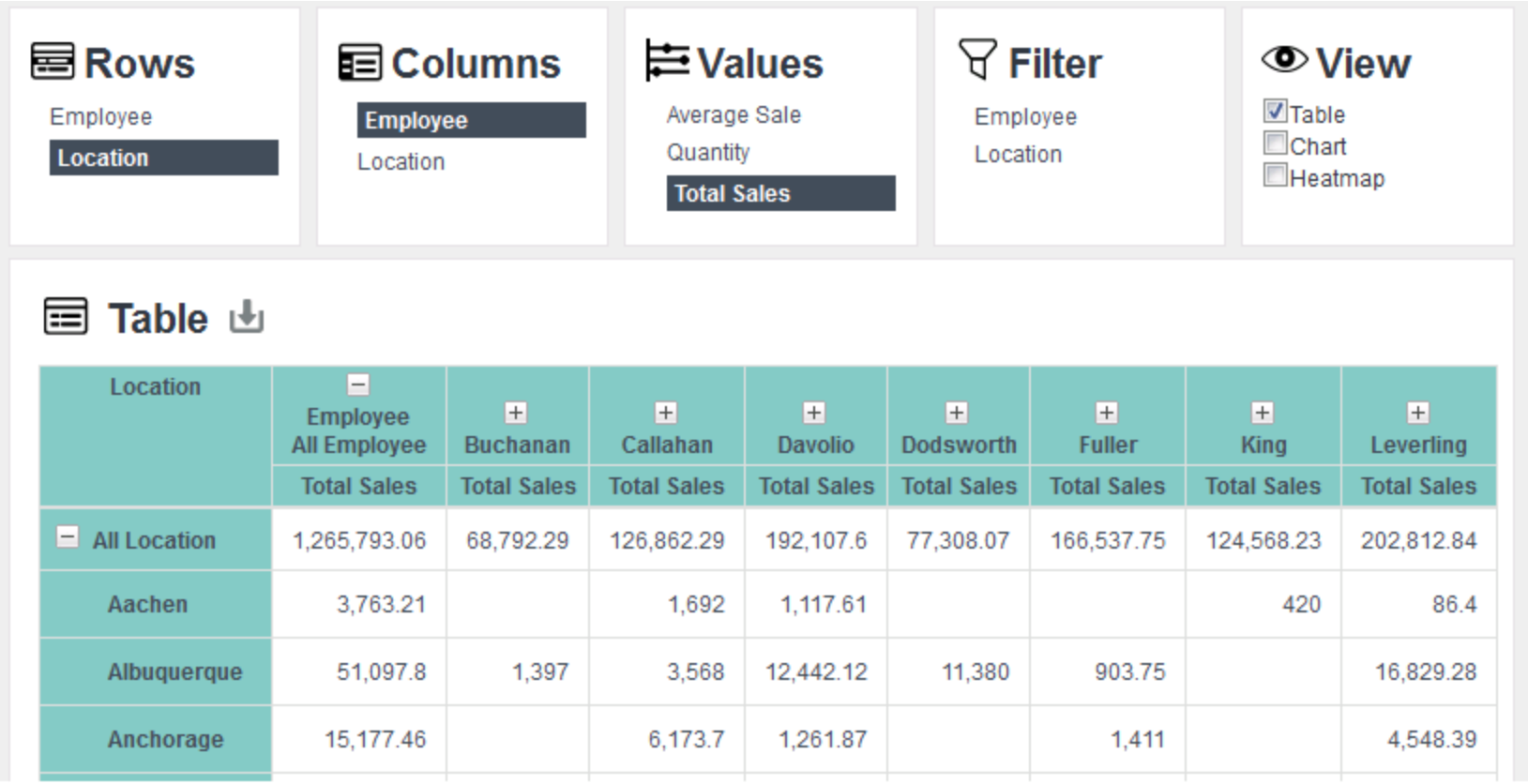


10. **Add More Measures** - Repeat the process until all measures have been configured, then click **Next**.



13. **Done** - Click **Finish** to close the wizard.

The wizard has added all the appropriate elements to your report definition. Save the definition and run it.



Location	Employee All Employee	Buchanan	Callahan	Davolio	Dodsworth	Fuller	King	Leverling
	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales
All Location	1,265,793.06	68,792.29	126,862.29	192,107.6	77,308.07	166,537.75	124,568.23	202,812.84
Aachen	3,763.21		1,692	1,117.61			420	86.4
Albuquerque	51,097.8	1,397	3,568	12,442.12	11,380	903.75		16,829.28
Anchorage	15,177.46		6,173.7	1,261.87		1,411		4,548.39

When it appears, select dimensions, measures, and values from the Dimension Grid and view your data.

You may need to manually set the Format of Date type columns.

Sorting Dimensions by Dates

Dates in dimension values are sorted as strings, which can be confusing. To make a date-type dimension appear in sorted order across the Dimension Grid, ensure that its data is returned into the datalayer as a DateTime value in ISO 8601 format (yyyy-mm-dd). To sort the dimension by Month and display the month number, you will need to **manually** setits Xolap Level element's **Format** attribute to *MM* once the wizard is finished. To sort it by Year and display the year number instead, set this attribute to *yyyy*.

Adding Tooltips

"Tooltips," those little balloons that can pop-up when you hover your mouse over an image, link, button, and other elements, provide a fast and easy method of providing additional explanation. They're generally a kind of on-the-spot "mini online help," though they can be much more. This topic describes adding tooltips to your Logi application, and provides links to tooltip elements.

The following topics discuss the multiple ways in which you can add tooltips to your Logi applications:

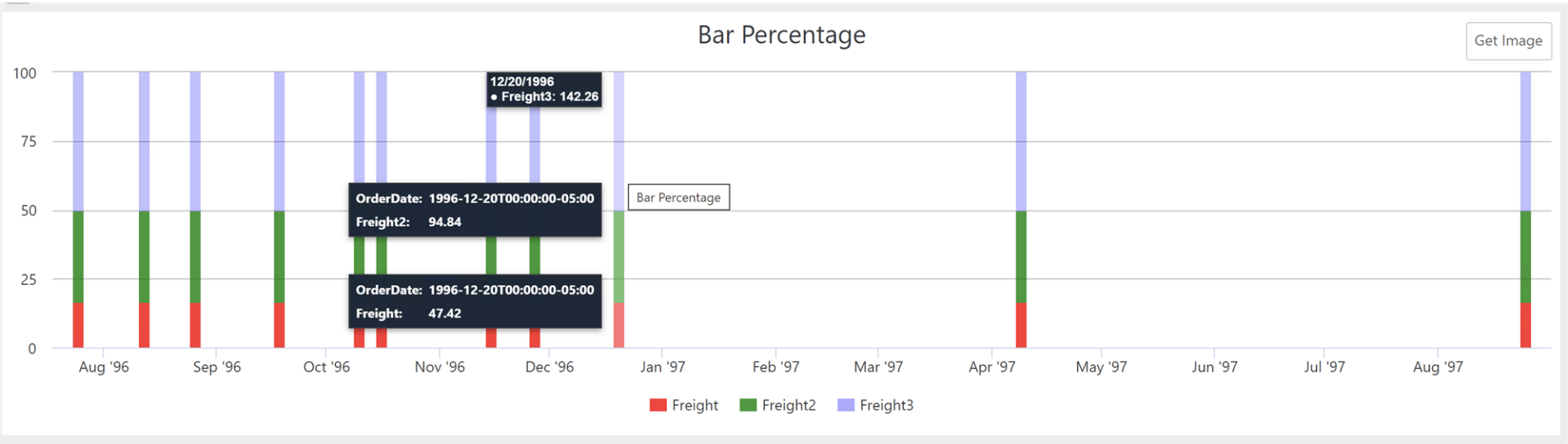
- Tooltip Attribute
- Quicktip Element
- Tooltip Panel Element

About Your Tooltip Options

Logi Info offers several methods for adding tooltips to your Logi application and each is discussed in one of the topics mentioned above, which are presented in order of increasing complexity and functionality. Adding tooltips is an excellent idea in situations where a little more explanation would be useful (but it's not worth going to another web page), or there's underlying data that would clutter up a report if all of it was shown at once.

Some Logi super-elements, such as the Analysis Chart and Analysis Grid, automatically incorporate tooltips into their charts, so that the underlying data value is displayed when the mouse is hovered over different points on the charts.

Charts with multiple series or aggregations are encouraged to use the ChartCanvas element's attribute, Tooltip Split. This attribute allows you to split a chart's tooltip into one label per series, rather than per segment. The default value for this attribute is False. Once enabled, hovering over a data point exposes all the values for the series:



This method is recommended over shared tooltips for charts with multiple line series, and as such, takes precedence over `tooltip.shared`.

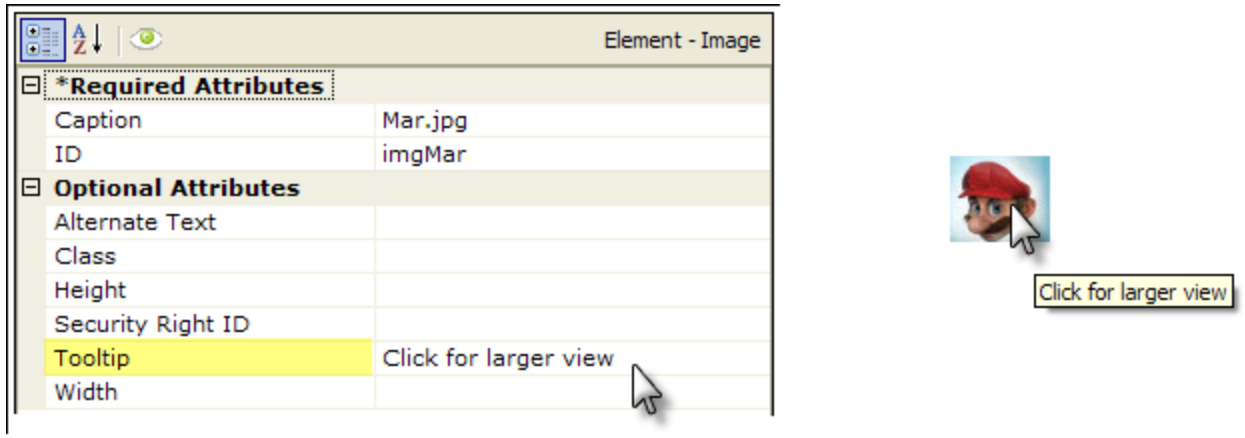
v23.1 Another attribute, Tooltip Outside, can be incorporated to optimize space when working with smaller charts. Setting this attribute to True allows the tooltip to display outside the chart's area.

Given how easy it is to add tooltips, you may not want to consider your application really *finished* until you do.

Tooltip Attribute

Many Logi elements have a **Tooltip** attribute, and using it is the easiest way to add a tooltip to your application.

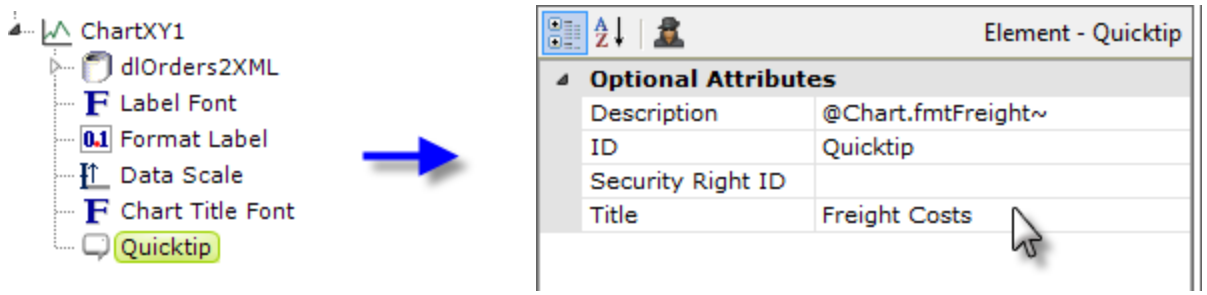
In the example below, we see a very common situation: a small "thumbnail" image can be clicked to view a larger image, and this is communicated to the user via a tooltip. The example shows the attributes for the small image on the left, and the resulting tool-tip text that's automatically displayed when the mouse hovers over the image, on the right. Tokens can be used in the Tooltip attribute.



This tooltip is limited to text only, can't be controlled dynamically, and provides no easy way to apply style to it in order to change the font size, color, etc.

Quicktip Element

The **Quicktip** element is almost as easy to use as the Tooltip attribute, but provides a tooltip with a little different look. It can used beneath static Chart, Gauge, Division, and Heatmap Group Column elements. The Quicktip element automatically provides tooltips for the charts found within the Analysis Chart and Analysis Grid super-elements, as well as all Chart Canvas charts.



In the example above, we see the **Quicktip** element beneath a **Chart.XY** element, and its attributes are also shown. The resulting tooltip that appears when the mouse *hovers* over the chart line looks like this:

Visually, this is more complex than basic tooltips. The pop-up panel has space for both a title and a description and includes a small, angular protrusion that points to the cursor, and its color and fonts can be controlled by assigning a theme, in this case Professional Green, to the application.

Tokens can be used in both the **Title** and **Description** attributes. The Quicktip element has a **Security Right ID** attribute, so its appearance can be dynamic, based on security roles. Because the Quicktip is displayed based on the *onMouseOver* DHTML event, you can't put anything inside the Quicktip itself that a user could click, such as a link.

If you're using Quicktips with a chart that includes **Extra Grid Layers** and the primary chart datalayer is reduced to zero rows at runtime, for example, because of filtering, then Info will not display the Quicktips, not even for the Extra Grid Layer plots.

Quicktips should *not* be used inside Data Tables.

The Quicktip is limited to text only and, as the text gets longer, the panel will get wider and wider until, at some point, the text will automatically wrap to another line. But you can't *force* it to wrap and the point at which it will wrap is based on font family and size (in the example shown above, it will wrap at 90 characters, so it can get pretty wide before it wraps).

However, you can deliberately add additional lines by adding child **Quicktip Row** elements beneath the Quicktip, one for each additional line of text you want to include.

You can also configure the tooltip to display values for any available column in the datalayer. To do so, enter a token representing the data column in the Value attribute of the Quicktip Row element.

 To use this feature with DataLayer.ActiveSQL, please make sure the keep Grouped Rows attribute of the SqlGroup element is set to *False*.

The Quicktip Row element has been made context-sensitive with the addition of a **Condition** attribute.

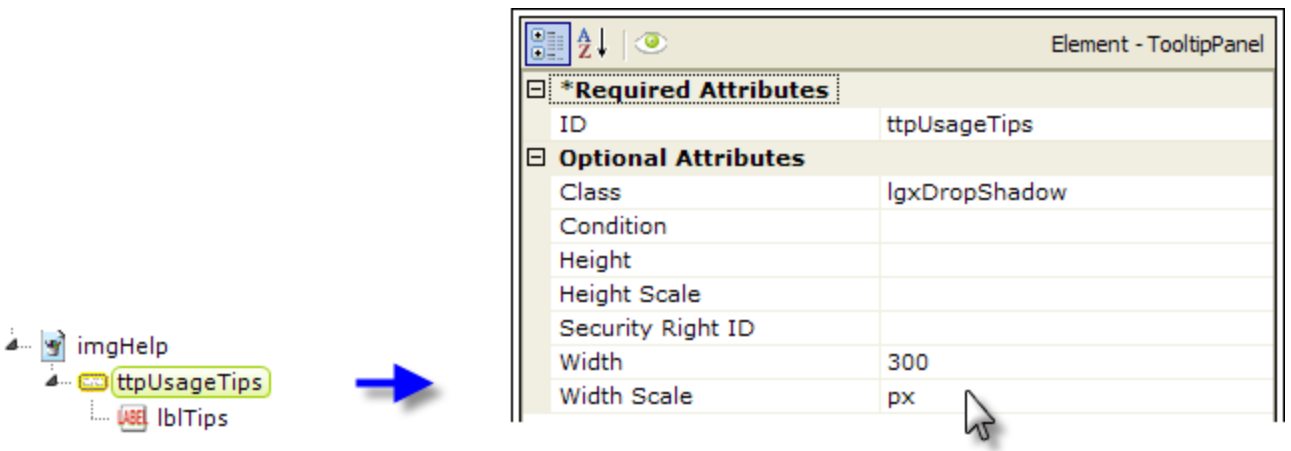
Tooltip Panel Element

The **Tooltip Panel** element is a flexible option for including more complex tooltips. This topic describes how to add a panel element.

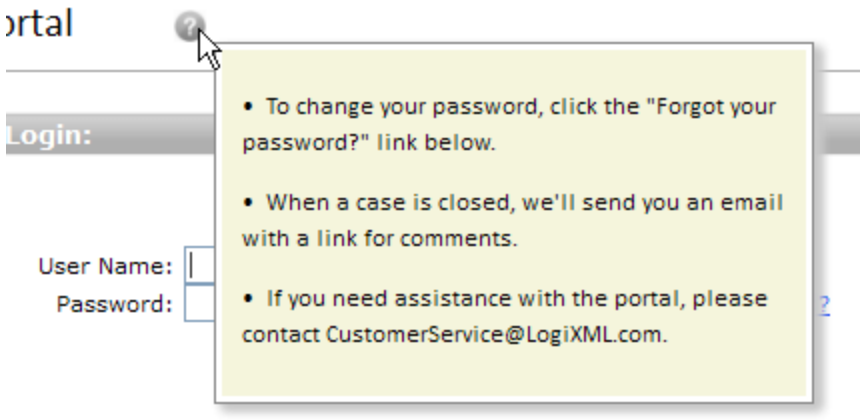
You can add the Tooltip Panel element as a child beneath these elements:

- Button
- Column Cell
- Data Table Column
- Division
- Image
- Label

As an element, the Tooltip Panel can have a variety of child elements, allowing you to get more creative with what you include in the tooltip.

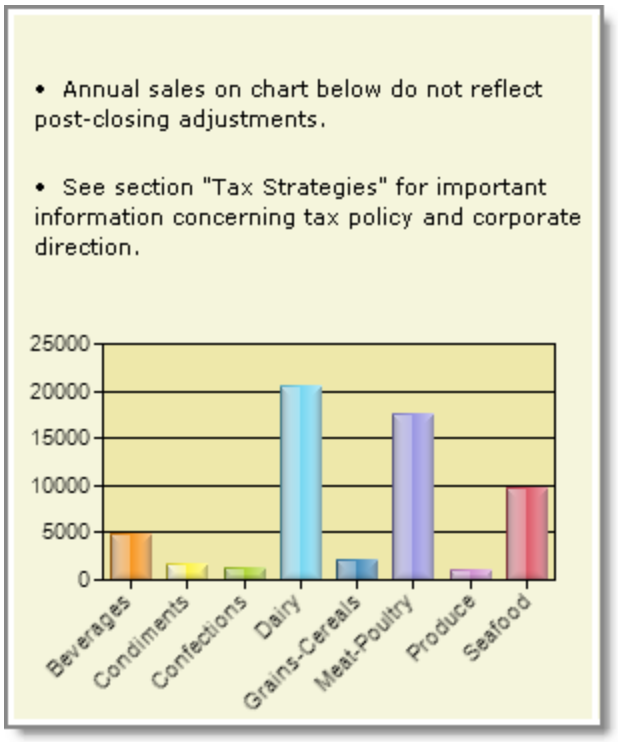


The example above shows an **Image** element with a **Tooltip Panel** element placed beneath it, with a child **Label** element. The Tooltip Panel has a full set of attributes that can be used to customize the tooltip. Let's see what it can look like:



In the example shown above, we see a Tooltip Panel that appears when the mouse cursor hovers over the "help" icon. Style has been used to color the panel background and give it a drop shadow, and the child Label element has been set to HTML format so that the help text will be displayed as an unordered list with bullets. Image and Label elements now include "HTML Attribute Params", enabling you to apply your application to work with other frameworks or libraries easier. Depending on the element you add the HTML Attribute Params to, there are additional parameters for you to specify. The IMG element includes "alt", "style", and "title" parameters, while the Label element allows you to include "Id", "type", and "value". If an attribute was set in both Element and HTML attribute params, the one set in the HTML attribute params will be ignored.

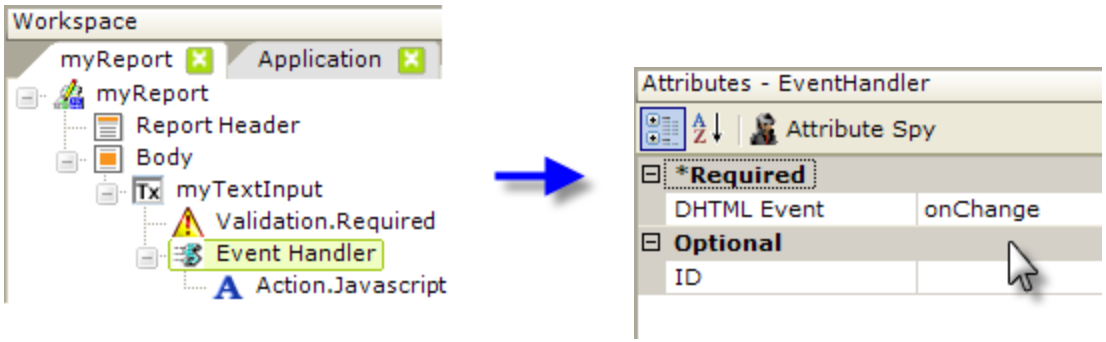
The Tooltip Panel element has **Condition** and **Security Right ID** attributes, so its appearance can be controlled dynamically. For example, separate sets of tooltips for different users are possible.



A variety of elements, including Charts, Gauges, Images, SubReports, and others, can be included as children of the Tooltip Panel, as shown in the example above, giving you the opportunity to make your tooltip contain more than just text.

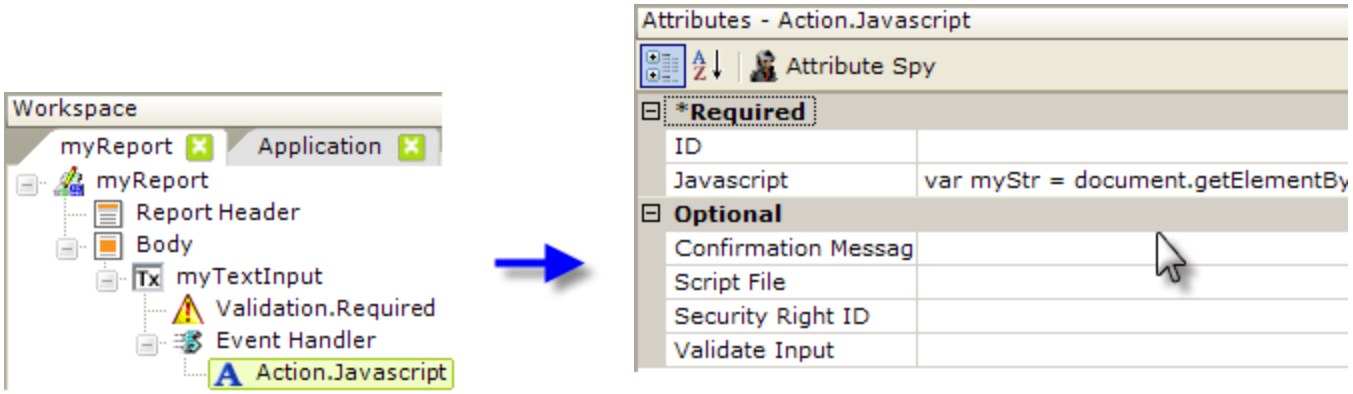
Validate User Input with JavaScript

Have you ever wanted to *limit* the amount of text users can enter into an **Input Text Area** element? Or to modify double-quotes or special symbols entered into an **Input Text** element before processing because they foul up your SQL query? Here's how you can use JavaScript to control user input.



The key is using an **Event Handler** element with your input element, along with an **Action.Javascript** element (or **Action.Link** and **Target.Link** elements), as shown above. Set the Event Handler's DHTML Event attribute to *onChange*, which occurs when the user submits the data they entered.

This technique will not work if you also assign a **Change Flag Element ID** attribute for your input text elements - both use the same event.



The JavaScript used to validate the input is entered in the **Javascript** attribute (or the **URL** attribute of the Target.Link element), as shown above. Don't forget that you can get more editing space if you double-click the attribute name to open the Attribute Zoom window. Here's some JavaScript that will remove double-quotes and the % symbol, convert single-quotes to two single-quotes (to keep SQL happy), and change the symbol "@" to the word "token". Preface this with "`javascript:`"if using Target.Link.

```
var myStr = document.getElementById('myTextInput').value;
myStr = myStr.replace(/"/g, '' );
myStr = myStr.replace(/%/g, '');
myStr = myStr.replace(/'/g, "''");
if ( myStr.indexOf("@") != -1 ) myStr = "token";
document.forms[0].myTextInput.value = myStr;
return true;
```

This JavaScript example limits the maximum number of characters that can be entered into an **Input Text Area** element and displays an error message (and stops further processing) if the limit is exceeded:

```
var myStr = document.getElementById('myInputTextArea').value;
if ( myStr.length > 4000 ) alert('Your message is too long: maximum length is 4,000 characters.');
if ( myStr.length > 4000 ) return false;
return true;
```

Optional: You can also use other **Validation** elements with an Event Handler element. Need help with JavaScript syntax? Check out this JavaScript Referencesite.

Send Cell Phone SMS Messages

Logi Info includes the ability to send notification messages to **mobile devices**. This can be very useful for those who need to be contacted when various events occur in Logi applications.

The following topics show developers how to build this capability into their Logi apps:

- Connecting to an SMTP Server
- Sending Email to the Phone

About Sending Email to Cell Phones

SMS **text messaging** is predominantly used to send messages from one mobile device to another. However, the technologuy can also be used for sending **email** to a mobile device. Logi Info developers can create a task in a Process definition that sends an SMS message to a mobile device, thus providing an automated notification capability within Logi reports and applications. Most mobile carriers provide **SMS gateways** which will accept email messages and deliver them to their customers' mobile devices as SMS text messages. However, the key requirement necessary in sending such an email message is knowing what **carrier** the recipient's phone uses. In this case, unlike cell phone-to-cell phone text messaging, simply providing a cell phone number is *not* sufficient. Each carrier determines what the "email address" of the receiving phone is going to be and so you must know the carrier's "SMS domain". Typically, cell phone email addresses consist of the cell phone number followed by the SMS domain. For example, 7035551212@txt.att.net. The following U.S. carrier SMS domain names are provided for your convenience:

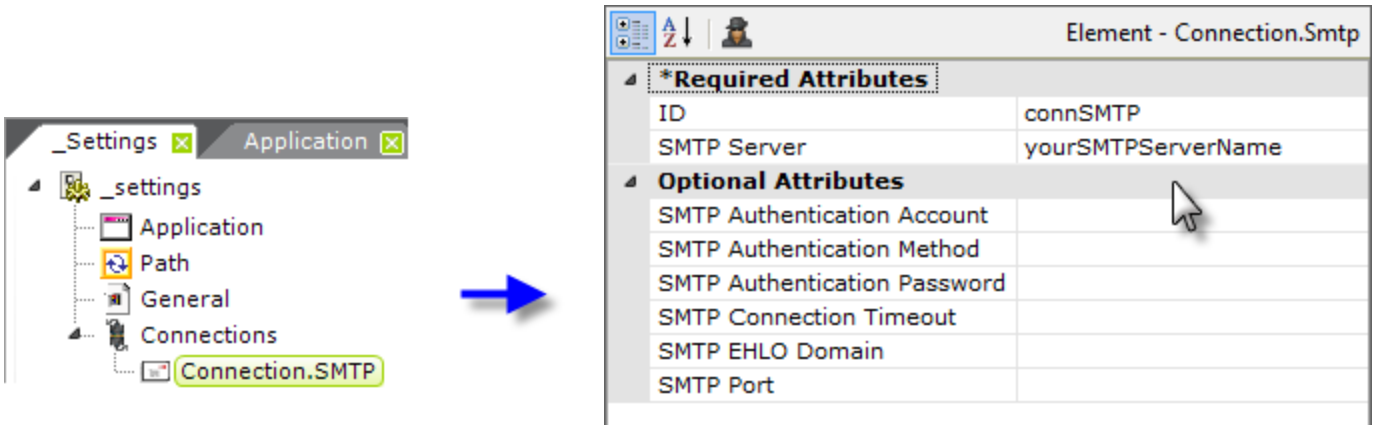
Carrier	Domain Name
Alltel	@message.alltel.com

Carrier	Domain Name
AT&T Mobility	@txt.att.net
Boost Mobile	@myboostmobile.com
Qwest Wireless	@qwestmp.com
Sprint	@messaging.sprintpcs.com
T-Mobile	@tmomail.net
Verizon Wireless	@vtext.com
Virgin Mobile USA	@vmobl.com

Some foreign carriers also provide this service. The **Connection.SMTP** supports International Carrier domains and is no longer restricted to *.net* or *.com* domains. The domain names in the examples above are subject to change, so you may need to contact your carrier or search the Internet for currentdomain names. Generally, this service is provided without charge in the U.S. but fees may apply elsewhere. If in doubt, check with your carrier.

Connecting to an SMTP Server

The basic connection for sending an email is managed by the Connection.SMTP element, and requires that you have an SMTP server set up. Many web servers include an SMTP server, though it may not be installed by default along with the web server. Ensure that you have an SMTP server installed and properly configured.



As shown above, the **Connection.SMTP** element is added beneath the Connections element in the _Settings definition and its attributes are configured as appropriate for the SMTP server.

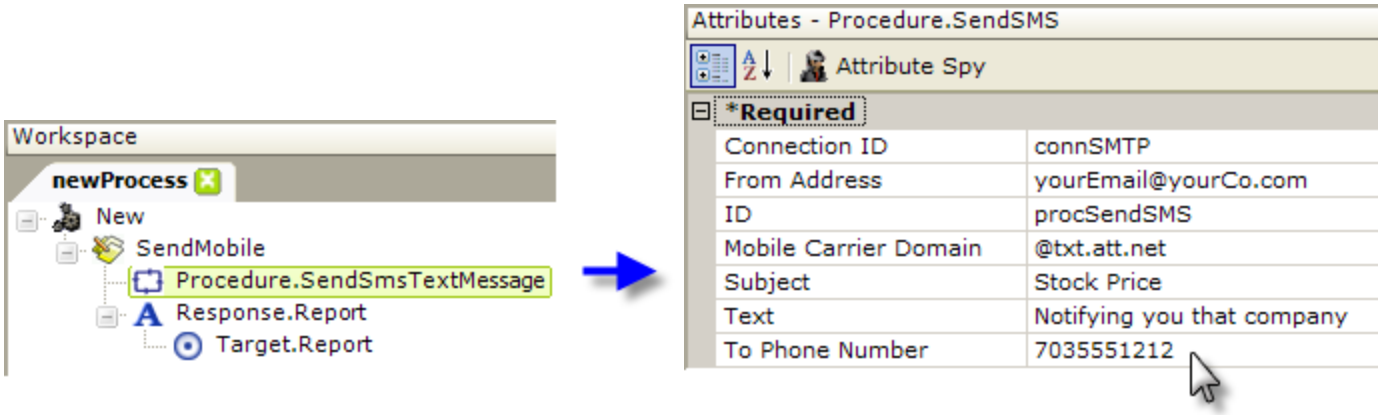
The following attributes are available for Connection.SMTP:

Attribute	Description
ID	(Required) A unique element identifier.
SMTP Authentication Account	Specifies the account name to be used to login to the SMTP server.

Attribute	Description
SMTP Authentication Method	Specifies the method to be used to authenticate the login. Values can be: *0* = None (the default) *1* = Login *2* = Use Cram-MD5 challenge-response authentication *3* = Use TLS/SSL encrypted communications. May require use of SMTP Port = *587*
SMTP Authentication Password	Specifies the account password to be used to login to the SMTP server.
SMTP Connection Timeout	Specifies the amount of time, in seconds, before the request to connect to the SMTP server is presumed to have failed. Default: *30 seconds*
SMTP EHLO Domain	Specifies an alternate EHLO domain name to be sent to the server.
SMTP Port	Specifies the port used by the SMTP server. Default: *25* TLS/SSL Authentication may require port *587*
SMTP Server	(Required) Specifies The computer name or IP address of the SMTP server.

Sending Email to the Phone

Once the connection has been configured, you can proceed to create a process task that will send the email to the phone.



As shown above, a **Procedure.SendSmsTextMessage** element is added beneath a **Task** element in the process definition. Its attributes are set as follows:

1. **Connection ID** - The ID of the Connection.SMTP element configured in the _Settings definition.
2. **From Address** - The address that will appear as the sender's address in the text message on the cell phone.
3. **ID** - The usual required, unique element ID.
4. **Mobile Carrier Domain** - The mobile carrier's domain; be sure to include the @ symbol!
5. **Subject** - The message's subject text. This can be parameterized, for example, with an @Data or @Request token.
6. **Text** - The message text. This can be parameterized, for example, with an @Data or @Request token.
7. **To Phone Number** - The cell phone number, without any dashes or other separators. This can be parameterized, for example, with an @Data or @Request token.

When the Task is run, the message will be send to the cell phone. The **If Error** element can be used beneath Procedure.SendSmsTextMessage to capture error messages and send them as a request parameter to a report definition for display. Because recipient information can be parameterized, the sending of messages can be driven from a datasource, making it very easy to send messages to a community of users at run time.

Bookmarks

Bookmarks provide a mechanism for storing and re-using the request parameters, user input data, and configuration choices made by users at runtime.

The following topics discuss the use of bookmarks:

About Bookmarks

Depending on how they're developed, Logi reports can offer users many opportunities to customize their reports at runtime. Developers can include user input controls and "super-elements", such as the Analysis Grid, in the reports. Super-elements have their own user interface and give users a lot of freedom to work with data, arrange table columns and rows, and add charts to their reports.

However, the time users invest in configuring their reports may be lost when their session ends. Logi "bookmarks" allow these settings to be saved, on an individual user basis, so users don't have to re-configure their reports each time they view them. Bookmarks can save the following:

- Values selected in user input controls
- Session variables and link parameters
- Super-element UI configurations

When a bookmark is created, it essentially "vacuums up" all the values that will become Request parameters, all super-element UI configuration settings, and all session variables and saves them in an XML data file. The contents of the file can be "replayed" at runtime to recreate the report, with all of its selections and settings intact.

 Bookmarks will not work with values passed using a Target-type element's **Request Forwarding** attribute. When this attribute is used, request variable values are not passed in a query string so they will not subsequently be available to be saved in a bookmark. To achieve the desired results, instead of using Request Forwarding, use a **Link Parameters** element to send request variable values to the target report.

Bookmark Types and Scope

Manual bookmark functionality, which uses special **Action.Bookmark** elements, is report-wide in scope. All of the values listed above will be saved. If a report definition includes two super-elements, the UI configuration values for *both* of them will be saved.

The special bookmark Action elements are available as children of Label, Button, Image, and other elements that can be used to initiate actions. These special elements are discussed in more detail in "Adding Manual Bookmarks to a Report" on page 268.

Automatic bookmarks provide similar functionality but their scope is limited to recording the configuration details and parameters of their parent super-elements. These details are saved automatically every time a user makes a change in the super-element

user controls. This makes working with bookmarks extremely easy. The **Auto Bookmark** element is available as a child of several super-elements and its use is discussed in "Adding Automatic Bookmarks" on page 273.

 You can use *either* manual *or* automatic bookmarks in a report, but not both.

Bookmark Data

Regardless of type, bookmark data is stored in one or more XML data files, called a "Bookmark Collection" and, typically, there's one collection per user. The developer specifies the storage location and ID of the collection. If Logi Security is being used, these files can be named using the @Function.UserName~ token, thus tying the bookmark collection files to an individual user.

Bookmarks can be saved in a SQL database instead of in the file system; see "Storing Bookmark, Gallery, and Save Files in a Database" on page 328.

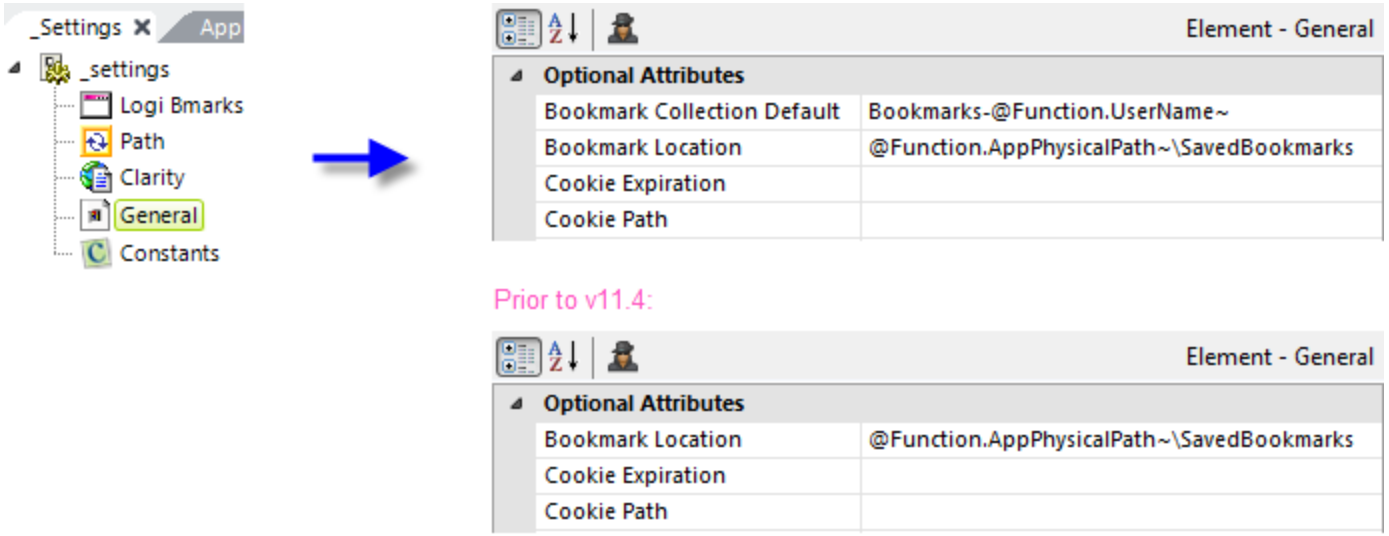
Bookmarks are identified by report name and a description. Special elements, discussed later, are available so developers can create their own Logi applications to manage bookmarks.

The **Bookmark Organizer** element allows users organize bookmarks into folders and sub-folders, which can be shared with other users.

Working examples of the Bookmark elements can be seen in our sample application.

Adding Manual Bookmarks to a Report

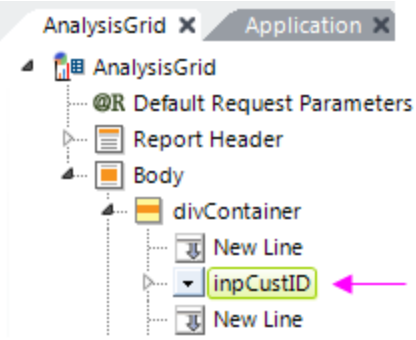
Adding manual bookmarks to your report is easy and the following examples provide step-by-step guidance. First, ensure that your report is a **good candidate** for bookmarks: it needs to include a super-element and/or some kind of user input selection that affects the displayed data. A simple report that just executes a SQL query and displays the entire results in a Data Table *is not* a good candidate, for example.



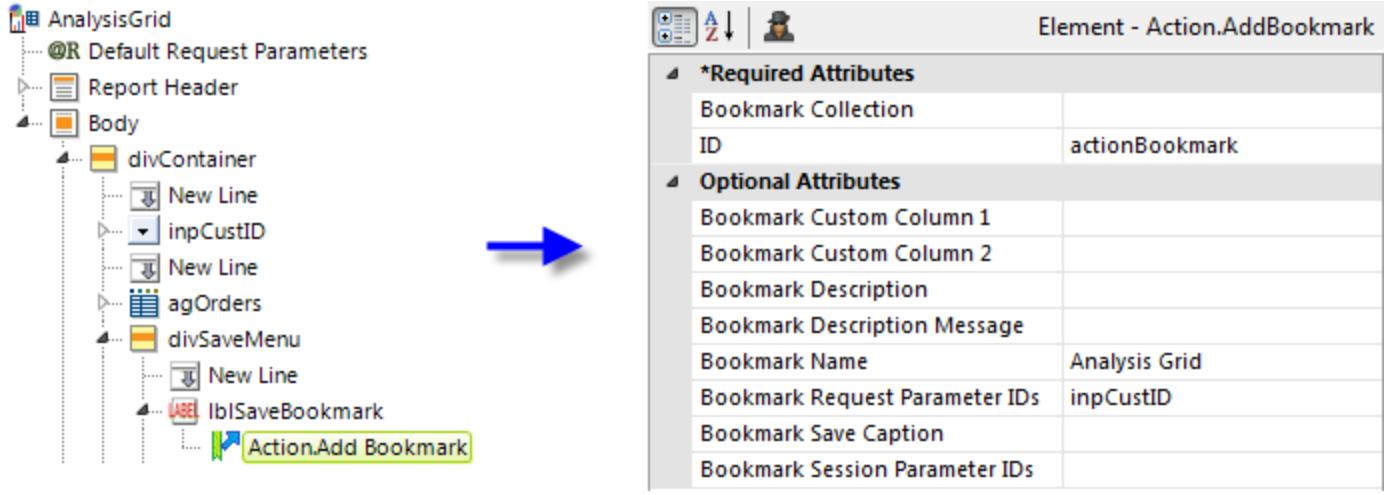
1. In Studio, open your application's **_Settings** definition and select the **General** element.
2. Set the **Bookmark Collection Default** attribute to the name for the bookmark data file. Do not include a path or .xml file extension. If you're using Logi Security, you can use the `@Function.UserName~` token to individualize the collection; if not, enter some other value of your choice.

   In general, specifying this value here and leaving the corresponding attributes blank in individual bookmark-related elements is good practice. If necessary, you can override this value by providing one in those other elements.

3. Set the **Bookmark Location** attribute value to the name of a folder, underneath the Logi application root folder where you want the bookmark files to be stored. You can use a token for the root folder here, as shown above. Don't use the system `rdDataCache` or `rdDownload` folders as these are periodically "cleaned up" automatically.
4. If it doesn't already exist in the file system, create the bookmark location folder and be sure it inherits all the file access permissions of the application root folder.

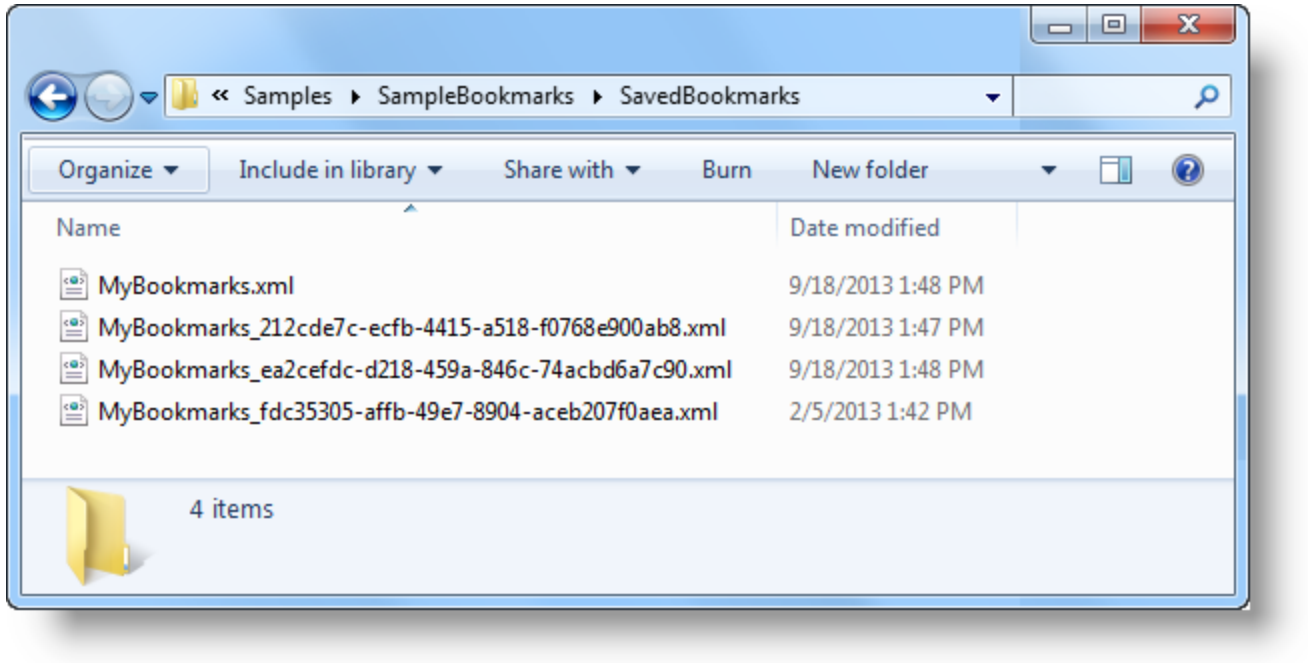


5. In the report definition you want to add bookmark features to, make a note of the element ID of any user input elements whose values you want to save. In the example shown above, there's an **Input Select List** element with an ID of "inpCustID" that we want to save.



6. In an appropriate place in the definition, add a Label, Button, Image or similar element, which will become the "Save Bookmark" link, as shown above.
7. Beneath it, add an **Action.Add Bookmark** element. Unlike most Action elements, it needs no Target element.
8. You can leave the **Bookmark Collection** attribute blank, unless you want to override the default value set in the _Settings definition.
9. Set its **ID** attribute to a *unique* ID. This is *very important*, especially if you have multiple reports with "Save Bookmark" links in the same application.
10. Set its **Bookmark Name** attribute to a meaningful name, usually the name of the report. This value can be used later, along with the Description, to identify individual bookmarks.
11. Set its **Bookmark Request Parameter IDs** attribute to a comma-separated list of one or more element IDs for user input control elements or link parameter IDs. In the example above, the element ID from Step 4 is entered. These identify the values to be stored in the bookmark; any value from this report that would be referenced in a subsequent report using an @Request token can be identified here.

That's all it takes to save bookmarks. Run the report, click the Save Bookmark link, and inspect the files created in the specified bookmark file location. Assuming a bookmark collection name of "MyBookmarks", you should see:



The first file is the main bookmark file and will always be present. The other files, with a GUID as part of their names, are referenced in the main file and contain information about any super-elements in the report. These may nor may not exist, depending on your report definitions.

Here are the other Action.Add Bookmark element attributes you might want to use:

Attribute	Description
Bookmark Custom Column 1 Bookmark Custom Column 2	Specifies a custom value to saved in the bookmark, which will be returned by DataLayer.Bookmarks. This could be use to provide additional information or datalayer filtering.
Bookmark Description	Normally, when saving a bookmark, the user enters a description of their own. This attribute value specifies custom default description text.
Bookmark Description Message	Specifies the prompt text for the description in the bookmark "save" dialog box. When this attribute has a value and the bookmark link is clicked at runtime, a dialog box will open prompting the user to enter a description, which is saved in the bookmark. If this attribute has no value, no dialog box will be displayed when the link is clicked.
Bookmark Save Caption	Specifies a custom caption for the Save button that appears in the bookmark dialog box. The default caption is "Save".
Bookmark Session Parameter IDs	Specifies one or more Session variables names, in a comma-separated list, to be saved in the bookmark. For example, to save the value of the token `@Session.UserID~`, you would enter *UserID* here.  If you create **Metadata** with **Custom Tables** and include Session tokens in its SQL queries, you should enter the session variables here. For example, if the query uses tokens like `@Session.CustomerID~` and `@Session.OrderID~` then enter `CustomerID,OrderID` here.

Adding Automatic Bookmarks

Adding *automatic* bookmarks to your report is easy and the following examples provide step-by-step guidance.

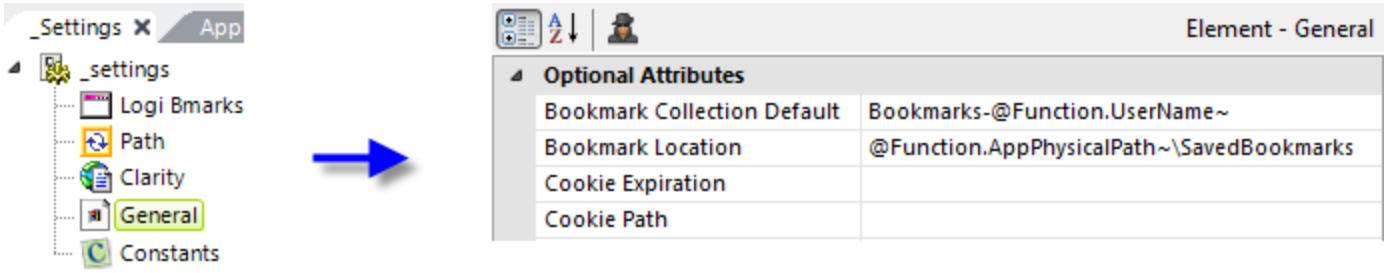
Auto Bookmark Attributes

Here are the attributes for the Auto Bookmark element:

Attribute	Description
Bookmark Collection	(Required*) Specifies the name of a bookmark collection. Typically, there is one collection for each user. In this case, specify @Function.Username~ here.. For consistency, set the _Settings definitions General element's BookmarkCollectionDefault attribute with a global value, rather than setting BookmarkCollection in individual elements. ** This attribute is not required when a global value has been specified in _Settings.*
Bookmark ID	(Required) Specifies a unique ID for the bookmark. The ID is returned by DataLayer.Bookmarks.
Bookmark Custom Column 1 Bookmark Custom Column 2	Specifies a custom value to saved in the bookmark, which will be returned by DataLayer.Bookmarks. This could be use to provide additional information or datalayer filtering.

Attribute	Description
Bookmark Description	Specifies default bookmark description text.
Bookmark Name	Specifies a text string saved in the bookmark and returned by DataLayer.Bookmarks. It is typically used to store the report name.
Bookmark Request Parameter IDs	Specifies one or more Request variables names, in a comma-separated list, to be saved in the bookmark. For example, to save the value of the token `@Request.State~`, you would enter *State* here. Values for Request variables specified here will be automatically saved when the report is submitted or refreshed.
Bookmark Session Parameter IDs	Specifies one or more Session variables names, in a comma-separated list, to be saved in the bookmark. For example, to save the value of the token `@Session.UserID~`, you would enter *UserID* here. Values for Session variables specified here will be automatically saved when the report is submitted or refreshed.  If you create **Metadata** with **Custom Tables** and include Session tokens in its SQL queries, you should enter the session variables here. For example, if the query uses tokens like `@Session.CustomerID~` and `@Session.OrderID~` then enter `CustomerID,OrderID` here.

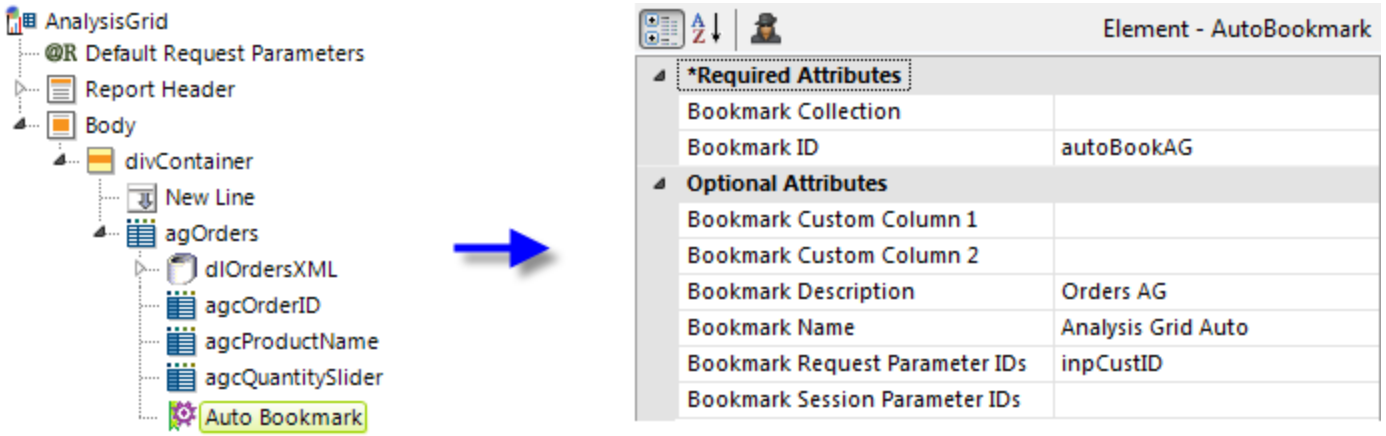
Ensure that your report is a good candidate for automatic bookmarks: it needs to include one of the super-elements listed in the earlier About Bookmarks section of "Bookmarks" on page 265. Remember that automatic bookmarks *do not* record values for user input controls outside of super-elements.



1. In Studio, open your application's **_Settings** definition and select the **General** element.
2. Set the **Bookmark Collection Default** attribute to the name for the bookmark data file. Do not include a path or .xml file extension. If you're using Logi Security, you can use the `@Function.UserName~` token to individualize the collection; if not, enter some other value of your choice.

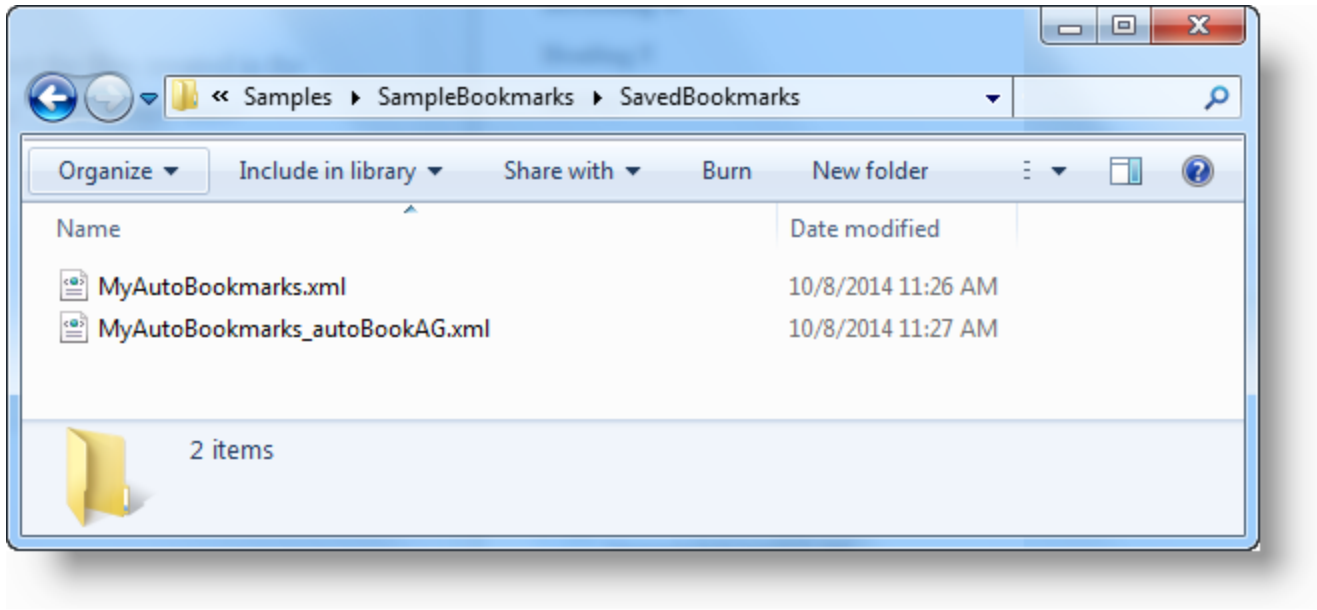
   In general, specifying this value here and leaving the corresponding attributes blank in individual bookmark-related elements is good practice. If necessary, you can override this value by providing one in those other elements.
3. Set the **Bookmark Location** attribute value to the name of a folder, underneath the Logi application root folder where you want the bookmark files to be stored. You can use a token for the root folder here, as shown above. Don't use the system `rdDataCache` or `rdDownload` folders as these are periodically "cleaned up" automatically.
4. If it doesn't already exist in the file system, create the bookmark location folder and be sure it inherits all the file access per-

missions of the application root folder.



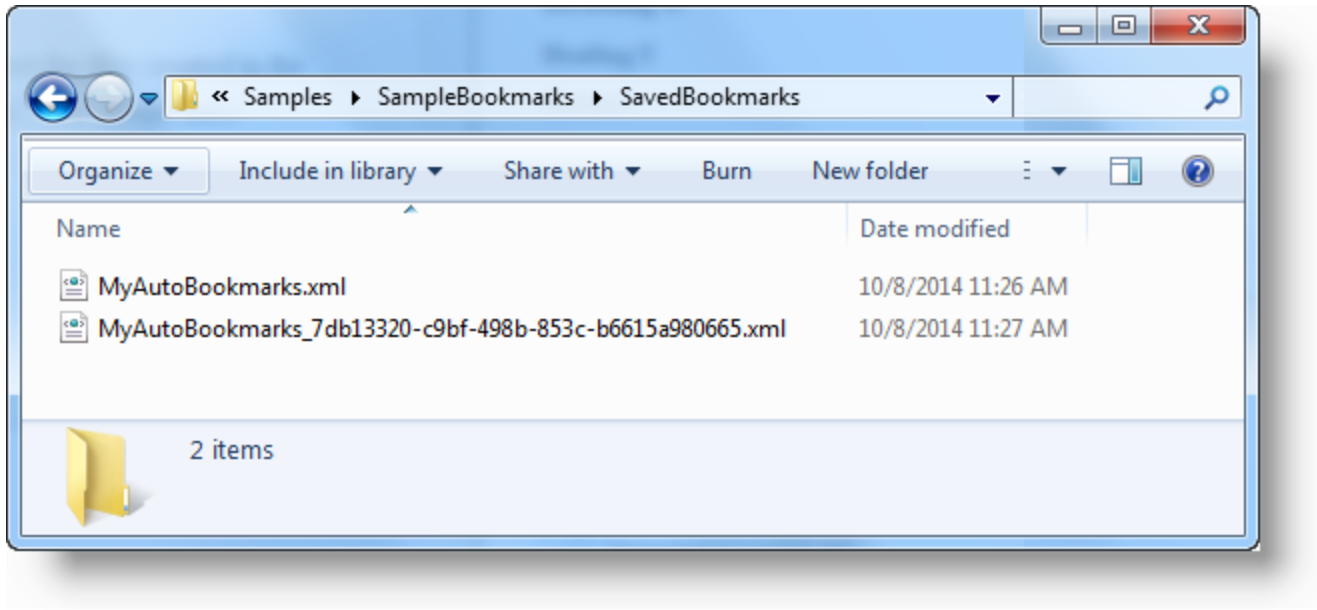
5. Add an **Auto Bookmark** element as a child of the target super-element, as shown above.
6. You can leave the **Bookmark Collection** attribute blank, unless you want to override the default value set in the _Settings definition.
7. Specify a unique **Bookmark ID** to differentiate this bookmark in the collection. This value will be incorporated into the bookmark file names, so don't use any exotic characters here that the web server file system considers invalid in file names.
8. With the exception of **Allow Shared Bookmark Updates** (discussed later), the rest of the element's attributes are identical to those for Action.Add Bookmark but, of course, without those for the Save Bookmark dialog box user interface. Refer to the table in "Adding Manual Bookmarks to a Report" on page 268 for information about these attributes.
9. The last thing you need to do to is initiate automatic bookmarks. This is done by calling the report with `rdLoadBookmark=True` in the query string. One way to do this is by using Link Parameters with the link that calls the report.

Now you can run the report and manipulate the super-element interface. After that you can inspect the files created in the specified bookmark file location. Assuming a bookmark collection name of "MyAutoBookmarks", you should see:



The first file is the main bookmark file and will always be present. The other file, with Bookmark ID you specified as part of its name, is referenced in the main file and contains information about the super-element in the report. It will appear once the super-element UI is manipulated.

Here are some additional important usage tips:



To **automatically generate** the Bookmark ID: If you want the application to generate the Bookmark ID automatically, using a GUID, as shown above, leave the element's Bookmark ID attribute blank and call the report with `rdNewBookmark=True` in the query string.

With **Report Center Menu**: If the report includes a **ReportCenter Item**element, the automatically-created bookmark will be listed with the ReportCenter Menu element.

With **Dashboards**: If using Auto Bookmark with a Dashboard, *do not* set the Dashboard element's **Save File** attribute. The bookmark will in effect become the Dashboard's Save File.

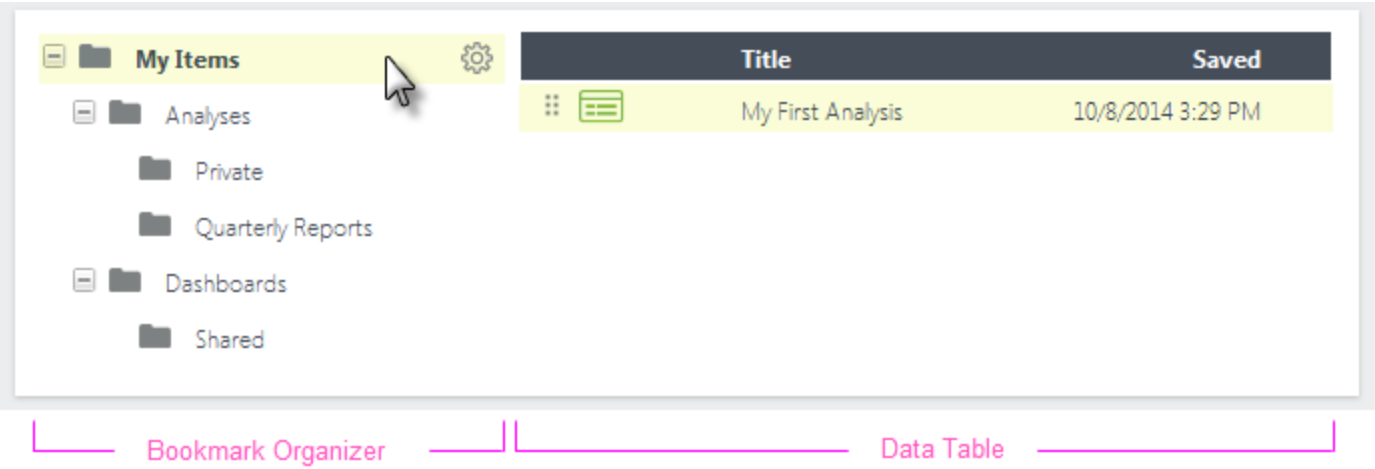
Organizing Bookmarks

Users may want to group and organize the bookmarks they create. The **Bookmark Organizer** element allows users to place their bookmarks into "bookmark folders" at runtime.

A bookmark folder is not an actual folder in the web server file system; instead, the bookmark data is referenced in the master bookmark data file in a hierarchy that creates "logical folders". The Bookmark Organizer lets users manage bookmarks and folders. Multiple bookmarks can be collected into a folder and folders can have sub-folders.

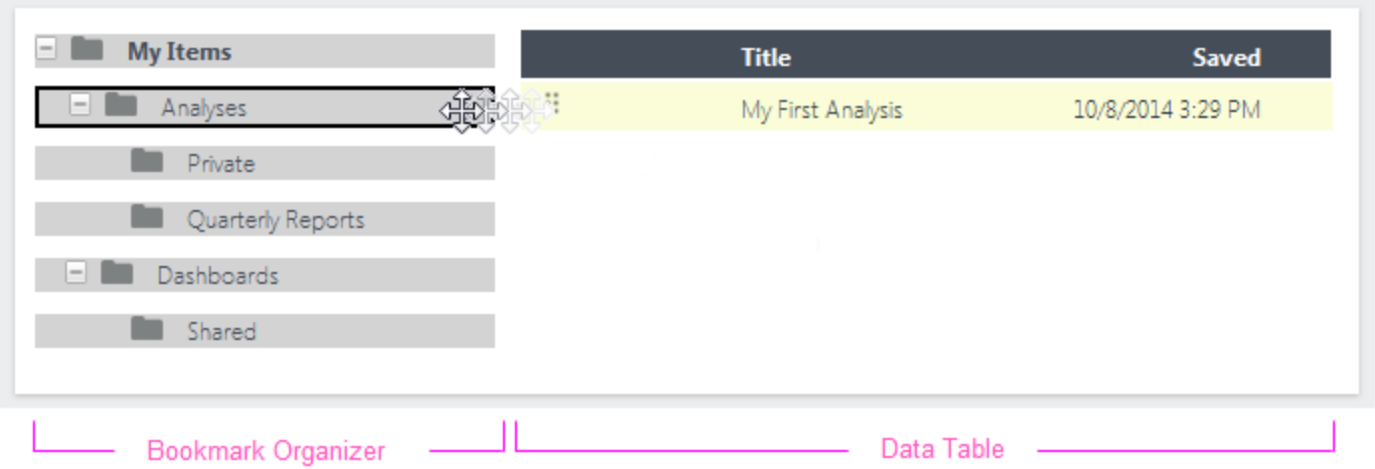
Runtime Display and Manipulation

The Bookmark Organizer displays the bookmark folders and works with a Data Table element, in the same definition, that lists the bookmarks. Clicking an entry in the Data Table runs the "bookmarked" report. When a folder is clicked, the Data Table is refreshed to show the contents of that folder. Folders can be created, renamed, and deleted, and bookmarks are organized by dragging them into folders. Here's an example:

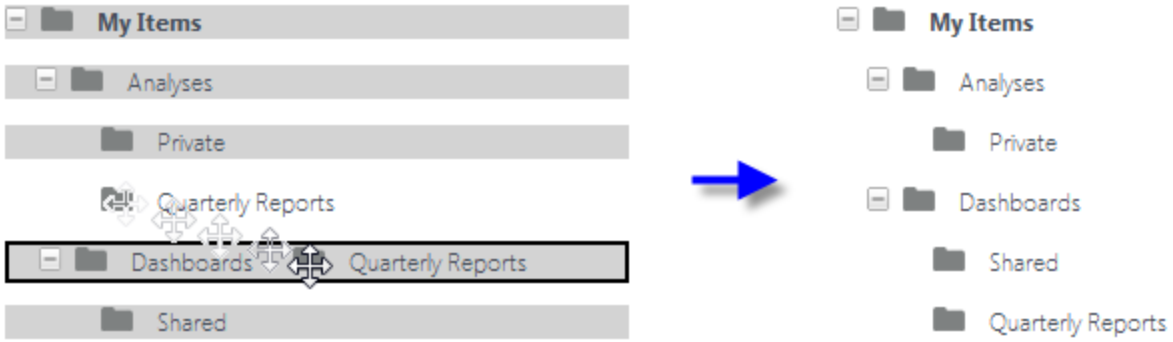


The example above shows a Bookmark Organizer and its companion Data Table. They can be physically arranged on the page however you desire, and theme selection and/or styling can vary their appearance.

The example shows the default "My Items" bookmark folder and several sub-folders created by the user. When bookmarks are first created, they're placed in the My Items folder by default. The "gear" icon appears when the mouse hovers over a folder and allows you to add new sub-folders, and rename or delete sub-folders. The Data Table displays the bookmarks in each folder (the selected folder name is shown in bold text).



To move a bookmark from one folder to another, the user just drags the Data Table row's drag icon to the desired folder, as shown above.

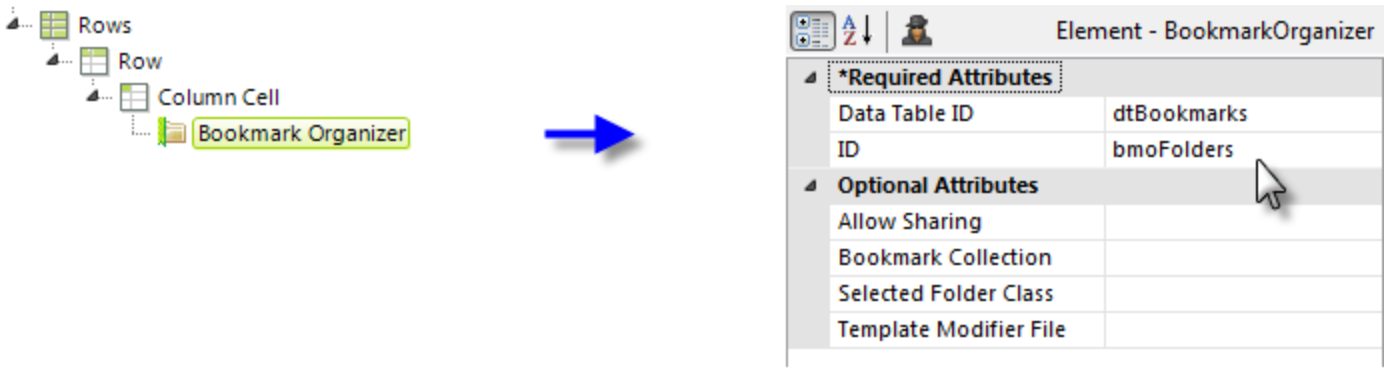


Sub-folders can also be rearranged into other folders by clicking their folder icon and dragging, as shown above.

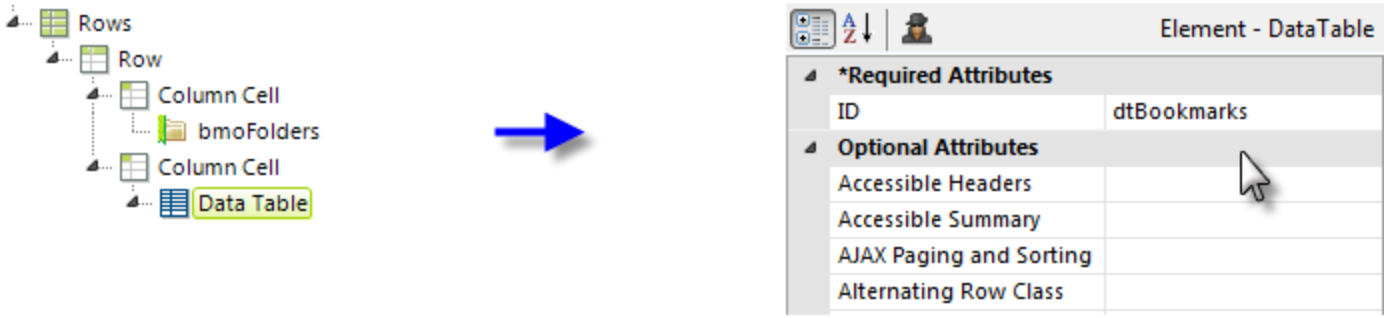
Implementation

As mentioned above, to implement this you'll need to add a **Bookmark Organizer** element and a companion **Data Table** element. Here's how:

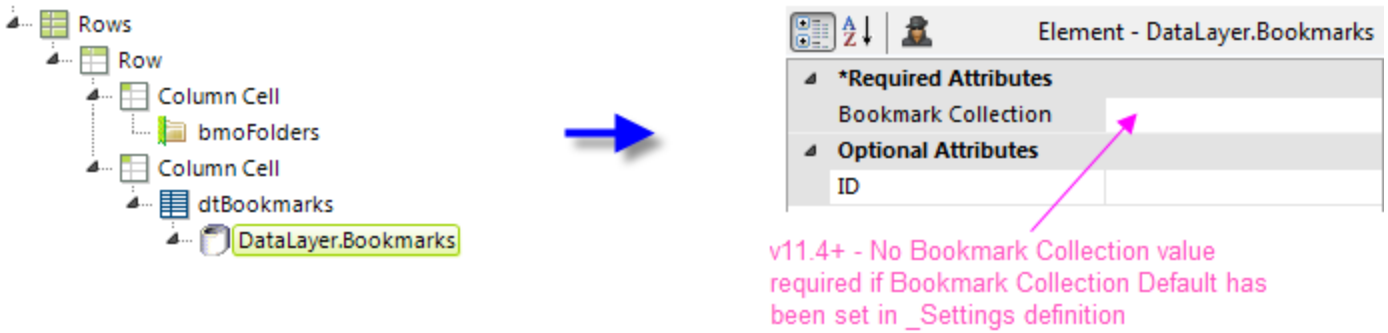
1. Implement manual or automatic bookmarks, as discussed in earlier sections.



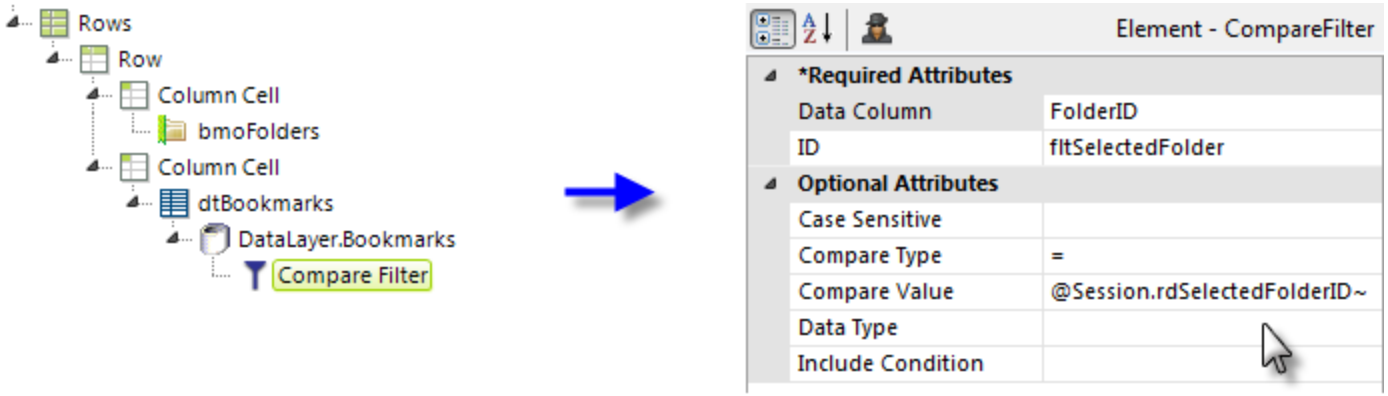
2. Include a **Bookmark Organizer** element in your report definition (one possible arrangement is shown above). Set its **Data Table ID** attribute to the element ID you'll give its companion Data Table.



3. Add the companion **Data Table**, as shown above, and ensure its ID matches the one entered in the previous step.

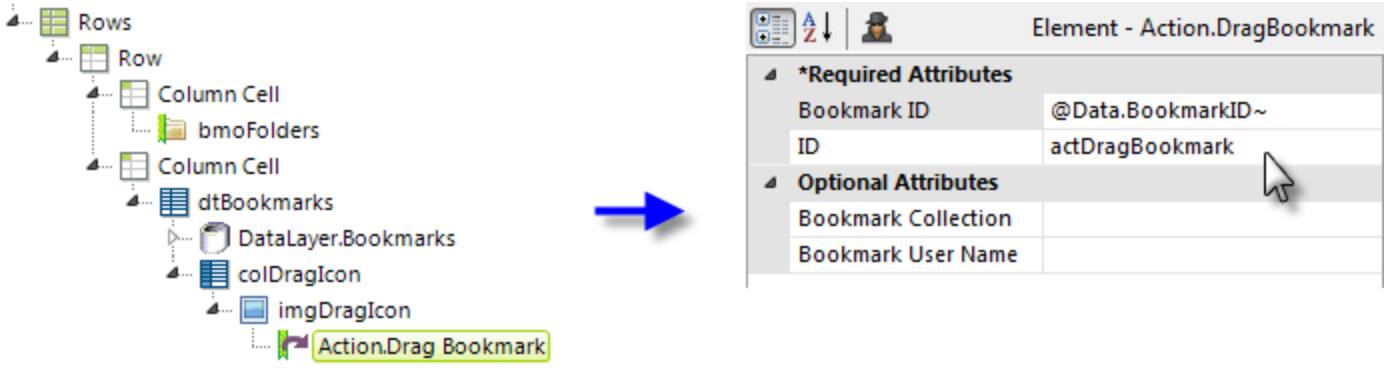


4. Add a *DataLayer.Bookmarks* element beneath the Data Table, as shown above. Unless you want to override the Bookmark Collection Default value set in `_Settings.lgx`, you can leave the collection name blank here.



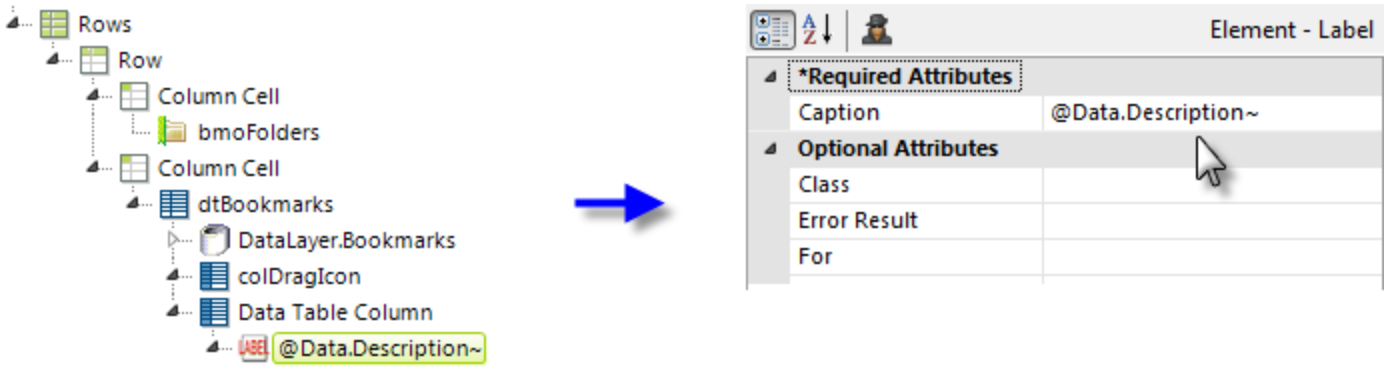
5. Add a **Compare Filter** beneath the datalayer and set its attributes as shown above. The Bookmark Organizer element automatically creates and updates the session variable `rdSelectedFolderID` each time the user selects a folder, so this filter

causes the Data Table to only show the bookmarks in the selected folder.

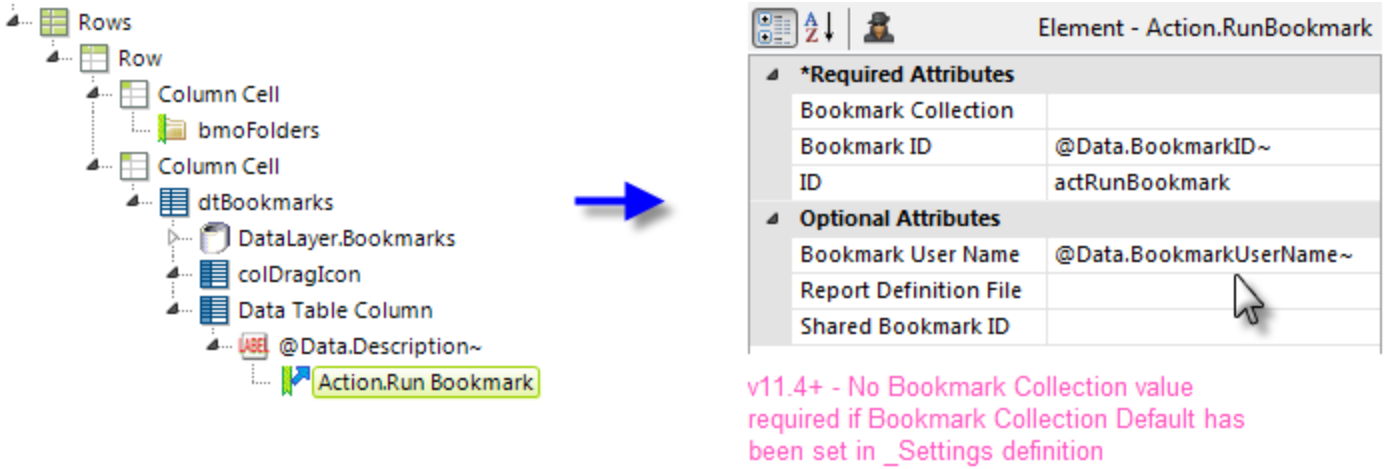


6. Add **Data Table Column**, **Image**, and **Action.Drag Bookmark** elements, as shown above. If you prefer, you can use a Label or Button element instead of the image. Set the action element's attributes as shown.

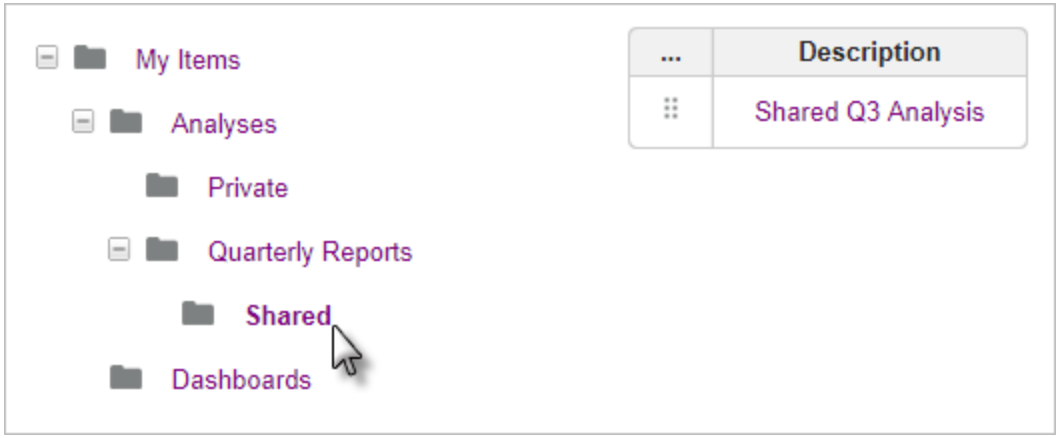
   If you want to use the standard "drag icon" image - ⠿ - provided with Logi Info, enter the following for the Image element's **Caption** attribute value: `../rdTemplate/rdBookmarkOrganizer/rdDragHandle.png`



7. Enter one or more **Data Table Column** and **Label** elements, as shown above, to display information identifying the bookmarks.



8. Finally, in order to actually run the bookmarks, add an **Action.Run Bookmark** element beneath the Label element and set its attribute, as shown above.



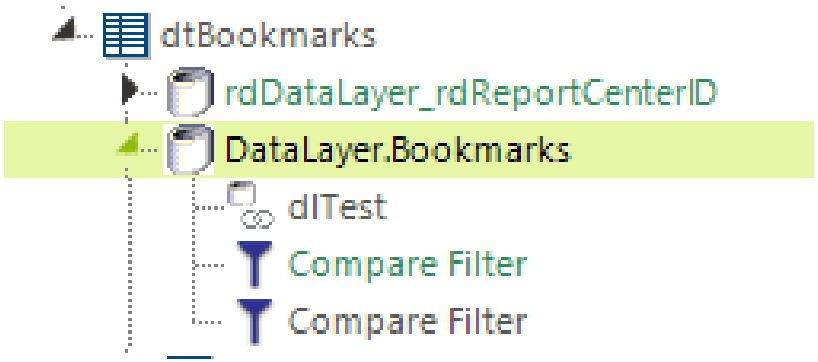
When run, with the Clarity theme applied, this simple example looks like the image shown above. The bookmark can be dragged into any of the folders and clicking its bookmark description will run the report and apply the bookmark.

Enable Item Counts

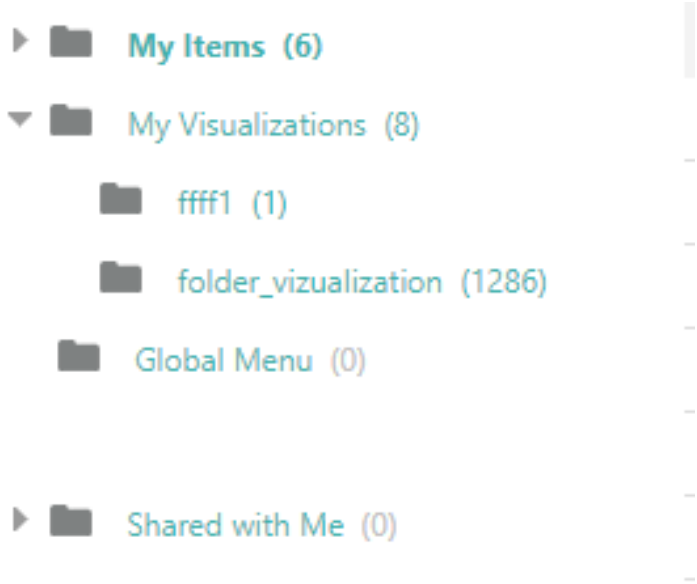
You have the option to display item counts for folders (My Items, My Visualizations, Global Menu, and Shared with Me). In Info Studio, two optional attributes were added to represent this new feature via BookmarkOrganizer: "BookmarkDatalayerLinkID" and "ShowItemCount".

```
<BookmarkOrganizer
    AllowSharing="True"
    BookmarkDatalayerLinkID=""
    DataTableID="dtBookmarks"
    ID="idBookmarkOrganize"
    ShowItemCount="True"
/>
```

By default, the "BookmarkDataLayerLinkID" attribute is empty, as shown above, and the Show Item Count works as usual. If "BookmarkDataLayerLinkID" has a value, the DataTable to which the DatalayerLink belongs to is the same as the DataTable specified by DataTableID.



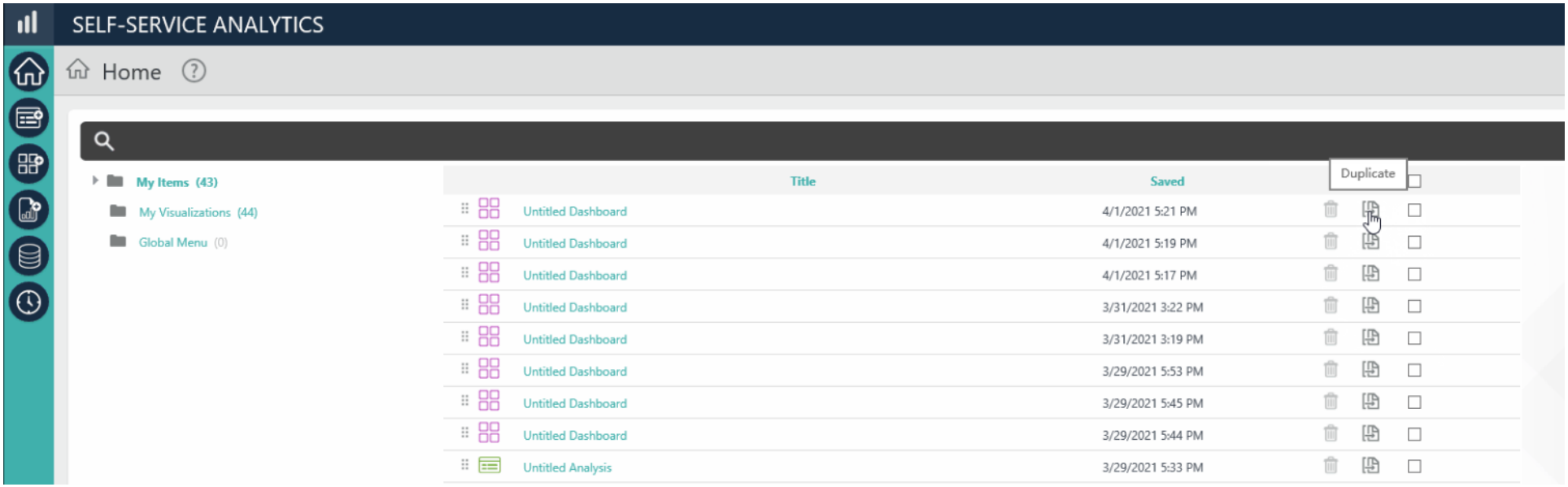
Sub-folders show item counts, as well. Parent folders show their own number, not including the items from its sub folder. This number will be updated automatically when a new item gets added to or deleted from the specific folder.



Duplicating Bookmarks

Bookmarks can be duplicated to store back ups and enable you to edit bookmarks that were shared with you, without making changes to the original.

Access the desired folder and select the **duplicate** icon next to the bookmark you want to copy:

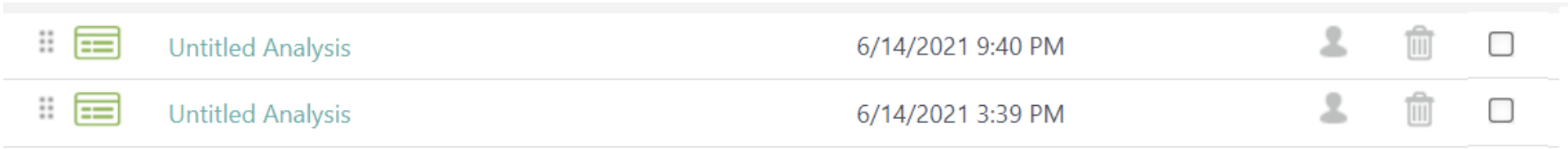


Duplicated bookmarks display in the same folder as the original, with the prefix "Copy of" to differentiate the two.

Batch Duplicating

If your application has been configured for it, you can also batch duplicate SSRM bookmarks to their original folder. To enable this feature, set `goAllowBatchBookmarkDuplication` to "True". By default, this constant is "False" and your screen will look like the

following:



You are still able to duplicate *individual* bookmarks and batch *delete* bookmarks when `goAllowBatchBookmarkDuplication` is set to "False".

Once this feature has been enabled, choose which bookmarks to copy by selecting the corresponding **check box**, or select all bookmarks in a folder by checking the check box at the top of the list:

Then, select the **duplicate** icon:

Duplicated bookmarks display in the same folder, as shown below:

Batch duplicating shared bookmarks stores them in the "My Items" folder.

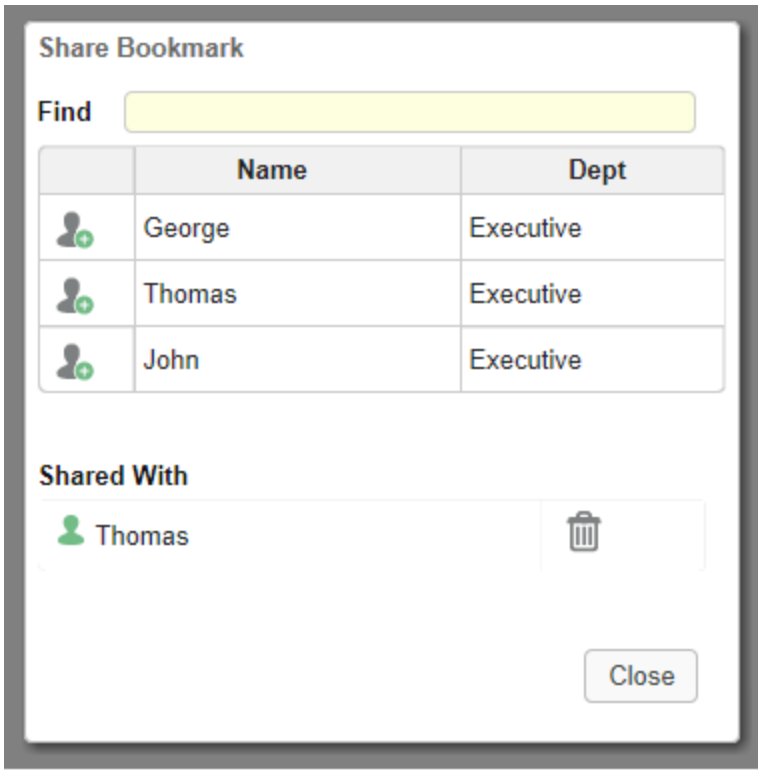
Sharing Bookmarks

Up to this point, we've seen how bookmarks can be used to preserve an individual user's configurations. It might also be useful to be able to *share* configurations with other users of the same application. Once the Bookmark Organizer has been implemented, adding sharing takes very little development time. This topic describes sharing in general. For information about sharing with groups, see "Sharing with Groups of Users" on page 317.

Bookmark sharing is enabled by setting the Bookmark Organizer element's Allow Sharing attribute. Some variety of Logi Security must also be enabled so that the application knows who the users are.

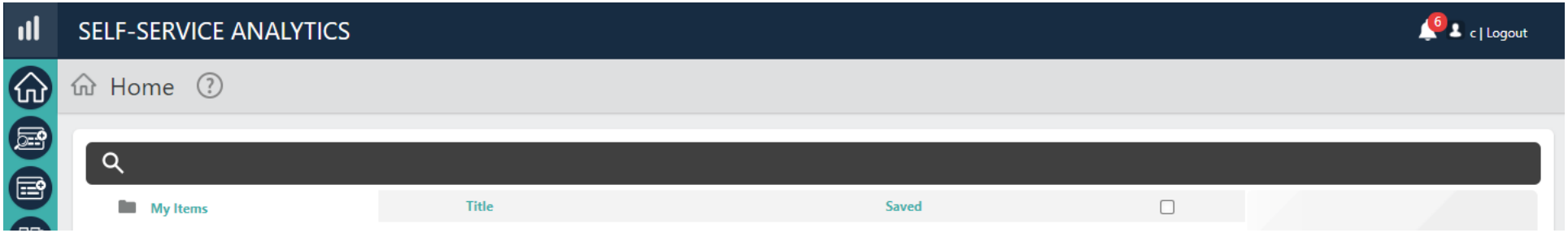
Users who create bookmarks can share them by giving other users permission to access them. Here's our earlier organization example:

We've added a few icons in the bookmark row for sharing and deleting the bookmark, and you'll notice the new Shared with Me folder has appeared. This folder will contain links to bookmarks that have been shared with you by other users.

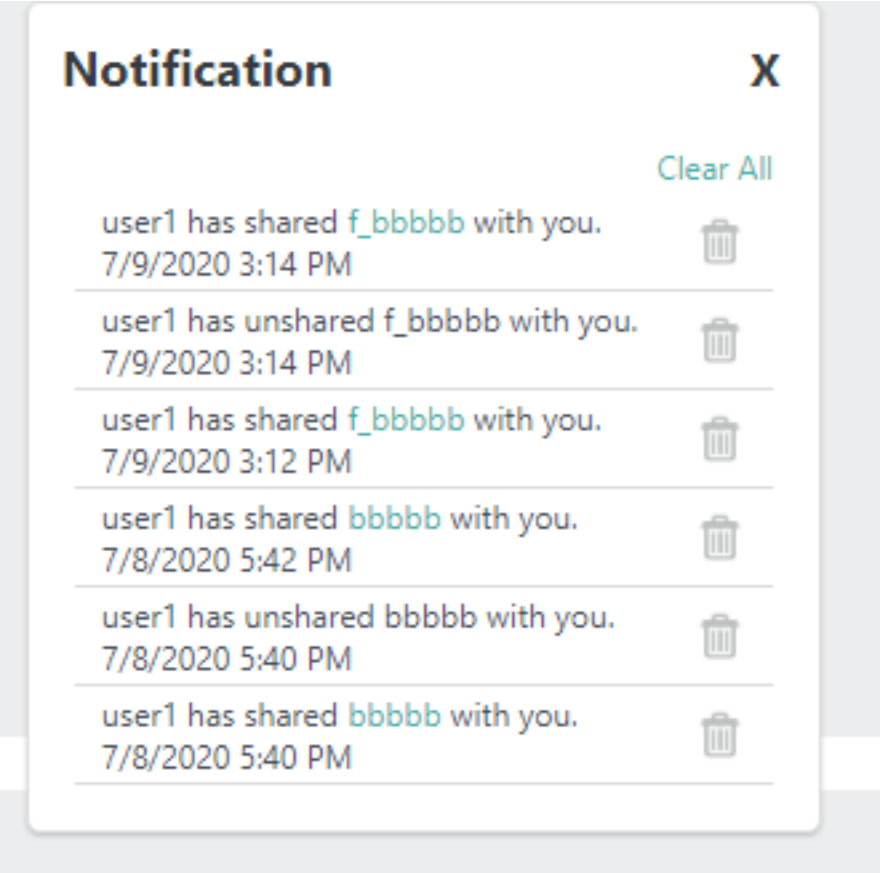


When the Share icon is selected, a modal pop-up panel (shown above) displays. In it, users can select the other users who can share the bookmark. Developers can control the styling and content of the pop-up panel.

If your application has been configured for it, you will receive a notification on the SSRM Home page when an item is shared/un-shared with you:

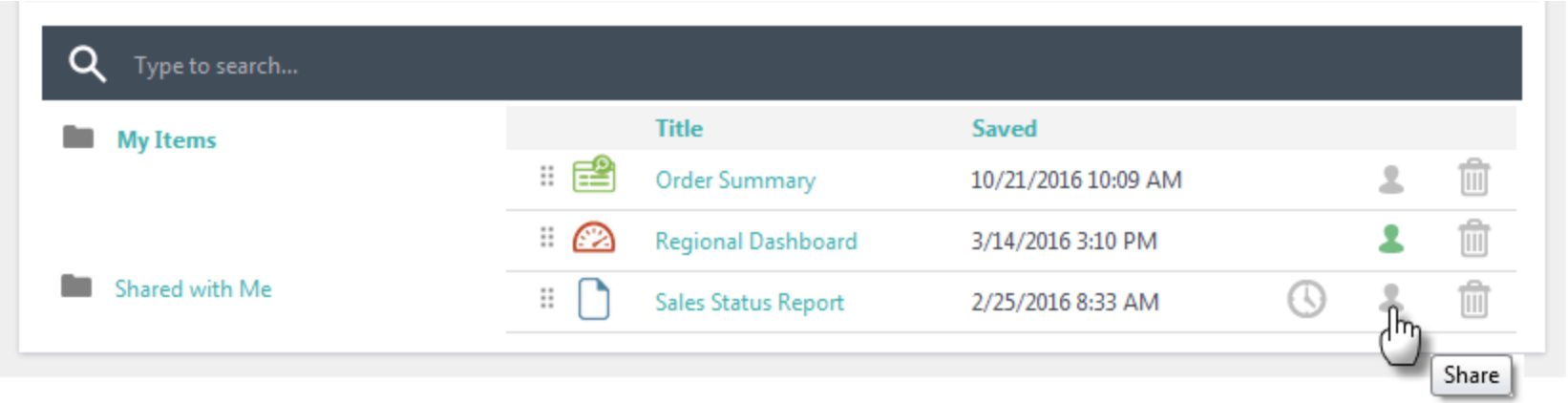


Selecting this notification will display information about the author and shared folder. Select the **shared item** to access the folder:

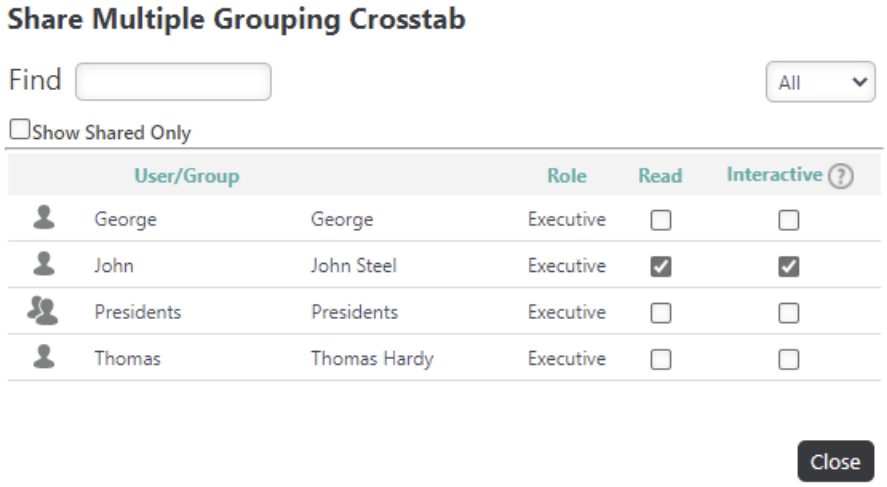


Sharing Permissions

As a content creator, you can now manage access for content shared with other users. With this enhancement, you can configure different access types, "Read" or "Interactive", at the individual bookmark level. Selecting the **share** icon produces new access types in the sharing pop-up window.



Developers can control the styling and content of the pop-up panel.



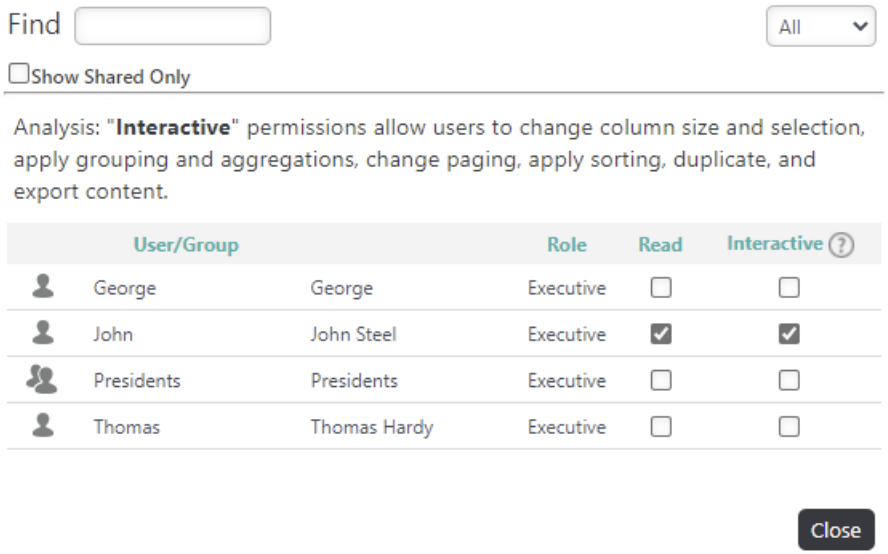
Below is a list of the types of permissions available when sharing bookmarks:

Permission	Access Type
Read	• Read-only
Interactive	• Schedule reports • Utilize Drill-to feature • Apply Dashboard filters • Add aggregates • Duplicate bookmark • Export content • Interact with charts

All of the changes made in Interactive mode are local and limited to that particular session; they can not be saved to the original bookmark, as this can only done through specific security roles. Selecting the "Interactive" permission automatically selects the "Read" check box, as well. Note that you must select at least one access type to share content. You can edit these permissions by checking/unchecking the corresponding check boxes. Deselecting the "Read" check box automatically deselects the "Interactive" check box (if it was selected), and in this case, unshares the content entirely. Schedules launched by the shared user are removed if their permission changes to "Read". Likewise, if the report or folder is unshared with that user, schedules launches by the shared user are also removed.

- Sharing permissions are only available when the constant `goSharePermissionEnable` is set to "True". For more information about configuring InfoGo to enable sharing, see Setting Up Sharing.
- All permissions are granted based on the type of bookmark, honoring Security Right IDs and Object Level permissions. Bookmarks shared prior to this enhancement have the "Read" check box selected by default.

Select the **question mark** icon next to the Interactive permission to expose a permission description:

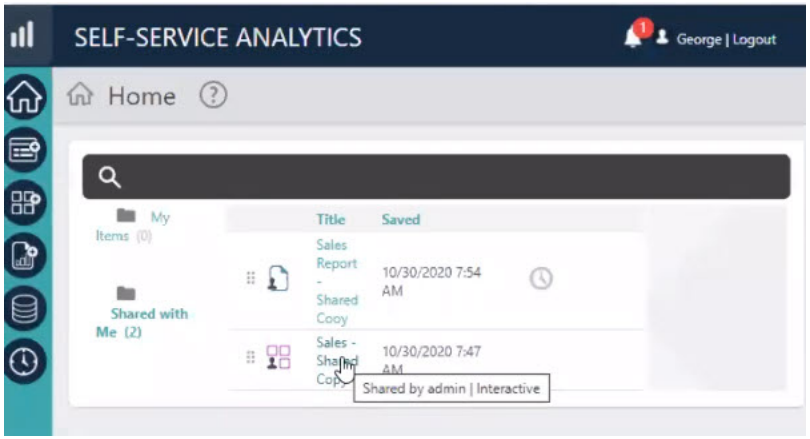


As users are selected and included in the "Shared With" list, their entries will be removed from the list of available users. To view users that a bookmark has already been shared with, select the **Show Shared Only** check box.

All permissions are granted based on the type of bookmark, honoring Security Right IDs and Object Level permissions. Bookmarks shared prior to this enhancement have the "Read" check box selected by default.

Self-Service User

As a self-service user, you now have the ability to interact, schedule, and filter the content being shared with you. Hovering over the item in your "Shared With Me" folder exposes a tooltip that indicates who shared the item with you, as well as your permission type:



If a report was shared with you and you have an "Interactive" permission, a clock icon displays next to the shared report, like above.

Upon selecting "Duplicate", the bookmark reflects any user changes, such as filters, panels, ordering, etc. Duplicated items are stored in your "My Items" folder.

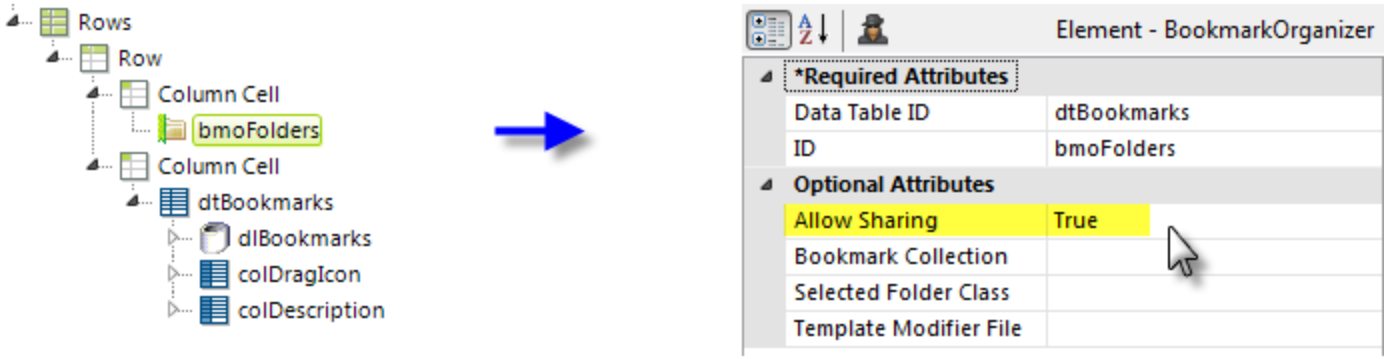
For more information on sharing permissions, see *Sharing Your Work*.

As users are selected and included in the "Shared With" list, their entries will be removed from the list of available users.

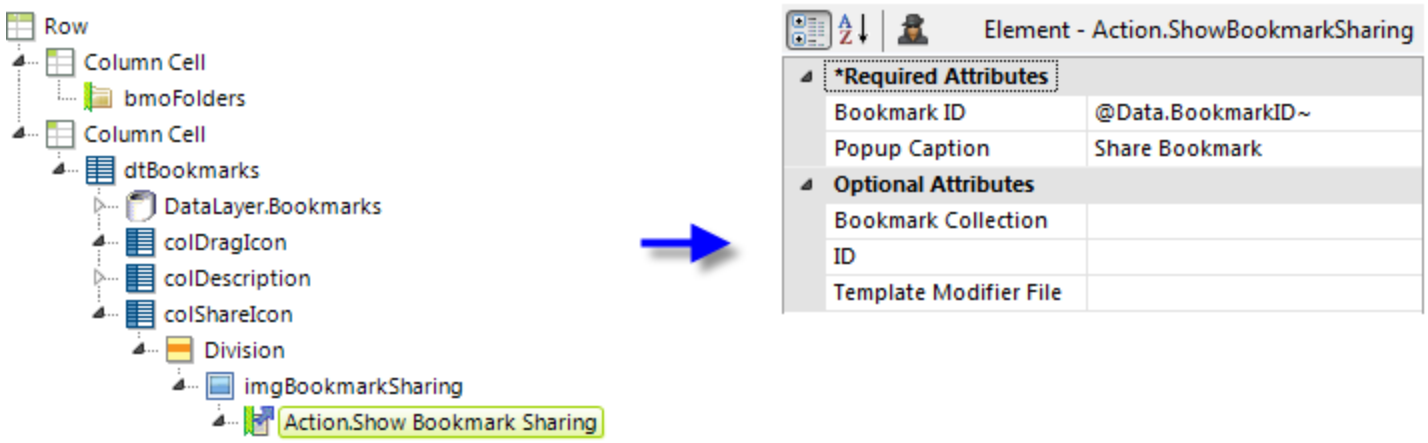
Implementation

Assuming you've already implemented the Bookmark Organizer, here's how to add bookmark sharing:

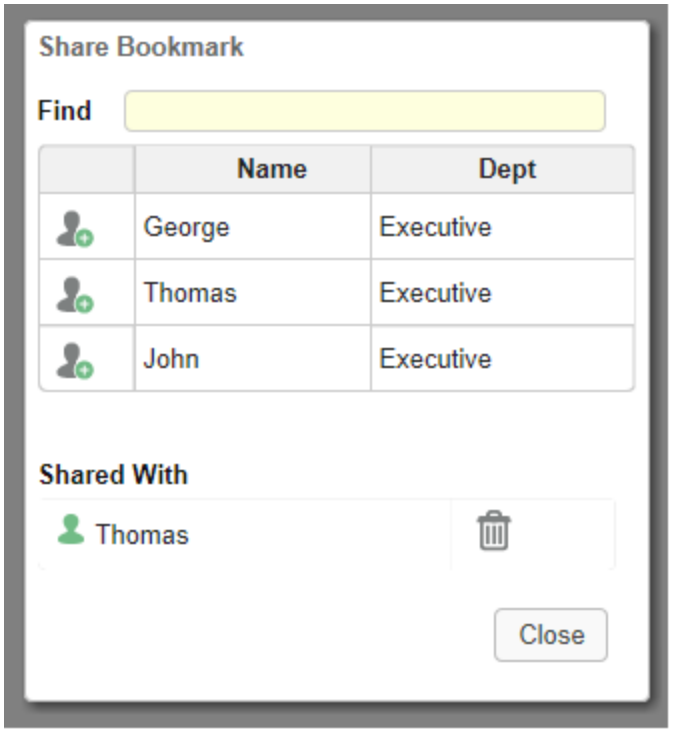
1. Ensure that some variety of **Logi Security** is being used.



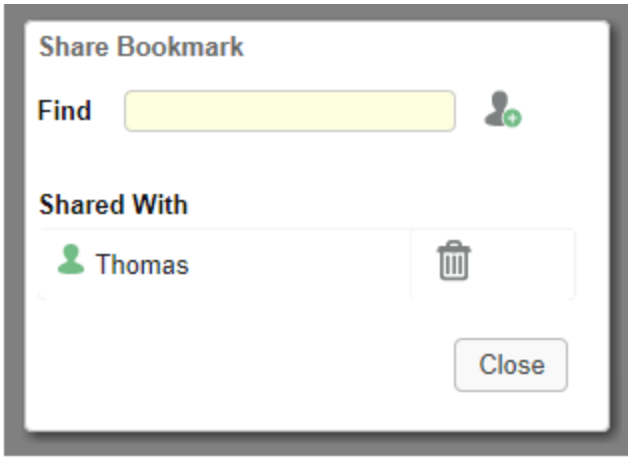
2. Building on our earlier example definition, set your **Bookmark Organizer** element's **Allow Sharing** attribute to *True* (tokens may be used here), as shown above.



3. Add a new Data Table column, with **Division**, **Image**, and **Action.Show Bookmark Sharing** elements, as shown above. Set the Action element's **Bookmark ID** attribute value to the Bookmark ID column returned in the datalayer.



With **Sharing List**

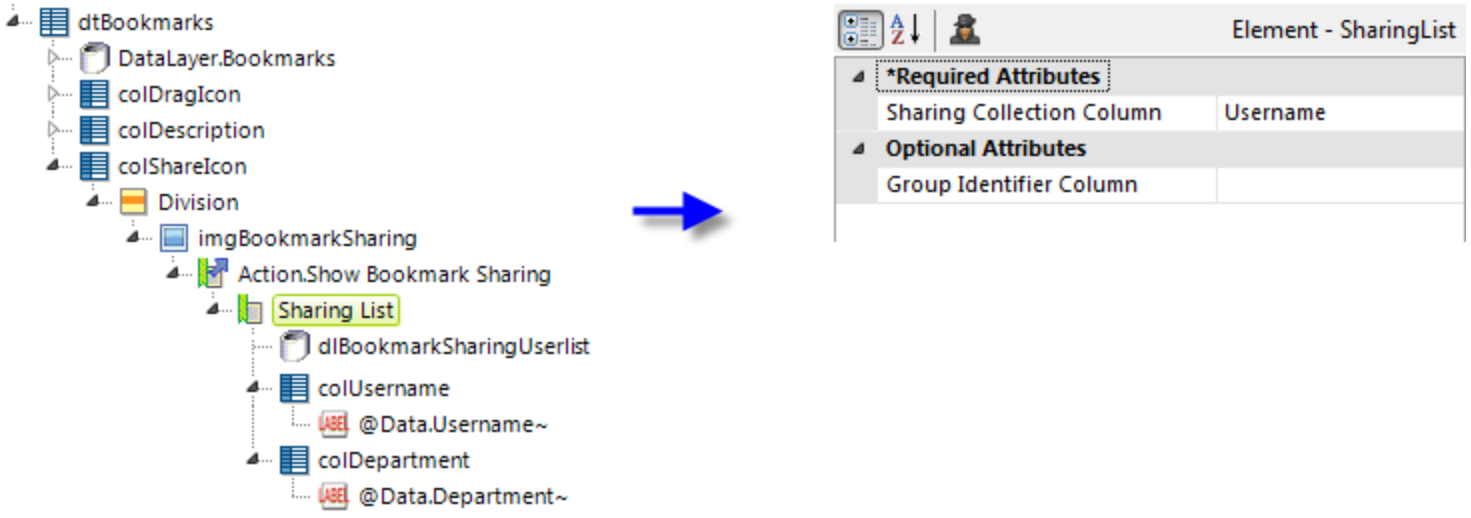


Without **Sharing List**

4. Next, decide if you want to the pop-up Share panel to include a pick-list of other users with whom a user can share bookmarks. This is determined by the addition of the **Sharing List** element beneath the Action element. Referring to the images shown above:

   With the Sharing List element, user names are displayed in an embedded table, as shown above right, and the names come from a datalayer. If a user in the table doesn't have a personal bookmark collection yet, one will be created for her automatically if a folder is shared with her. The Sharing List element also allows you to specify *user groups* for sharing.

Without the Sharing List element, user names can be typed-in and will be compared to the names associated with all of the existing personal bookmark collections. A type-ahead feature will help filter the available names.



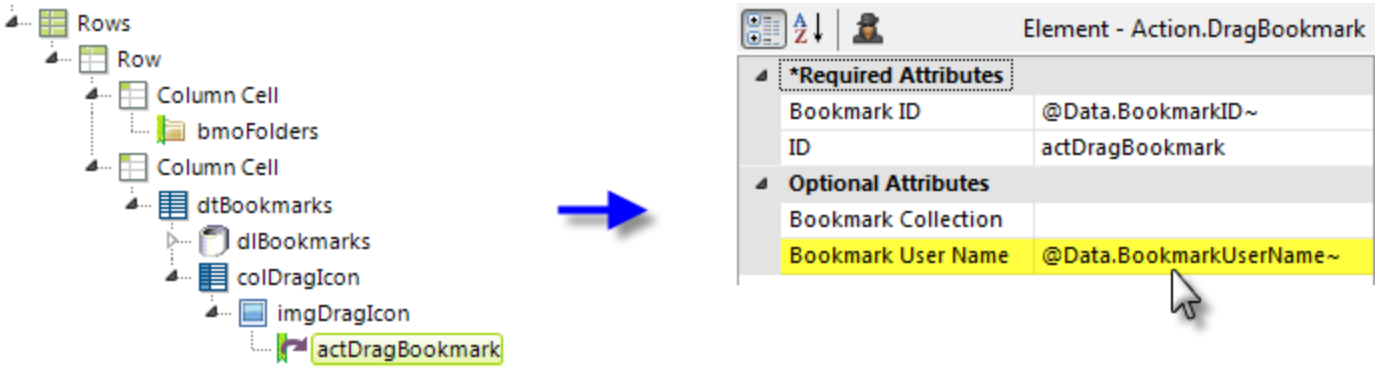
The example above shows the **Sharing List** element in use. You must specify the name of the datalayer column containing the user names in its **Sharing Collection Column** attribute. The `SharingCollectionDisplayColumn` property can be used to search for users by their first/last name. The default value is empty. Users can set the property value in `goCustomizations.goBookmarkSharingUserlist.lgx`. Once shared, the display name is saved in the bookmark and reflected in the Shared With list.

As a special type of Data Table, Sharing List gives developers some control over the contents and formatting of the list. Any type of datalayer can be used and users are selected by clicking table rows. You can add whatever descriptive data you want and the pop-up panel will automatically expand to include the width of the table.

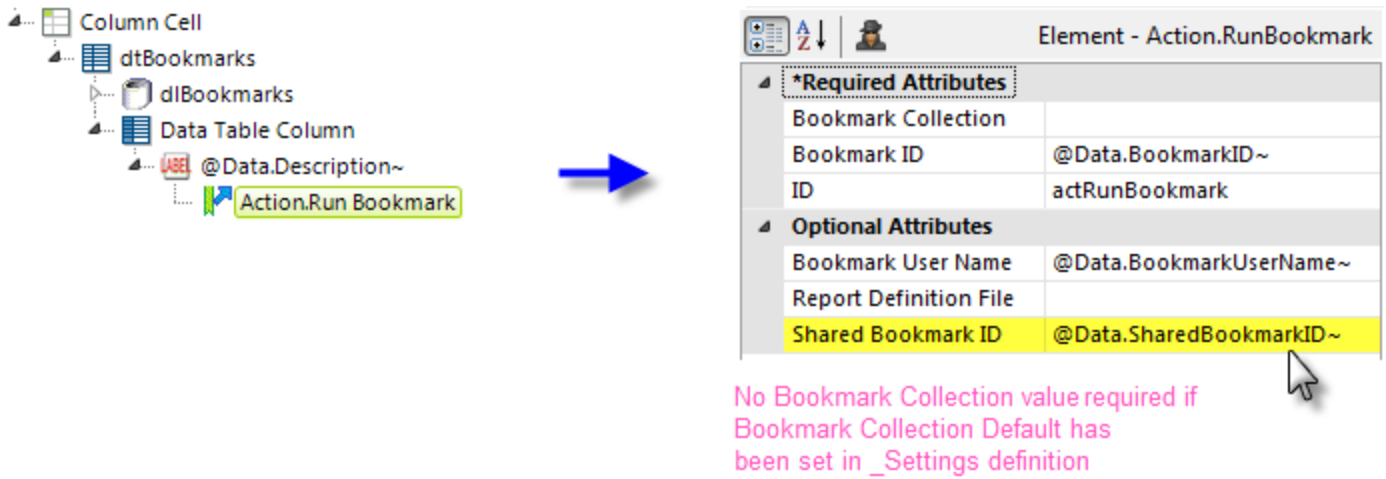
UserName	Department
Bob	Executive
Eddie	Executive
Roger	Employee

Sharing with users

The Sharing List element's datalayer result set only needs columns for the username and some descriptive value, like a department name.



5. Find the **Action.Drag Bookmark** element you're using in the bookmarks Data Table and set its **Bookmark User Name** attribute as shown above. The *BookmarkUserName* column is returned by DataLayer.Bookmarks. As with all tokens, this is case-sensitive so mind your spelling.



6. Find any **Action.Run Bookmark** elements you're using in the bookmarks Data Table and set their **Shared Bookmark ID** attributes as shown above. The *SharedBookmarkID* column is returned by DataLayer.Bookmarks. As with all tokens, this is case-sensitive so mind your spelling.

And now you have bookmark sharing ready to go.

In your _Settings definition, use the **Security** element's **Development Username** attribute to run the application as different users in order to see that bookmarks are shared correctly.

A nice refinement you may want to add is to have differently colored sharing icons to indicate whether a bookmark has already been shared with others or not. To do this you'll need the two icons (assumed to be in _SupportFiles) and these formulae for the Image element's attributes:

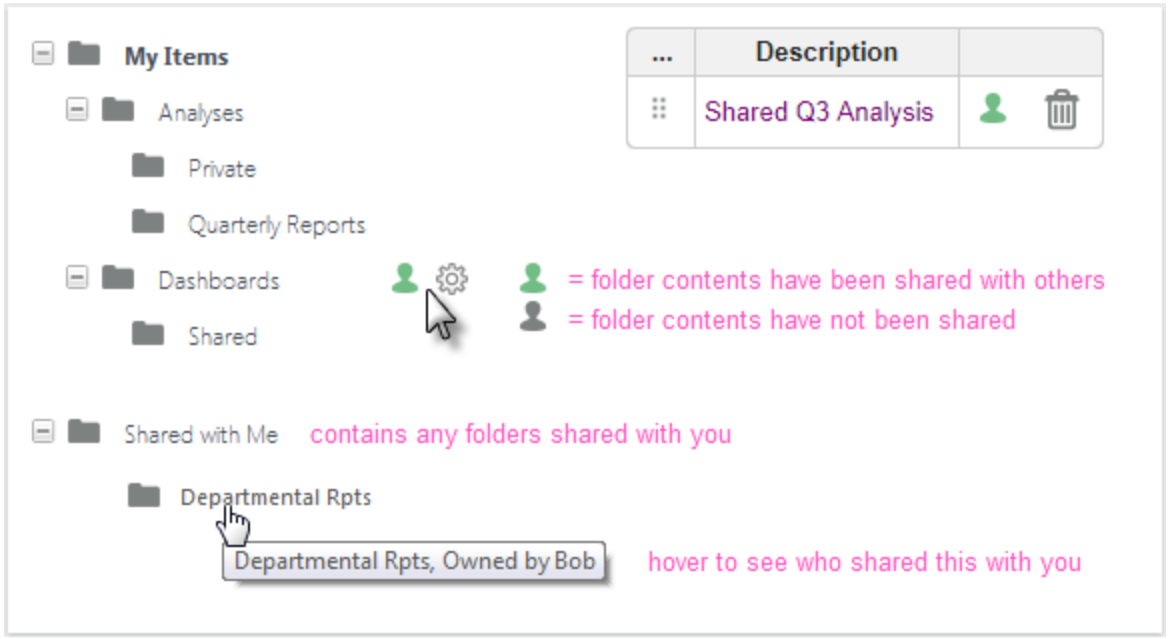
Attribute	Formula
Caption	`=IIF("@Data.IsShared~" == "True" && "@Data.BookmarkUserName~" == "@Function.UserName~", "_SupportFiles/BookmarkShared.png", "_SupportFiles/BookmarkNotShared.png")`
Tooltip	`=IIF("@Data.IsShared~" == "True" && "@Data.BookmarkUserName~" == "@Function.UserName~", "Edit Sharing", "Share")`

If you have Logi's SSRM Add-on Module installed, you may want to copy and use these icon images:

`C:\Program Files\Logi Analytics\InfoGo\_SupportFiles\rdBookmarkSharingOff.png` and `rdBoomarkSharingOn.png`

Sharing Bookmark Folders

Entire bookmark folders can also be shared with other users:

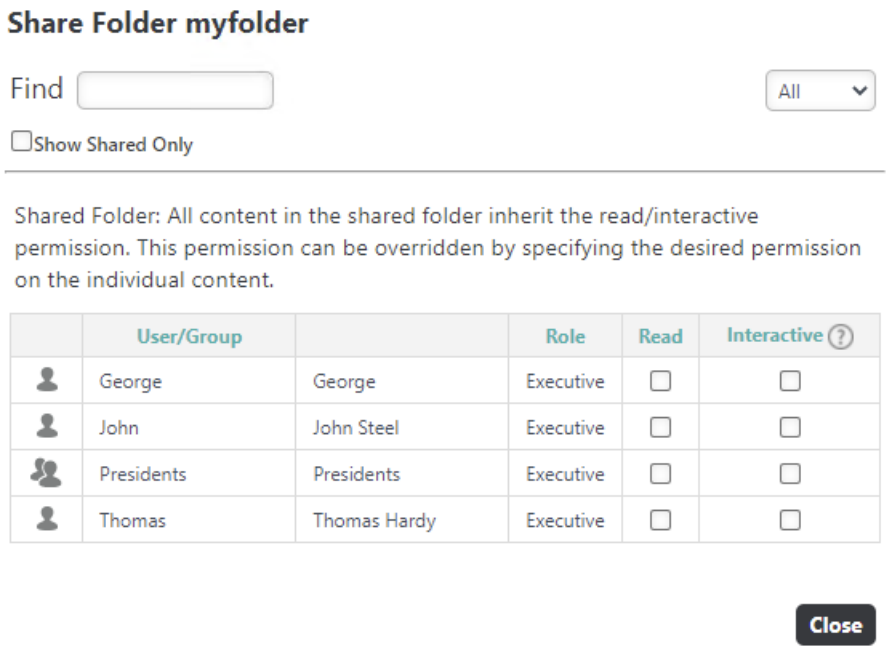


When sharing is enabled and you hover your mouse over sub-folders in the Bookmark Organizer, you'll see the built-in sharing icon, as shown above. The icon color indicates whether the folder has been shared with anyone.

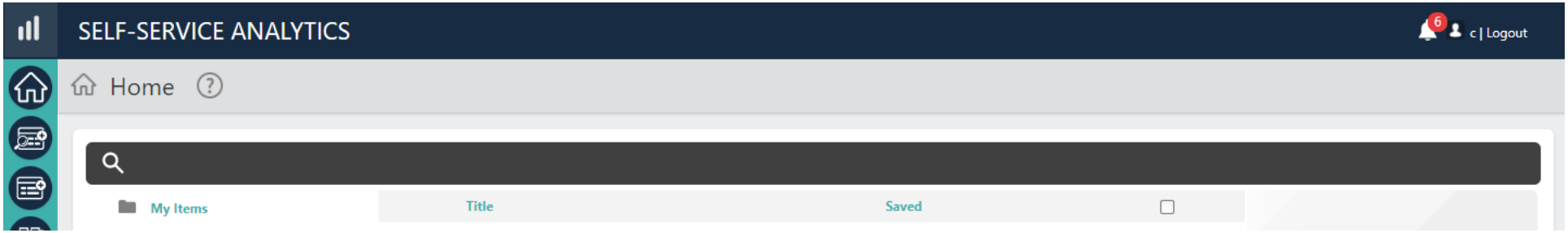
A special *Shared with Me* folder will appear containing the folders, if any, that others have shared with you. Click the **shared folders** to see their contents and run the bookmarks. You can hover your mouse over them to see who shared them with you.

Clicking a sub-folder's green or black **sharing icon** will display the same Share pop-up panel used when sharing bookmarks. In it, you can see who you've shared the folder with, share the folder with others, and revoke sharing. The *My Items* personal folder cannot be shared.

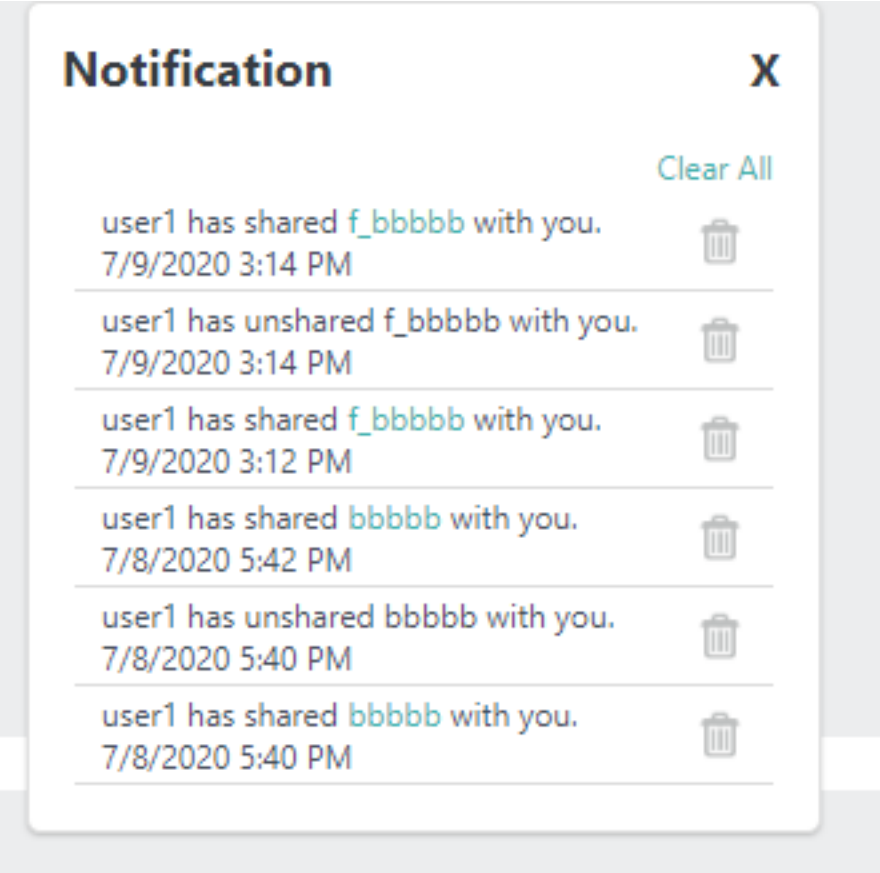
Select the **question mark** icon next to the Interactive permission to expose a permission description:



If your application has been configured for it, you will receive a notification on the SSRM Home page when an item is shared/un-shared with you:

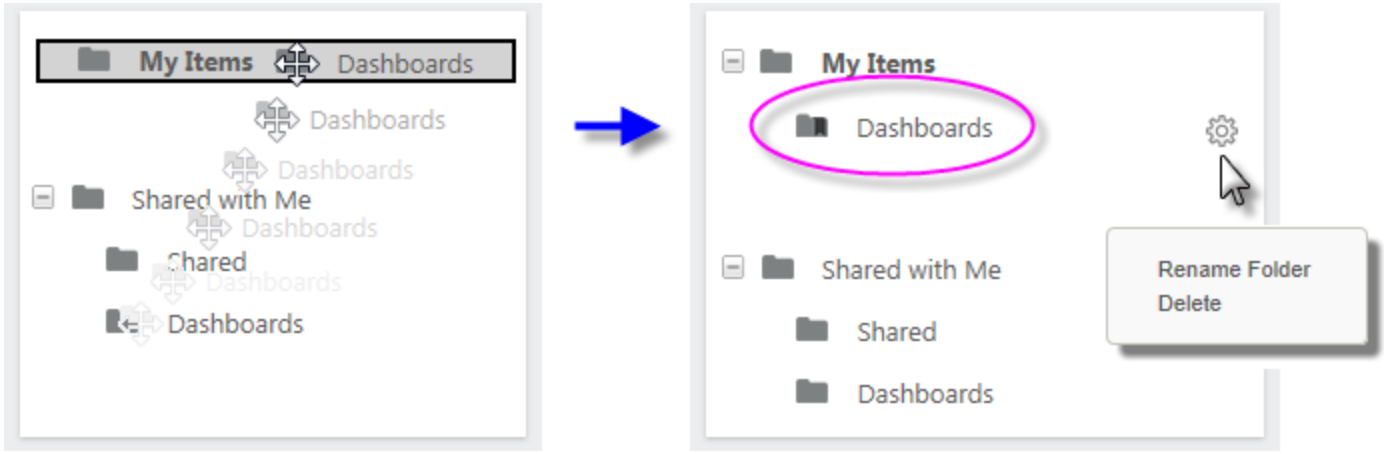


Selecting this notification will display information about the author and shared folder. Select the **shared item** to access the folder:



Shared Folder Shortcuts

Suppose a folder shared with you is many levels of sub-folders deep and it's a nuisance to have to drill-down to it every time you need to use its bookmarks. The *shared folder shortcut* can make life easier for you in this case. This is created by drilling-down to the shared folder you want and then dragging it onto one of your personal folders:



Drag a shared folder to a personal folder..

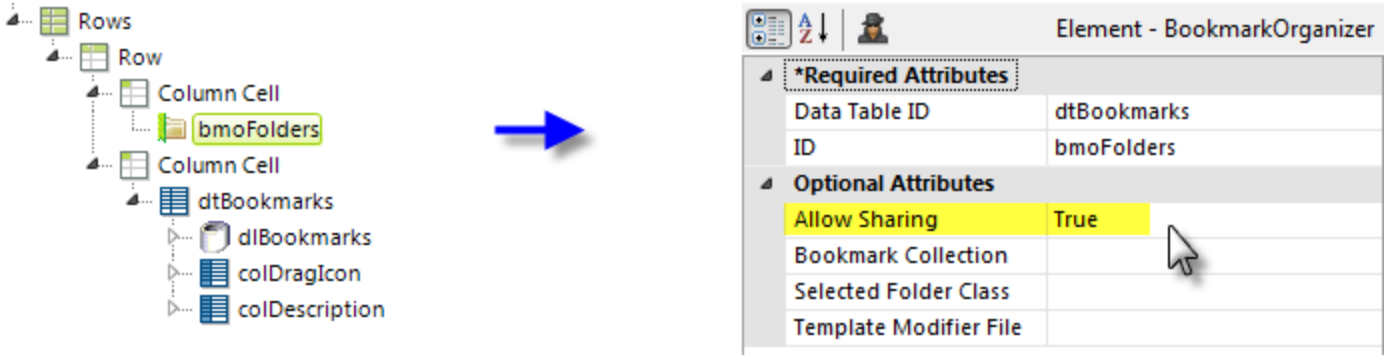
...to create a "folder shortcut"

This action creates the folder shortcut, as shown above. You can rename or remove shortcuts by clicking the **gear** icon next to the shared folder.

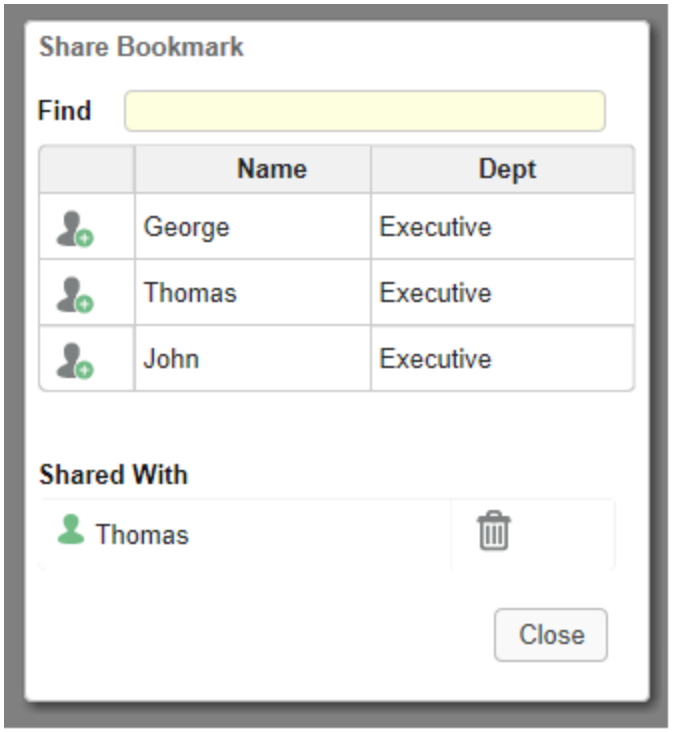
Implementation

Assuming you've already implemented the Bookmark Organizer, here's how to implement bookmark sharing:

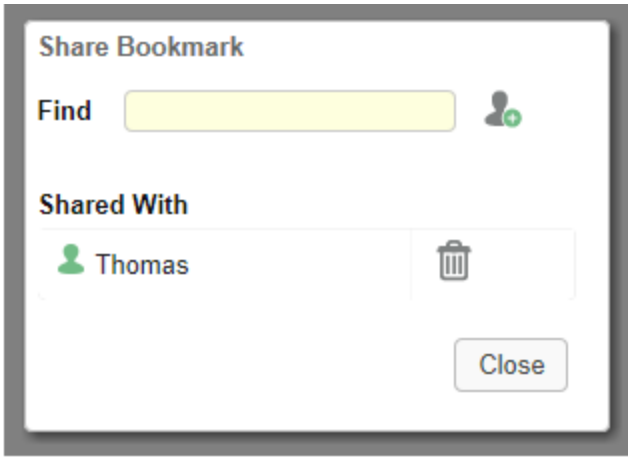
1. Ensure that some variety of **Logi Security** is being used.



2. Set your **Bookmark Organizer** element's **Allow Sharing** attribute to *True* (tokens may be used here), as shown above.



With **Sharing List**



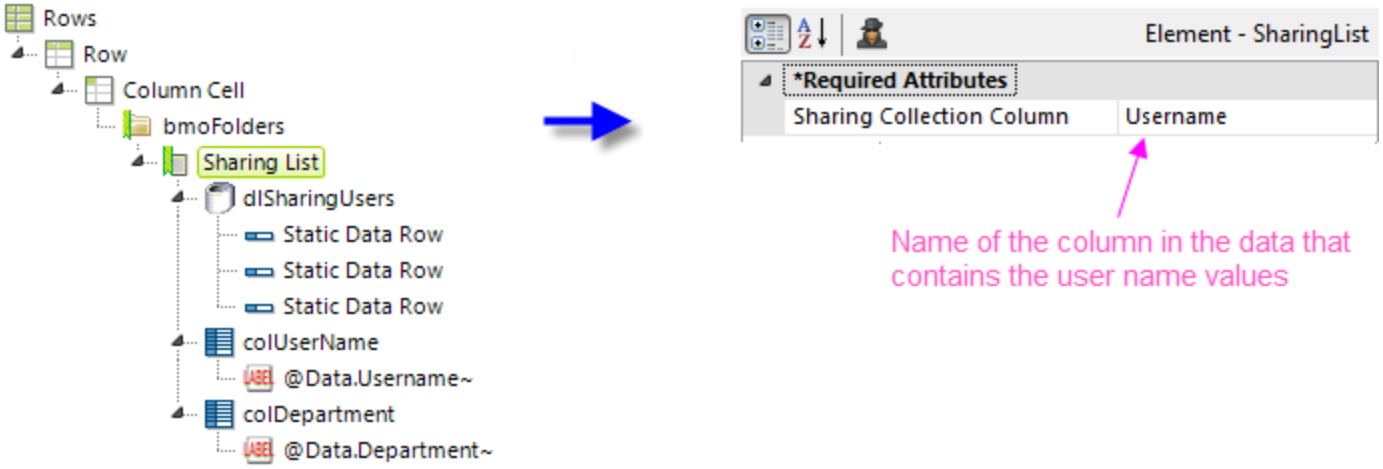
Without **Sharing List**

3. Next, decide if you want to provide users with a pick-list of other users with whom they can share folders. This is determined by the addition of the **Sharing List** element beneath Bookmark Organizer. Referring to the images shown above:

   Without the Sharing List element, user names can be typed-in and will be compared to the names associated with all of the existing personal bookmark collections. A type-ahead feature helps filter the available names.

   With the Sharing List element, user names are displayed in an embedded table, as shown above right, and the names come

from a datalayer. If a user in the table doesn't have a personal bookmark collection yet, one will be created for her automatically if a folder is shared with her.



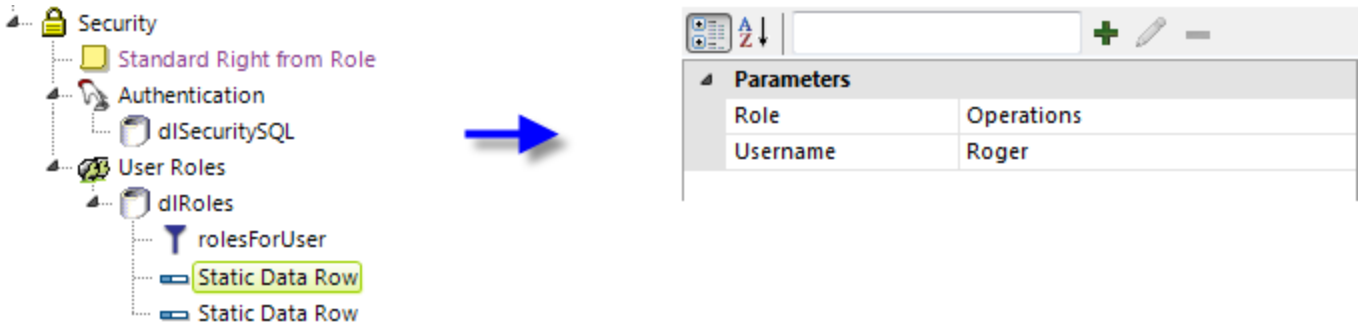
The example above shows the **Sharing List** element in use. As a special type of Data Table, it gives developers some control over the contents and formatting of the list. Any type of datalayer can be used and users are selected by clicking table rows. The Share pop-up panel will automatically expand to fit the width of the table. Specify the name of the datalayer column containing the user names in the Sharing List element's **Sharing Collection Column** attribute.

Your bookmark folder sharing functionality is now ready for use. Here's a testing tip:

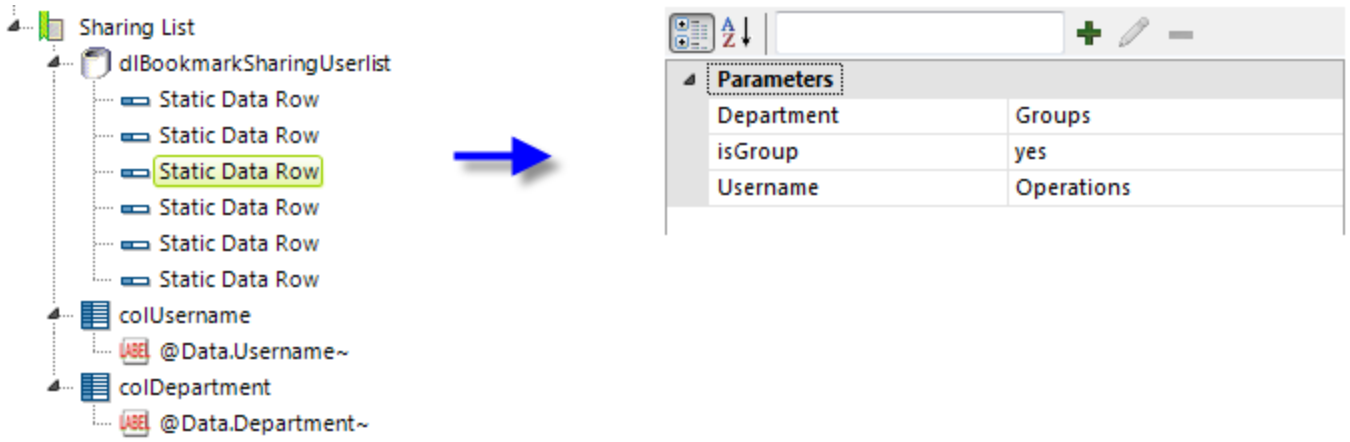
In your _Settings definition, use the **Security** element's **Development Username** attribute to run the application as different users in order to see that individual bookmark folders are created and shared correctly.

Sharing with Groups of Users

Sharing bookmarks with *groups* of users works in a manner similar to the sharing mechanism we've already seen, with two important implementation differences. First, the security configuration must, of course, identify groups and the users that belong to them.

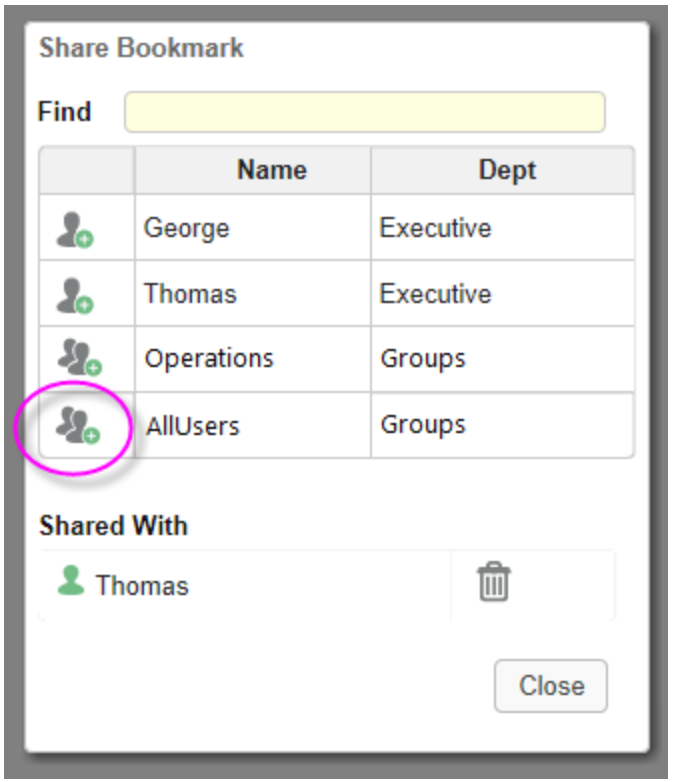


The example above shows a simple arrangement of security elements that uses a static datalayer to illustrate the assignment of roles, or "group membership", to users. It's through these elements, whether static or from other sources such as Active Directory, that a user is understood to "belong" to a group. We can see above that user "Roger" belongs to the group "Operations".



The second difference is that a Sharing List *must* be used and its configuration needs to include entries for groups, as shown above. These group names should match those in the security elements in the _Settings definition.

Your should include a column as a "this user name is really a group name" flag. When this column value is blank, the username column will be understood to be the name of an individual user; when the column has any other value, such as "yes", the username column will be understood to be a group name. The name of the group name flag column is entered into the Sharing List element's **Group Identifier Column** attribute. In the example above, that would be "isGroup".



With groups included in the Sharing List configuration, the sharing pop-up will look like the example shown above, with special icons for groups.

As you might expect, when you share a bookmark with a group, it will be available to all of the users in that group.

Creating a Bookmark Manager

If you don't want to use the Bookmark Organizer and you're not sharing bookmarks, you can easily create your own bookmark manager application with Logi Info. This topic describes how to create the Data Table shown below, which can be used to manage bookmarks.

Report	Description	Saved	Run	Actions
Analysis Grid	Avg Quantity by Category	9/25/2014 1:52:05 PM		
Analysis Grid	Stock Performance Chart	9/25/2014 1:47:40 PM		
Analysis Grid	Sum Sales by Employee	9/25/2014 11:49:56 AM		
Data Table	Quarterly Inventory Table	9/22/2014 2:02:03 PM		
Data Table	Serafield Market Orders	9/22/2014 2:01:45 PM		

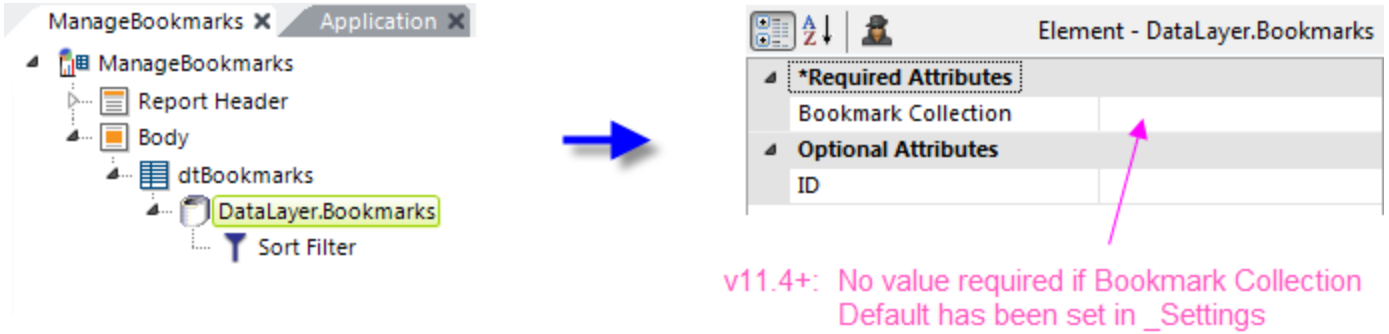
The table shown above contains these columns:

- **Report** - The name of the report.
- **Description** - Text entered when the bookmark is created to help identify it.
- **Saved** - The bookmark creation time stamp.
- **Run As** - Links that re-create the report, either by running it in a new window or by exporting it to PDF.
- **Actions** - Links that allow the description to be edited and the entire bookmark to be deleted.

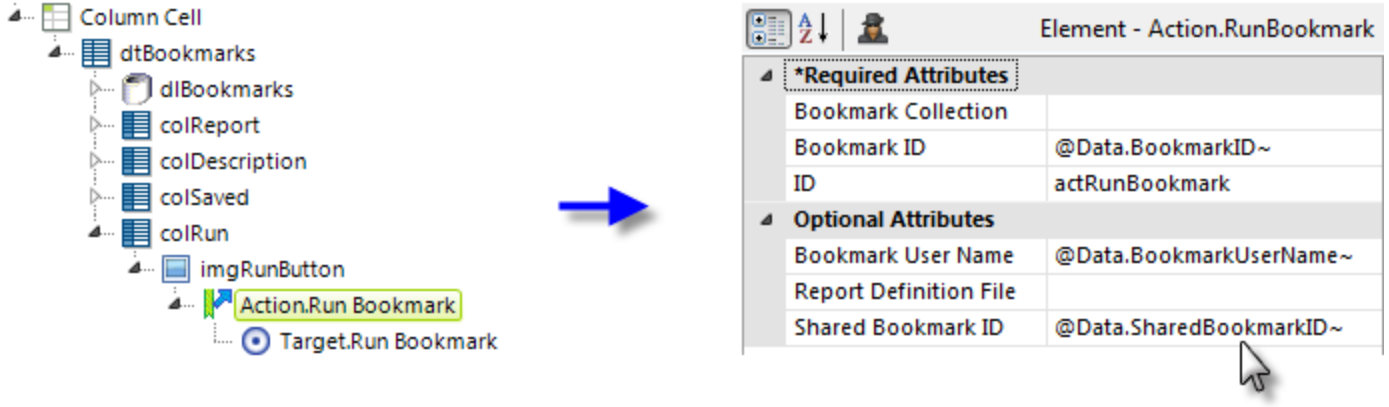
The following examples present the bookmark-related elements used to create the table. They assume that you've set the General element's **Bookmark Collection Default** attribute in the _Settings definition. If you're using an earlier version, you'll need to provide a value for the Bookmark Collection attribute in each element.

Re-run Reports with Bookmarked Values

Here's how to build your report definition to include links to re-run bookmarked reports:



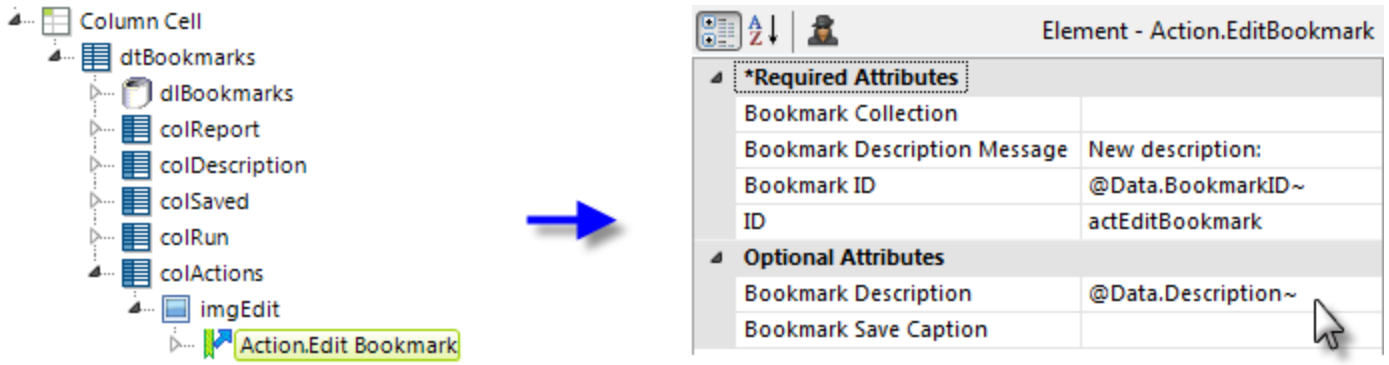
1. Add a **DataLayer.Bookmarks** element, as shown above, to retrieve the saved bookmark data. You may want to add a Sort Filter element to sort the data on the creation timestamp (the *SaveTime* column).



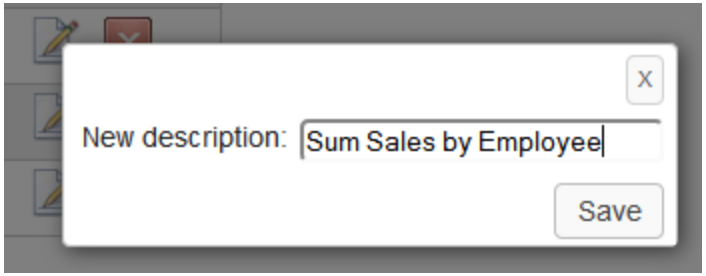
2. Add Data Table Columns, as shown above, and in order to re-run a report with bookmarked values, add an **Action.Run Bookmark** element as a link. You do not need to provide a Bookmark Collection attribute value for it if you've already configured it in _Settings. If you want to run the report in a different window, add a Target.Run Bookmark element as shown. If you want to pass parameters to the report, use a Link Parameters element as usual.

Edit the Bookmark Description

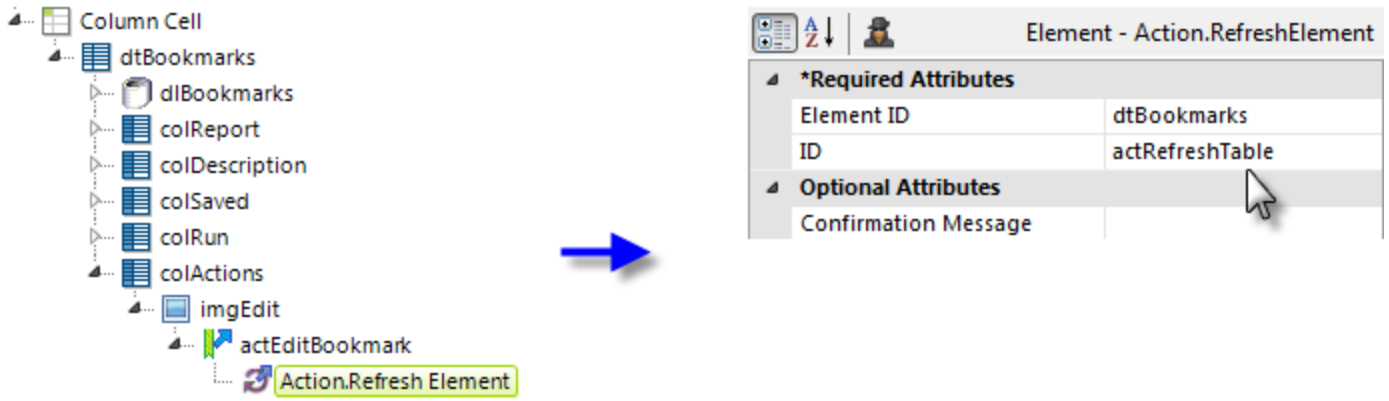
Here's how to include elements that allow bookmark descriptions to be edited:



1. Beneath the element to be used as a link, add an **Action.Edit Bookmark** element, as shown above.
2. If you set a **Bookmark Collection Default** in _Settings, leave this blank. If not, or if using earlier versions, provide the name of the bookmark collection to be edited.



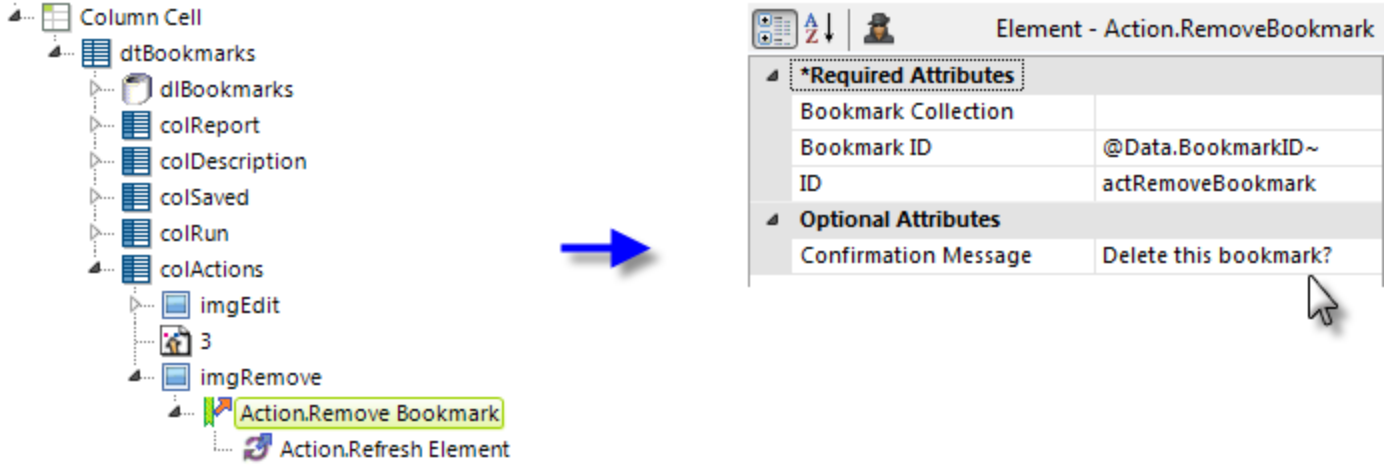
3. Set its **Bookmark Description Message** attribute to the text for the prompt in the edit box, as shown above. When the link is clicked at runtime (and this attribute has a value), a modal dialog box will open and prompt the user to enter a description, which is saved in the bookmark.
4. Set its **Bookmark ID** attribute value to the ID of the bookmark to be used. In this example, that information comes from the datalayer and is specified using an @Data token.
5. Set its **ID** attribute to a unique identifier.
6. Set its **Bookmark Description** attribute value to the current description of the bookmark to be used. In this example, that information comes from the datalayer and is specified using an @Data token.



7. Finally, to fresh the table immediately, add an **Action.Refresh Element** element beneath the Action.Edit Bookmark element, as shown above.
8. Set its **Element ID** attribute value to the ID of the Data Table.

Delete Bookmarks

Here's how to include elements that allow bookmarks to be deleted:

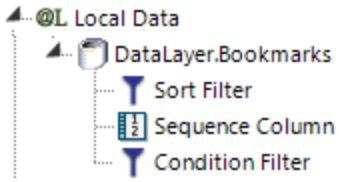


1. Beneath the element to be used as a link, add an **Action.Remove Bookmark** element, as shown above.
2. If you set a **Bookmark Collection Default** in _Settings, leave this blank. If not, or if using earlier versions, provide the name of the bookmark collection to be edited.
3. Set its **Bookmark ID** attribute value to the ID of the bookmark to be used. In this example, that information comes from the datalayer and is specified using an @Data token.
4. Set its **ID** attribute to a unique identifier.
5. Set its **Confirmation Message** attribute value to the text to be displayed when the user is prompted to confirm the deletion. If this is left blank, *no confirmationrequest* will appear.
6. As before, an optional Action.Refresh element can be added to refresh the table after the bookmark is deleted (see Steps 7 and 8 above).

Running Most-Recent Bookmark on Page Load

Some developers may want their users to run a report and have their most-recently saved bookmark applied by default.

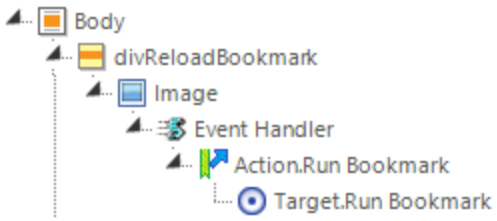
There are two steps required to make this happen: 1) run the report to get the ID for the most-recently saved bookmark, and 2) run that bookmark. Here's how to do this:



The elements shown above will let you reduce the data in the datalayer to the single row for the most-recently saved bookmark.

1. **Local Data** - This is used to get the bookmark. We'll control when it runs by setting its **Condition** attribute to the special token `"@Request.rdBookmarkID~" = ""`.
2. **DataLayer.Bookmarks** - Configure with the name of your bookmark collection or leave blank if a default collection has been specified in _Settings. A token can also be used here.
3. **Sort Filter** - Sort the bookmarks on the `SaveTime` column (*Date* type data), into *Descending* order.
4. **Sequence Column** - This adds a "row number" column to each row of sorted data and the most recent bookmark will wind up as Row #1.
5. **Condition Filter** - Configure its Condition attribute to `@Data.mySequenceColumn~ = 1` which will remove all rows except Row #1, the most recent bookmark.

Now that you've retrieved and isolated the most-recently saved bookmark data, you need to run the bookmark:



The elements shown above will reload the report with the appropriate request variables.

1. **Division** - This container is used to prevent endless recursive reloading of the report; set its **Condition** attribute to `"@Request.rdBookmarkID~" = ""`.
2. **Image** - Create and use a 1-pixel x 1-pixel transparent image (which will not be visible in the report).
3. **Event Handler** - Set this to handle the DHTML `onLoad` event.
4. **Action.Run Bookmark** attributes:
   - **Bookmark Collection** = leave this blank if you configured it in _Settings > General.
   - **Bookmark ID** = `@Local.BookmarkID~`.
   - **Report Definition File** = the name of this report definition (do not select *Current Report*).
5. **Target.Run Bookmark -** No attribute settings.

When run, your report will load, then immediately reload itself with the most-recently saved bookmark applied to it.

Storing Bookmark, Gallery, and Save Files in a Database

Bookmark, Gallery, and Save files are usually stored as XML files in the web server file system. However, this may not be useful or practical in some Logi application deployments, so it's now possible to store them instead in a SQL database.

Some of the benefits of this approach include making backups easier and eliminating the need for network shares and "sticky" sessions. Logi Studio includes a migration tool that makes it easy to transfer existing bookmark files into a database, and then use them from there.

You can store the following (which we'll collectively call "bookmark files" in this topic)...

- Bookmark Collection files
- Bookmark files
- Gallery files
- Gallery thumbnail files
- Files uploaded with Report Author

...in these databases:

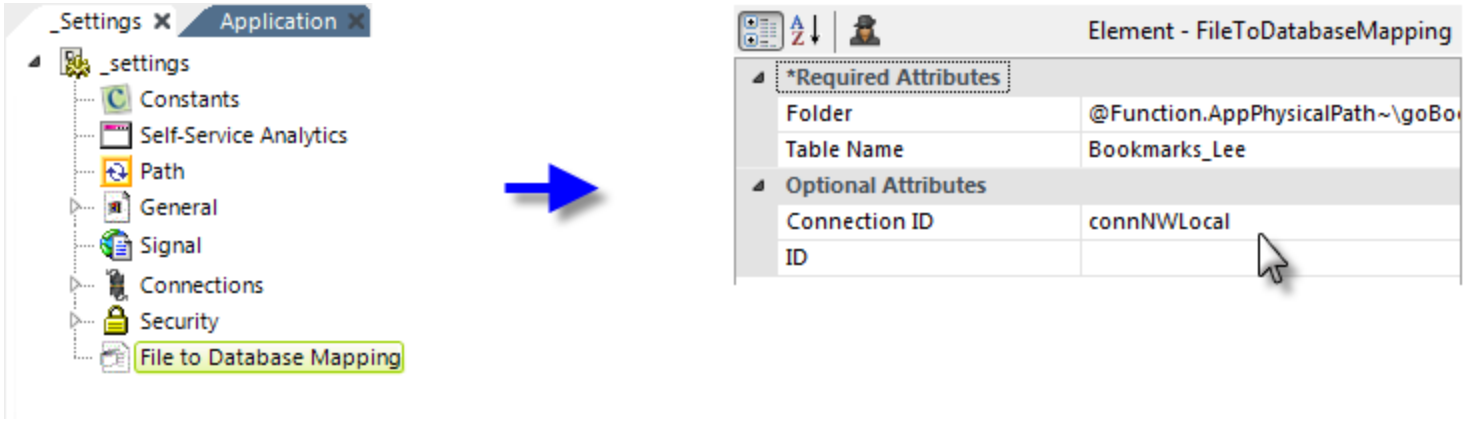
- Microsoft SQL Server 2005+
- MySQL
- Oracle
- PostgreSQL

Requirements

These are the requirements for migrating and using database bookmark storage. In your _Settings definition:

1. If one doesn't already exist, add a **Connection** element for the database where your bookmarks will be stored. *Ensure that the account used in the Connection element has sufficient rights to read from, and **write** to, the database*.
2. If not already in use, add and enable the **Security** element and enter a user name in its **Development Username** attribute. For the duration of the bookmark migration operation, grant the *rdBookmarksAdmin* right or role to that user.
3. If you're using Logi **SecureKey** security, set your Security element's **SecureKey Database Connection** attribute to the connection for your bookmark database.
4. Add a **File To Database Mapping** element under the root of your _Settings definition. When your application needs to read or save a bookmark, this element will "redirect" the command to a database table created for this purpose.

The migration takes place entirely within the context of the application currently open in Logi Studio, you can't use it to migrate files in other Logi applications.



As shown above, the Mapping element is added in the _Settings definition.

Its **Folder** attribute is set to a file folder that has been previously configured to store bookmark files. For example, to store bookmarks in the database, this attribute and the **General** element's **Bookmark Folder Location** attribute would be the same, *as*

long as you do not have any dynamic subfolders in the BookMark Folder location.. The @Function.AppPhysicalPath~ token can be used in this attribute but other tokens are not allowed.

If you DO have dynamic subfolders(such as in SSRM), then you need to specify it in this way (for example) if:

```
BookmarkLocation="@Function.AppPhysicalPath~\goBookmarks\UserName@Function.UserName~"
```

Then your mapping would look like this:

```
<FileToDatabaseMapping Folder="@Function.AppPhysicalPath~\goBookmarks"
```

The **Table Name** attribute specifies the database table to be used; if it doesn't exist, it will be created by the migration tool using this value. At a minimum, we recommend that you use a *different* table for each Logi application. Tokens cannot be used here.

The **Connection ID** specifies the database connection to be used. If left blank, the first Connection element found will be used.

The Mapping element's attributes now support tokens.

If you have multiple folders where bookmark files are stored, then add multiple File to Database Mapping elements to include them, and migrate them all to the same database table.

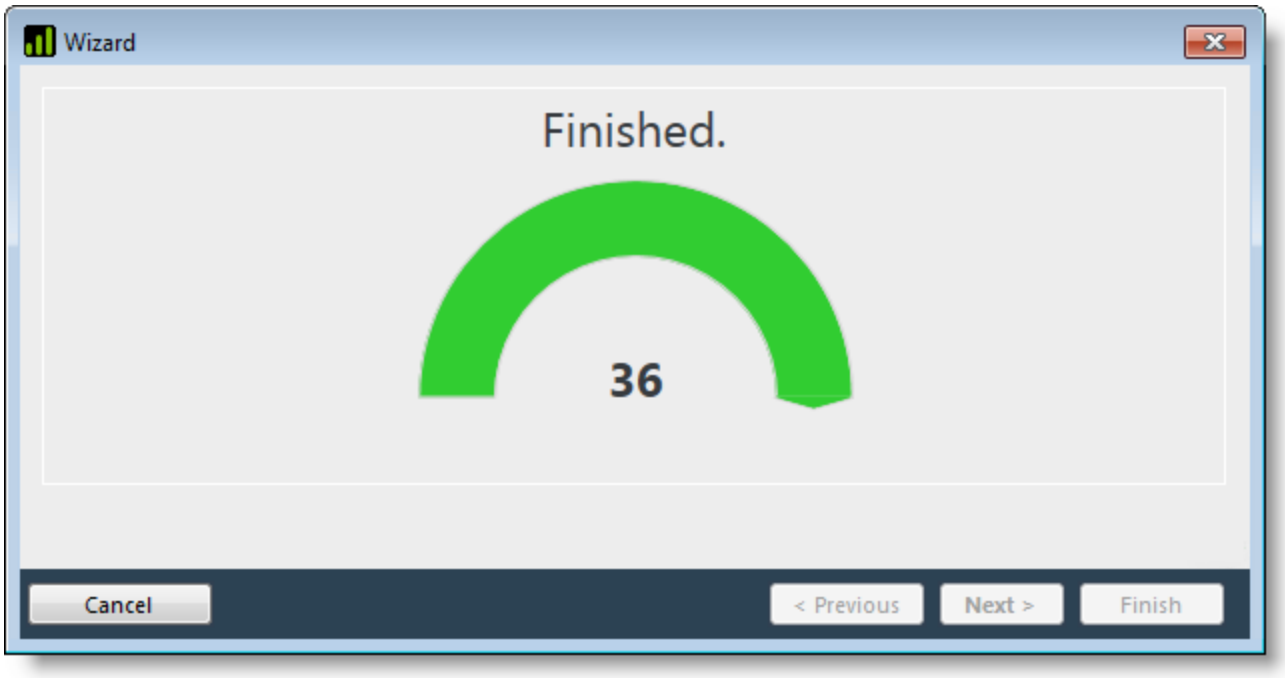
Migrating Existing Files into the Database

Once the requirements listed above have been met, you need to run Logi Studio's migration tool to create the database table and insert existing bookmark files into it. The migration of existing bookmark files into the database is something you'll do once, to create the table(s) and migrate any existing bookmark files, and won't need to do again.

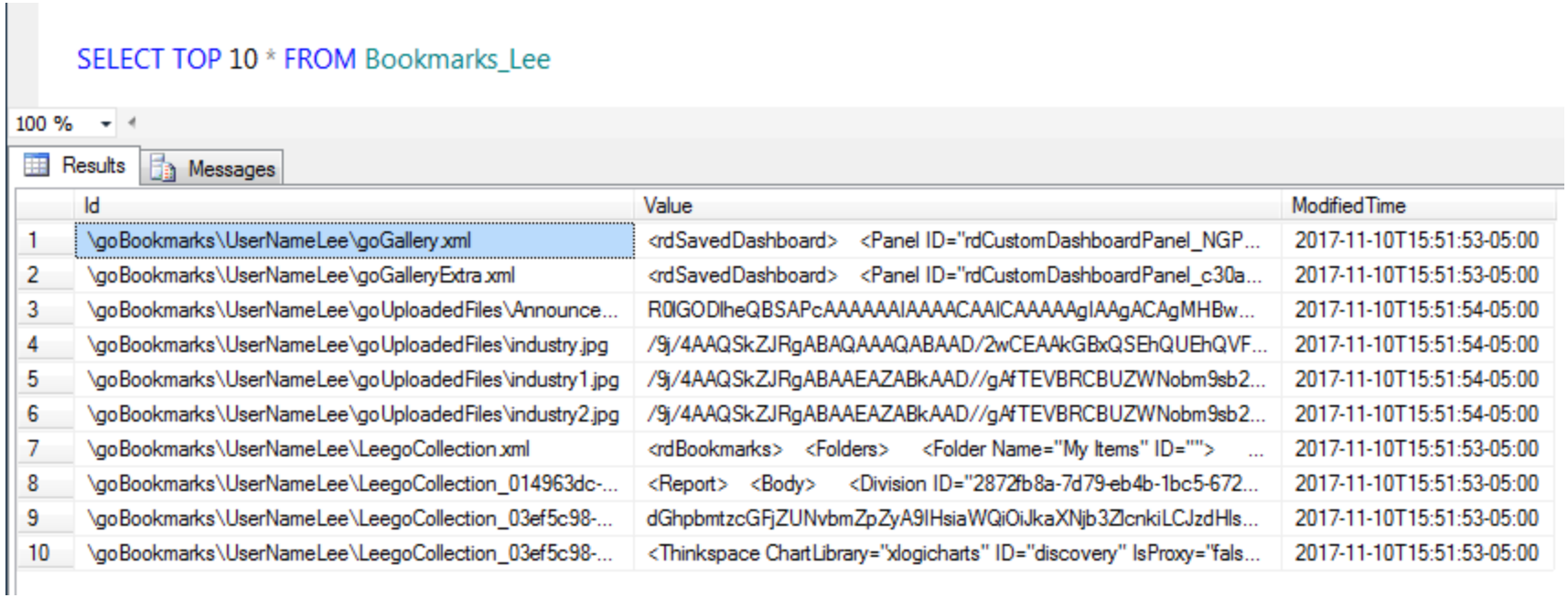
Important: Before you begin this operation, you will have to disable your Security element in the _Settings.lgx file. Find where security enabled = True, and reset it to **False**, and then save the settings file. Once you have completed the migration, set security enabled back to **True**.


The migration is a one-way operation: once bookmarks have been stored in the database and used, it's very difficult to accurately revert them back to being stored in files.

The migration tool is launched from Logi Studio's Tools menu, as shown above, by clicking the Bookmark Storage menu item. Click **Next** to start the migration.



A gauge and bookmark item counter will display progress as the migration proceeds. Click **Next**, then **Finish** when the migration is complete.



	Id	Value	ModifiedTime
1	\goBookmarks\UserNameLee\goGallery.xml	<rdSavedDashboard> <Panel ID="rdCustomDashboardPanel_NGP...	2017-11-10T15:51:53-05:00
2	\goBookmarks\UserNameLee\goGalleryExtra.xml	<rdSavedDashboard> <Panel ID="rdCustomDashboardPanel_c30a...	2017-11-10T15:51:53-05:00
3	\goBookmarks\UserNameLee\goUploadedFiles\Announce...	R0lGODlheQBSAPcAAAAAAIAAAACAAICAAAAAgIAAgACAgMHBw...	2017-11-10T15:51:54-05:00
4	\goBookmarks\UserNameLee\goUploadedFiles\industry.jpg	/9j/4AAQSkZJRgABAQAAAQABAAD/2wCEAAkGBxQSEhQUEhQVF...	2017-11-10T15:51:54-05:00
5	\goBookmarks\UserNameLee\goUploadedFiles\industry1.jpg	/9j/4AAQSkZJRgABAAEAZABkAAD//gAfTEVBRCBUZWNobm9sb2...	2017-11-10T15:51:54-05:00
6	\goBookmarks\UserNameLee\goUploadedFiles\industry2.jpg	/9j/4AAQSkZJRgABAAEAZABkAAD//gAfTEVBRCBUZWNobm9sb2...	2017-11-10T15:51:54-05:00
7	\goBookmarks\UserNameLee\LeegoCollection.xml	<rdBookmarks> <Folders> <Folder Name="My Items" ID=""> ...	2017-11-10T15:51:53-05:00
8	\goBookmarks\UserNameLee\LeegoCollection_014963dc-...	<Report> <Body> <Division ID="2872fb8a-7d79-eb4b-1bc5-672...	2017-11-10T15:51:53-05:00
9	\goBookmarks\UserNameLee\LeegoCollection_03ef5c98-...	dGhpbmtzcGFjZUNvbmZpZyA9IHsiaWQiOiJkaXNjb3ZlcnkiLCJzdHls...	2017-11-10T15:51:53-05:00
10	\goBookmarks\UserNameLee\LeegoCollection_03ef5c98-...	<Thinkspace ChartLibrary="xlogicharts" ID="discovery" IsProxy="fals...	2017-11-10T15:51:53-05:00

If you examine your database table, as shown above, you'll see that the files have been copied into it.

If it's not otherwise useful, delete the user name you set in the Security element's **Development Username** attribute and remove the *rdBookmarksAdmin* right/role assignment.

Migration tool can be run without using Studio, using `http://<Logi App URI/URL>/rdPage.aspx?rdDebugSession=True&rdForWizard=True&rdReport=rdTemplate/rdMigration` URL pattern. Note that same rules for Security, as defined for Studio driven wizard, apply. Please contact Customer Service for additional information.

Subsequent new bookmarks and changes to existing bookmarks will now be automatically applied to the table. Requests for the bookmark data will result in a query of the table. Bookmark files are *not* automatically deleted by the migration process, but you may delete them manually if you'd like, as they'll no longer be read or updated.

It's not recommended but if you do re-run the migration tool later, it will update all of the existing table rows with whatever is in the (possibly "stale") bookmark files and then add any new files. This could result in overwriting newer bookmark table data with the old bookmark file data.

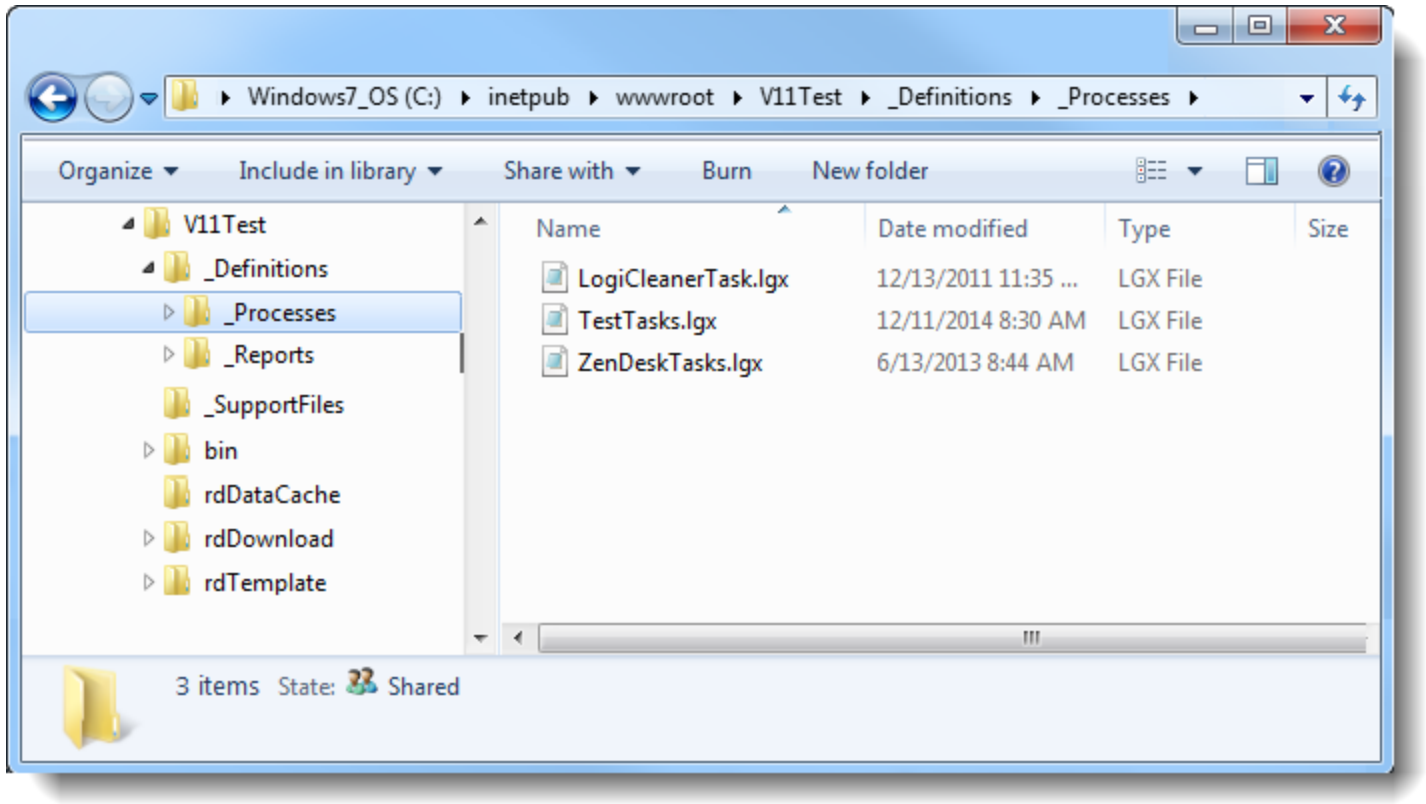
Process Tasks

Logi Info includes **Process** definitions, a special type of definition that lets you create subroutine-like tasks that can include conditional processing, branching, error handling, and other advanced features.

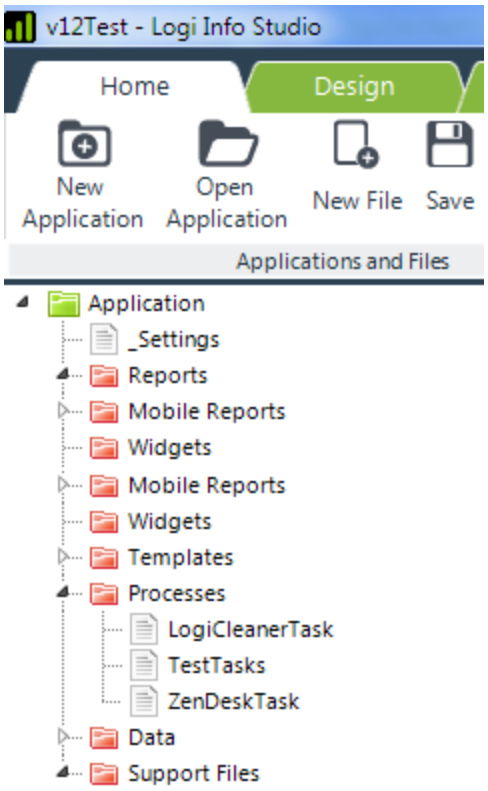
The following topics discuss Process Definitions and Tasks:

About Process Definitions and Tasks

Process definitions are similar to Report definitions and they're created and managed in Logi Studio in the same way. They contain element groups called **Tasks,** which are created in Logi Studio using elements in a manner similar to creating a Report definition. Unlike Report definitions, tasks run at the *web server* and don't present anything visual in the browser.

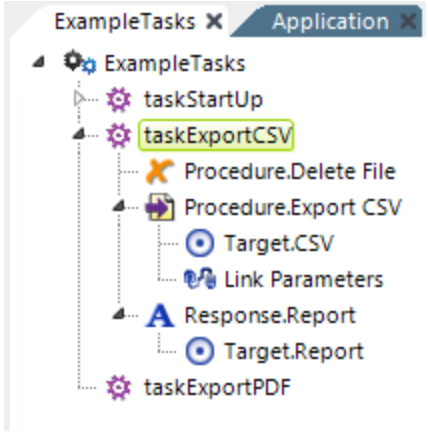


In the file system, Process definition files are physically stored in your application folder, as shown above, in the _Definitions → _Processes folder. They're .lgx type files and use the same XML source code found in Report definition files.



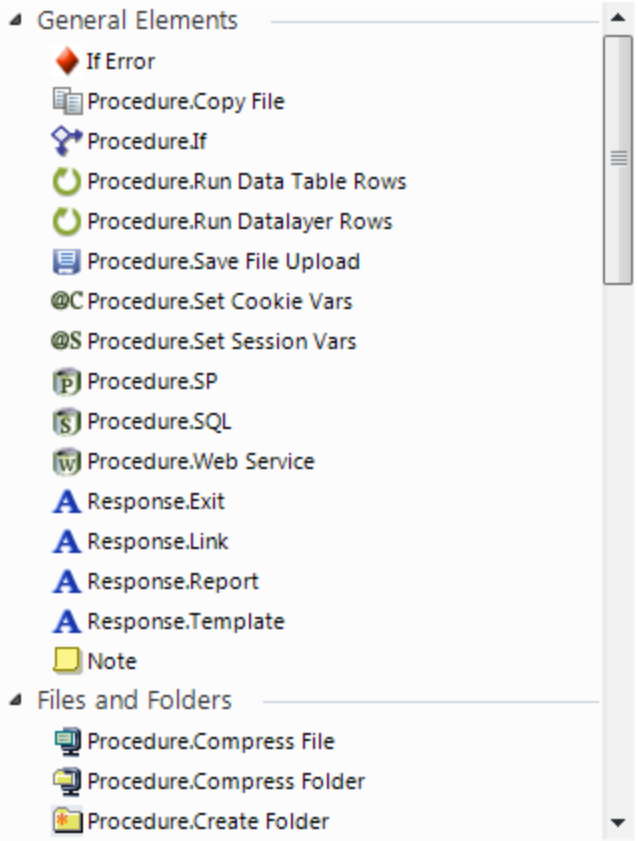
In Logi Studio, Process definitions appear in the Application panel listed under the Processes folder, as shown above. You can add a new process definition by selecting and right-clicking the Processes folder and manage them in Studio like any other definition.

Each Process definition file contains one or more tasks. So, you may decide to use *multiple* Process definition files in your application. You may prefer to put all of your tasks in a single definition file, or to group tasks, by function for example, in separate Process definition files. The choice is to yours; the Logi Server Engine will include all Process definition files when it runs the application.



Tasks are created in a Process definition by adding a **Task** element to the definition root. The example shown above includes three tasks.

The top-to-bottom order of Task elements in a Process definition does not matter (a suggestion: alphabetical order makes them easier to find). However, *within* a task the position of its child elements *is* important because they're processed in a top-to-bottom sequence.



The example above shows some of the elements available for use in a task; they can perform a variety of operations. Some look similar to those used in Report definitions while other are unique to Procedure definitions.

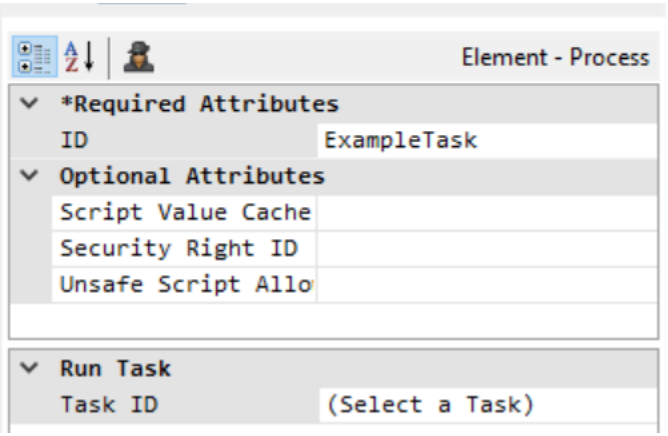
Tasks can accomplish many things that cannot be done in a Report definition. For example,

- Direct interaction with the web server file system (to save, copy, or delete folders and files)
- Conditional processing using If-Then branching of processing flow
- Row-by-row processing of datalayer or Data Table contents
- "Write back" to databases, to insert and update records

- Generic and error-specific error handling (including SQL errors)
- Send email, SMS messages, and Twitter tweets
- Save file uploads and the state of super-elements
- Set and clear client-side cookies
- Create and manage Logi Scheduler tasks
- Terminate server session

and they can do many of the same things, such as exports, plug-in calls, database operations, and more.

When using Logi Security, the Task element's **Security Right ID** attribute, shown below, provides greater security control over the running of tasks.



The **Unsafe Script Allowed** attribute, pictured above, controls whether the Procedure.Script element throws an 'Unsafe class reference' error when accessing the file system object. To overcome this error when running a procedure, set the Unsafe Script Allowed attribute to '*True*'. For more information about scripting objects and functions, see *Work with Scripting*.

Tasks are essential to use of the **Logi Scheduler**: when the Scheduler runs a scheduled event, it calls a Process task, which then runs the desired report and manages its output. For more information about this, see *Logi Scheduler*.

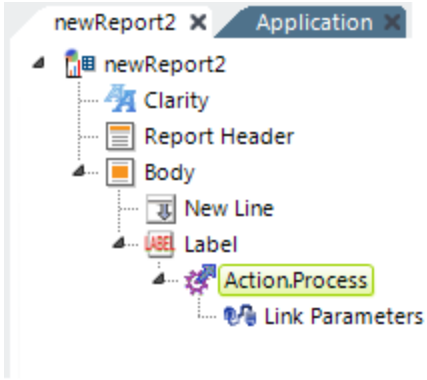
You can use a Default Request Parameters element in a Process definition, as you would in a Report definition, and you can also use it within individual tasks. For more information, see *Using Request Parameters*.
Process definitions and tasks extend the capabilities of Logi Info apps in many ways, and provide with tremendous functionality. The remainder of this topic provides examples of some of these capabilities in use.

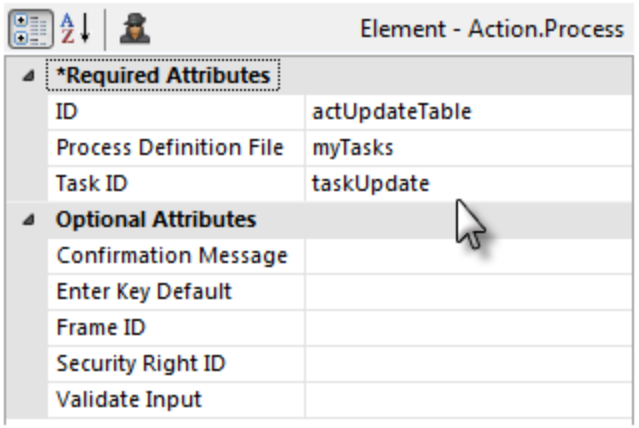
Calling and Completing a Task

In order to use a task, you must *call* it, and when it completes, it needs to *redirect* processing to someplace else.

Calling a Task from a Report Definition

As Logi report developers, we're used to "calling" one report definition from another report definition, using links, buttons, or events. When doing that, we can pass information to the next report using **Link Parameters** and we know that **User Input** element values are POST'ed to the next report as Request variables. The same is true when calling a Process task from a report.



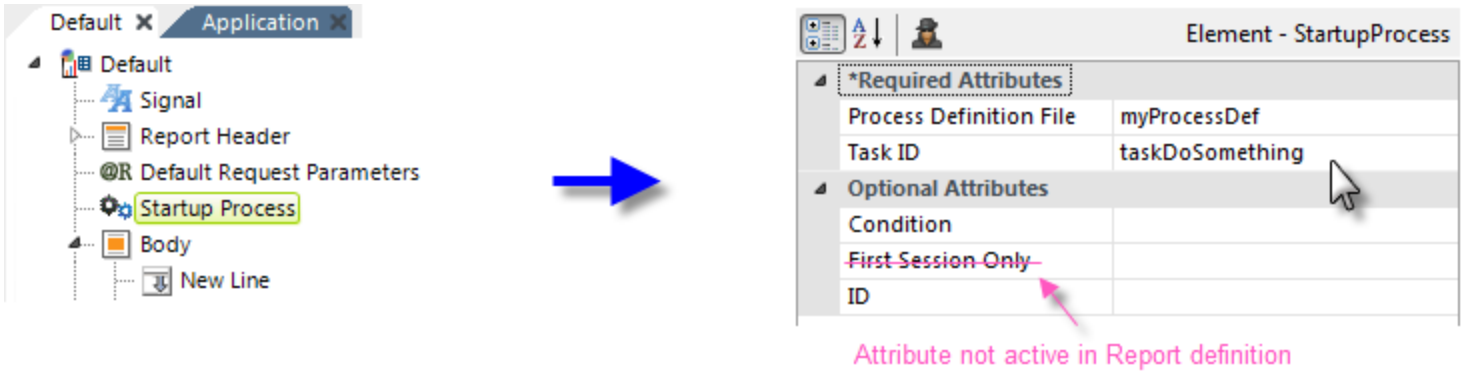


In the Report definition example shown above, an **Action.Process** element is used beneath a Label element to redirect processing to a task. The Action element's attributes identify the **Process Definition File** and the **Task ID** of the task to be called; these values can be selected from drop-down lists. A **Link Parameters** element can also be used beneath the Action element, if desired, to pass information to the task.

When processing is transferred to the task, the Link Parameters and any User Input element values will be available in the task as Request variables and can be used there via @Request tokens.

Calling a Task when a Report Loads

You can have a task run *automatically* when a Report definition loads. This "onLoad event" behavior allows you to set variables, run procedures, and accomplish any other initializations you might want to do, and then redirect the browser back to the report definition.



As shown above, the **Startup Process** element is added to a Report definition, and its attributes are set to identify the desired Process Definition and Task ID for the "startup task".

Any Request variables *in the query string* used to call the report definition will automatically be available as Request tokens in the Process task called with this element. For example, the `@Request.rdReport~` token will contain the name of the report definition specified in the URL. Note that any parameters set using **Default Request Parameters** element *are not* passed to the task.

You can use a **Response**element at the end of the startup task to redirect the browser to a specific report definition after the task completes but this is *optional*. Without one, the browser will be automatically redirected back to the original report definition.

The element's **Condition** attribute can be used to dynamically control when the Startup Process will be called.

Multiple Startup Process elements may be used to call different process tasks; they'll be run sequentially, one after the other.

Calling a Task from a URL

You can also call a process task directly from a URL, using the following syntax:

```
http://www.myReportSite.com/myLogiApp/rdProcess.aspx?rdProcess=myTasks&rdTaskID=taskUpdate&myParam=1
```

where *rdProcess* and *rdTaskID* are required and additional request variables, such as *myParam*, are optional.

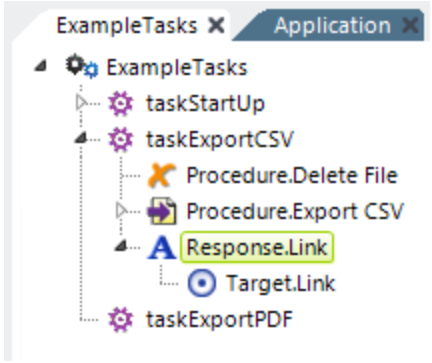
This URL can be used, for example, with an **Action.Link** element, or even from an external application.

Calling a Task from Another Task

Under certain circumstances, you may want to transfer operations from one task directly to another task.

The **Procedure.Process Task** element lets you directly call another task, in the same or in a different Process definition. A child Link Parameters element can be used to pass values to the target task. This does *not* operate like a function - the designated task will be executed but operations will not "return" to the first task afterwards.

You can also use a more "manual" approach, with the Response.Link element:



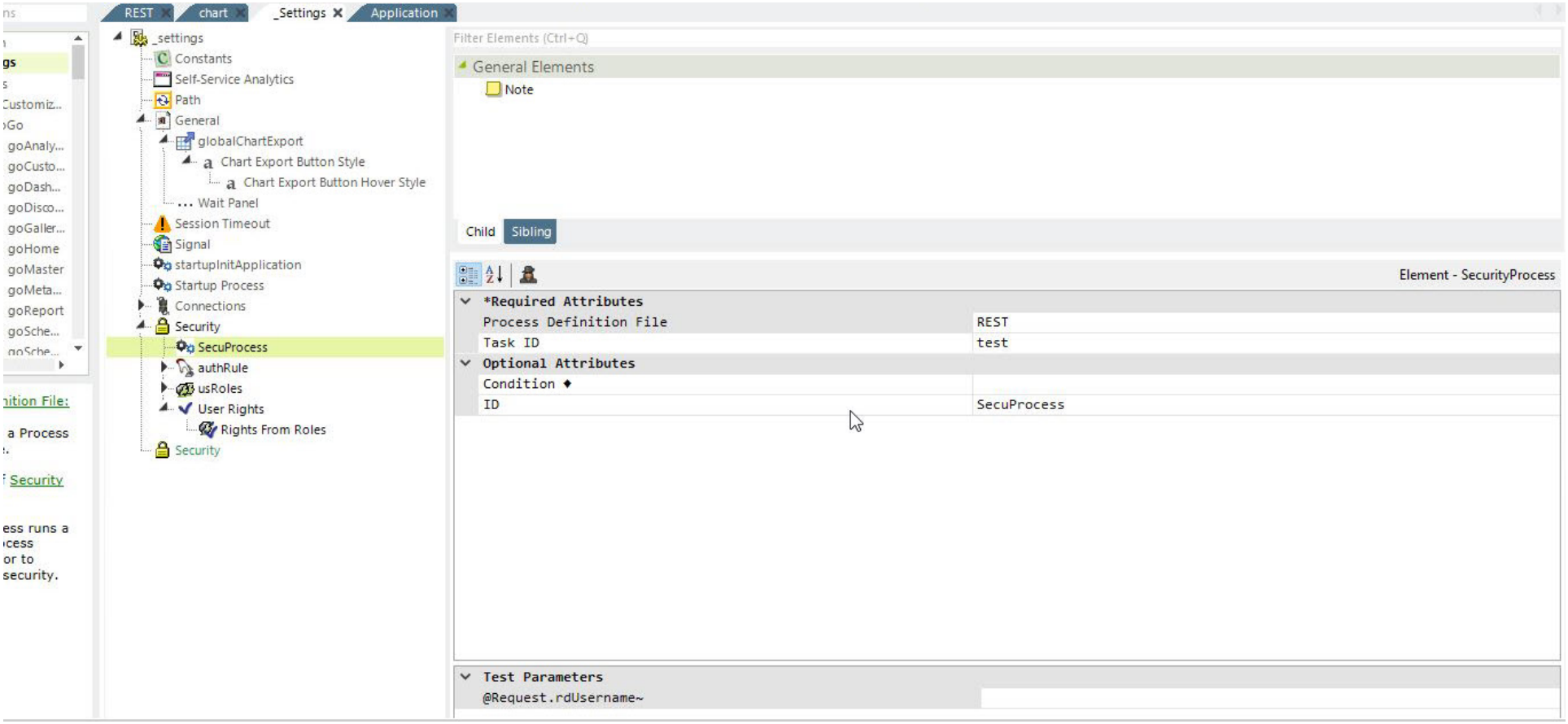
In this case, you would use a **Response.Link** element, as shown above and, in its **Target.Link** child element's **URL** attribute, use the syntax shown in the previous section for calling a task from a URL. You can use @Request tokens right in the URL to forward request variables sent to the first task, so they'll be available in the second task, like this:

```
http://www.mySite.com/myApp/rdProcess.aspx?rdProcess=myTasks&rdTaskID=taskNext&myVar=@Request.myVar~
```

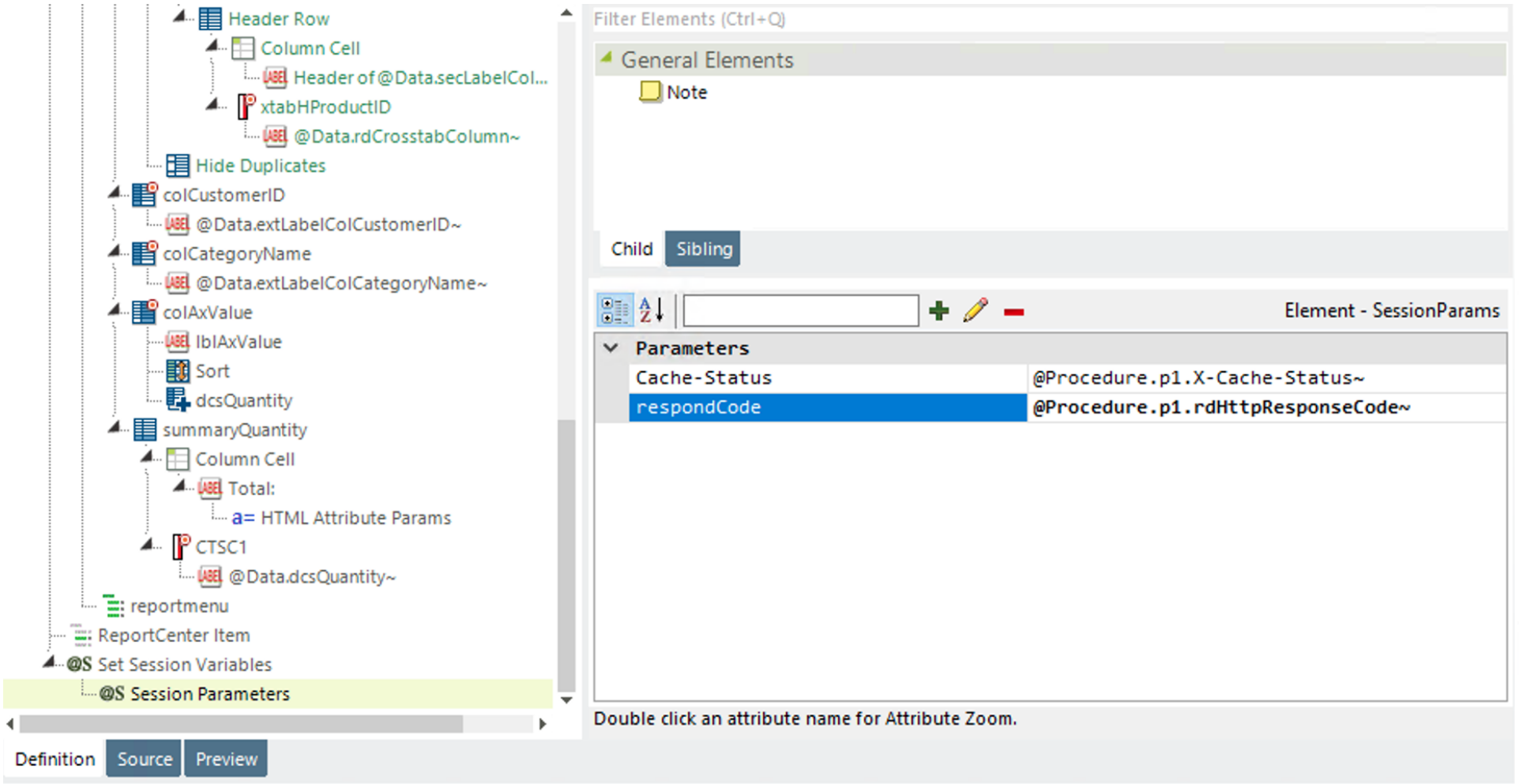
Recursion and nesting do not exist for tasks; one task just leads to another and the last task needs to handle redirection to a visible web page.

Calling a Task for Authentication

The Procedure.REST element now allows all subsequently called Task elements to reference response headers from the REST call. Additionally, the SecurityProcess element allows a Task to be called during authentication. Using these two new features together enables you to leverage response headers from a REST call to aid in authenticating a user at login time. A new child element, SecurityProcess, was added to the Security element. Now, the TaskID of Process.REST will be executed before the other Security element.

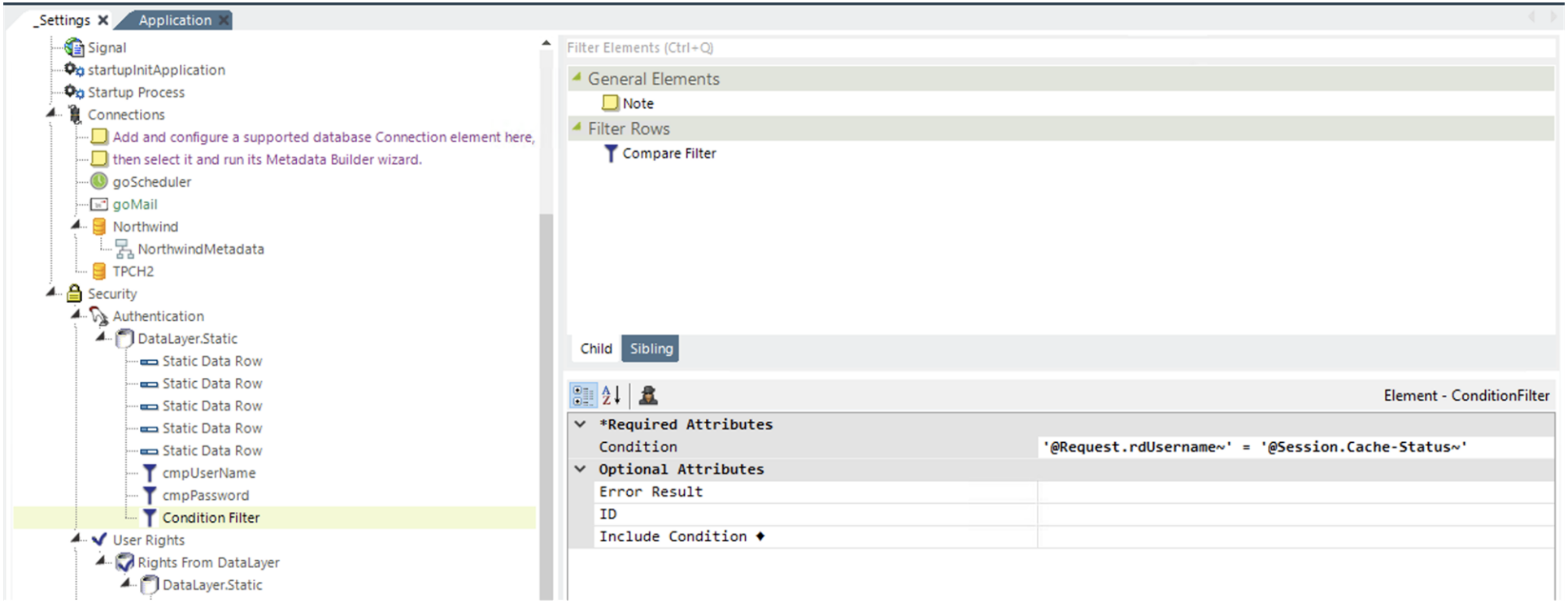


Set the Session Variable in the procedure and add a Condition Filter under the Authentication element in the _Settings file:



In our example, we used X-Cache-Status as the Session Variable in the REST process; but you can use whichever Session Variable or token fits your needs.

The Condition Filter will check whether the input username equals the session variable Cache Status; if not, the authentication will not pass.



The SecurityProcess element works exactly like the StartupProcess, except for *where* it is placed and *when* it runs. It is placed in the _Settings.lgx file as a child of the Security element, and it runs when the Security element is being processed. By adding these two new features, you are able to leverage response headers from a REST call to aid in authenticating a user at login time.

Here's an example of the code for your _Settings.lgx file:

1. In _Settings, add below connection:

```
<Connection

ID="connMusicBrainz"
```

```
Type="REST"

UrlHost="http://musicbrainz.org/ws/2"

>

<RequestHeader

RequestHeaderName="User-Agent"

RequestHeaderValue="SampleWebServer/12.5 SP2 ( devnet@logianalytics.com )"

/>

</Connection>
```

2. Then, add a Process named REST:
3. In _Settings, under the Security element, add SecurityProcess element, like below:

```
<SecurityProcess

ID="SecurityProcess1"

Process="REST"

TaskID="test"

/>
```

4. Next, in _Settings, add a user named "STALE" in AuthenticationRule, with a ConditionFilter using @Session.Cache-Status~ :

```
<AuthenticationRule

ID="authRule"

UsernameDataColumn="UserName"

>

<DataLayer

Type="Static"

>

<StaticDataRow

Password="password"

UserName="admin"

/>

<StaticDataRow

Password=""

UserName="STALE"
```

```
/>

<ConditionFilter

Condition="&apos;@Data.UserName~&apos; = &apos;@Session.Cache-Status~&apos;"

/>

</DataLayer>

</AuthenticationRule>
```

5. Add roles for user "STALE" in UserRoles, using the ConditionFilter @Session.Cache-Status~ :

```
<UserRoles

ID="usRoles"

RolesDataColumn="Roles"

>

<DataLayer

Type="Static"

>

<StaticDataRow
```

```
Roles-
="a-
dmin,endUser-
,InfoGoScheduleManager,InfoGoThemeManager,rdMetadataAdmin,InfoGoReportManager,InfoGoDataManager,rdBookmarksAdmin"

UserName="admin"

/>

<StaticDataRow

Roles="endUser,InfoGoReportManager"

UserName="STALE"

/>

<ConditionFilter

Condition="&apos;@Data.UserName~&apos; = &apos;@Session.Cache-Status~&apos;"

/>

</DataLayer>

</UserRoles>
```

6. Last, create a report:

```
<?xml version="1.0" encoding="utf-8"?>
```

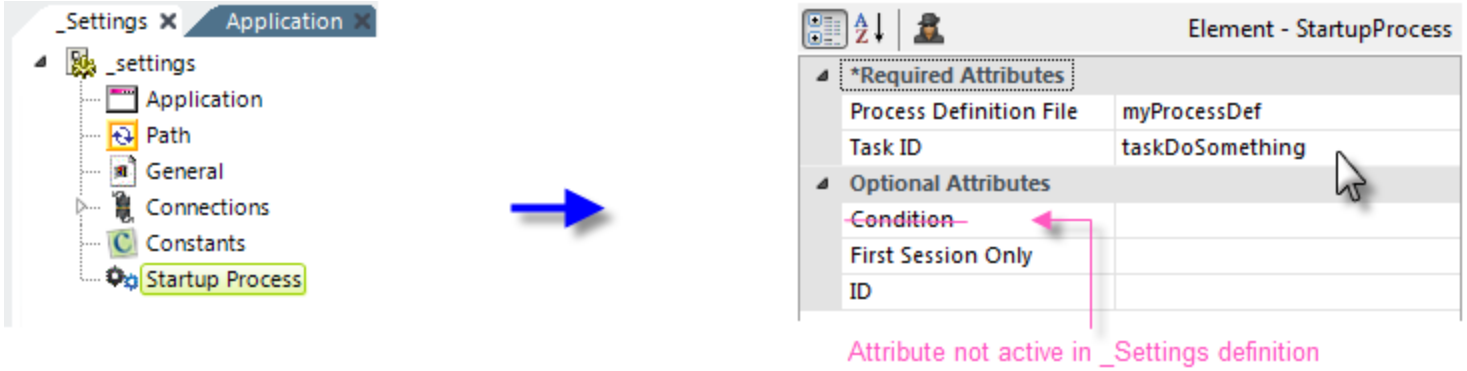
```
<Report

ID="RESTTest"

>

<Body>

<Label

Caption="return: "

/>

<Label

Caption="@Session.respondCode~"

/>

<Label

Caption="@Session.Cache-Status~"

ID="test"

/>

</Body>
```

```
<ideTestParams/>

</Report>
```

7. Run report with http://localhost/rdWeb1/rdPage.aspx?rdReport=RESTTest
   - Login as username: admin
   - You will get an "Invalid Username or Password"
   - Login as username: STALE
   - Report displays content, as below:
   - Return: 200STALE

Calling a Task at Session Start

You can have a task run *automatically* when a user session begins (i.e., when the application is browsed by a user for the first time). This allows you to set variables, run procedures, and accomplish any other "startup" work you might want to do, and then redirect the browser to a report definition.



As shown above, the **Startup Process** element is added to the _Settings definition, and its attributes are set to identify the desired Process Definition and Task ID for the "startup task".

Any Request variables in the query string used to call the application will automatically be available as Request tokens in the Process task called with this element. For example, the `@Request.rdReport~` token will contain the name of the report definition specified in the URL. Note that any parameters set using **Default Request Parameters** element *are not* passed to the task.

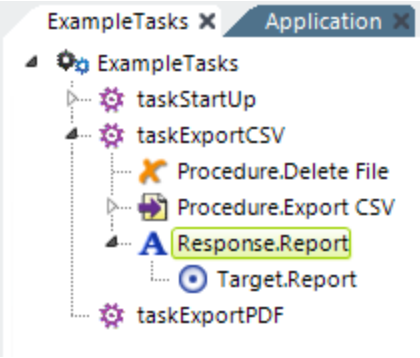
You can use a **Response** element at the end of the startup task to redirect the browser to a specific report definition after the task completes but this is *optional*. Without one, the browser will be automatically redirected to the application's default report definition.

The element's **First Session Only** attribute can be used to prevent the Startup Process from being called if the application is called recursively.

Multiple Startup Process elements may be used to call different process tasks; they'll be run sequentially, one after the other.

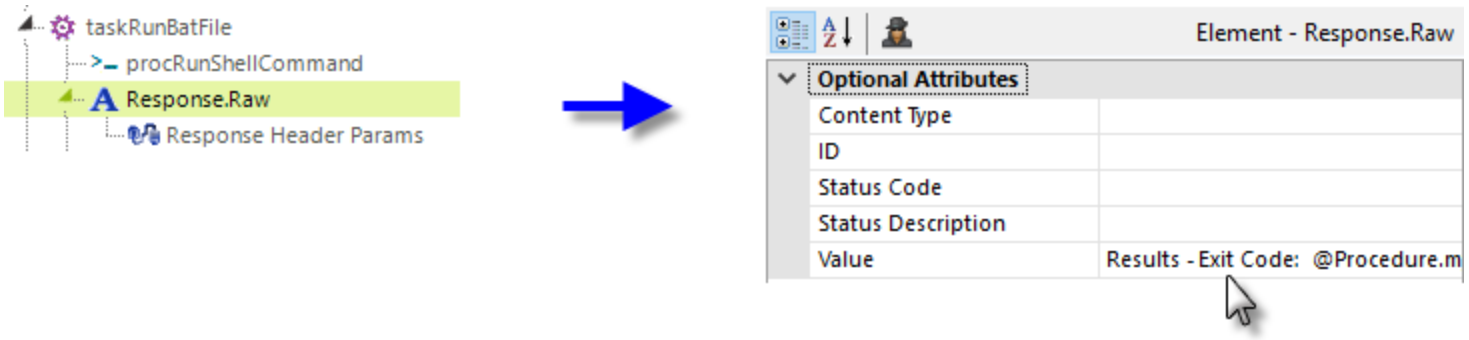
Completing a Task

When processing is transferred to a task and it completes, there is no "stack" to unwind, no built-in return path back to the calling Report definition. As the developer, you must explicitly transfer processing somewhere else after a task completes. If you don't, your user will be left looking at a blank browser screen!



Tasks use **Response** elements, as shown above, which end a task by redirecting processing, usually back to a Report definition. You can also redirect instead to another task, as discussed earlier. You should ultimately redirect to a visible page so the user sees something in their browser.

The **Response.Raw** element can be used to complete a task and gives you full control of the response. This can be especially useful when building a REST API call from a Process. Response.Raw responds with a value and optional response headers.



In the example above, the Response.Raw element is used to provide response values using @Procedure tokens. Its **Response Header Params** child element can be used to provide custom response headers and values. The Response.Raw element's attributes are:

- **Content Type** - If left blank, the default content type is *text/plain; charset=utf-8*
- **Status Code** - The HTTP response status code; if left blank, the default value is *200*
- **Status Description** - The HTTP response status description; if left blank, the default value is *OK*
- **Value** - The response value; @Procedure tokens may be used here.

If-Then Branching in a Task

Data validation provides a fine example of *conditional branching* within a task. Tasks are excellent places to do data validation if you can't, or don't want to, do it in a Report definition.

In the task example shown above, we introduce the **Procedure.If** element, which allows you to use conditional branching in your task. In the example, we want to validate a password entered by the user in a report definition. The Save button in the report calls this task and the Procedure.If element's **Expression** attribute uses an @Request token to access the password value. If the Expression attribute formula evaluates to *True*, then its child elements will be processed. If it evaluates as *False*, the child elements are ignored and the next sibling element down will be processed.

So, if the evaluation is *True* (the password is less than 8 characters long), we *skip* the **Procedure.SQL** element and run the **Response.Report** element. A further enhancement to this task would be to add a **Link Parameters** element beneath Response.Report to pass an error message back to the calling report.

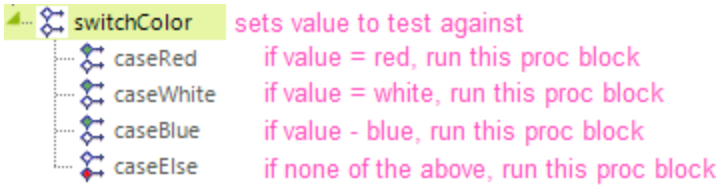
If the evaluation is *False*, the password length is acceptable and the next sibling element, the Procedure.SQL element, is processed. Once it completes, processing goes on to the Response element at the end of the task to redirect processing elsewhere.

You can have *multiple* Procedure.If elements in a task and they can be nested. They're useful, of course, for many other purposes besides data validation and can be used any time you want to direct processing flow based on evaluation of an expression.

Several other elements are available for use in controlling conditional branching:

Procedure.Else - This element *must* be a sibling of, and immediately follow, a Procedure.If element. It identifies a conditional block of procedures that will be run if the Procedure.If element immediately above it evaluates to *False*. If the Else block ran, the token `@Procedure.myProcedureID.rdReturnValue~` returns *True,* otherwise it returns *False*.

Procedure.Switch - This element works with one or more child **Procedure.Switch Case** elements to define conditional blocks of procedures to be run when a specified value matches. Use it when there are multiple conditional blocks defined and just one of them is to be run, based on the value of a variable. You can also use a child **Procedure.Switch Else** element, which runs its block of procedures if *none* of the previous Switch Case elements match the specified value.

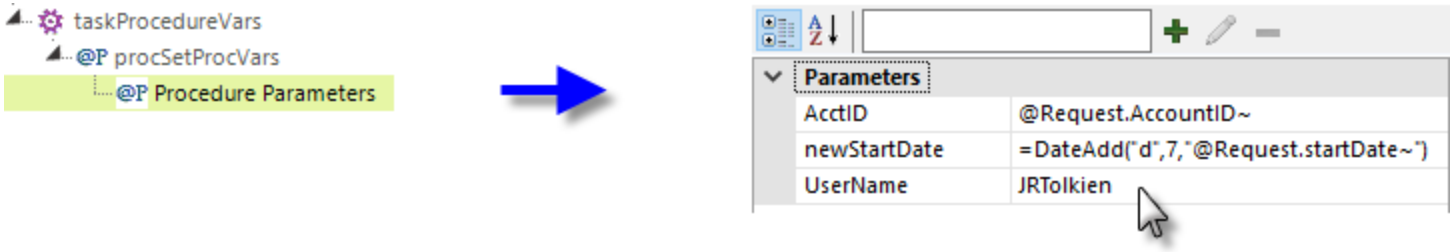


The value to test against is specified in the Expression attribute and can be literal value, a token, or even JavaScript that evaluates to a value. The Data Type attribute ensures comparisons, especially of dates, are made correctly.

Setting Variables in a Task

In a task, you may need to have working variables that you can manipulate. For example, you may need to perform a calculation on a Request variable value. There are several ways to create these "local variables".

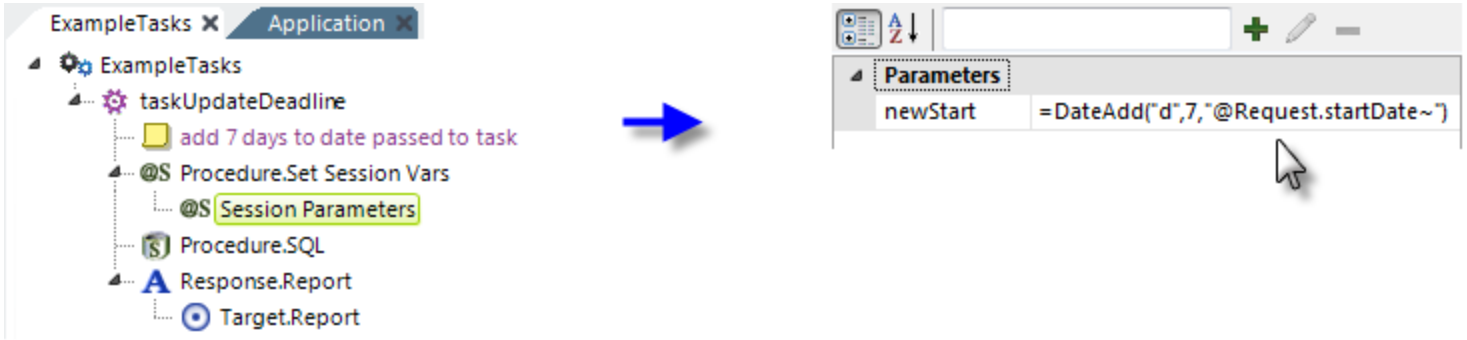
The **Procedure.Set Procedure Vars** element allows you to set variables and values that can be used for calculations, etc., within the scope of the task.



Variables are created using a child **Procedure Parameters** element, which defines name-value pairs, just like other parameters elements, as shown above. The values can be set using literals, tokens, or expressions prefaced with the "=" sign. The resulting procedure variables can be referenced anywhere in the task using this token:

@Procedure.*<Procedure.Set Procedure Vars element ID>.<variable name>*~

Session variables can be also used in the same way for this purpose:



In the example above, a **Procedure.Set Session Vars** element has been added. The new session variable it creates applies the DateAdd function to a date passed as a Request variable. Now, in the Procedure.SQL element, we can use the token @Session.newStart~ to update a database table with the new date.

Session variables set in tasks are, of course, globally available and can be accessed later in report definitions and other tasks.

File System Interactions from a Task

Tasks can include Procedure elements that allow direct manipulation of files on the web server or on network-connected drives. To do this, of course, the account used by the web server to run your Logi application needs to have appropriate file access permissions.

Here are the elements available for file system interactions:

Element	Description
Procedure.Compress File	Compresses the named file, using the .ZIP format. You must specify both the source and destination file names. Wild cards are allowed, see below. Specify a fully-qualified path and file name.
Procedure.Compress Folder	Compresses the named folder and its contents using the .ZIP format. Maintains the hierarchy of any sub-folders and files within it. You must specify both the source folder name and destination file name. Specify a fully-qualified path and folder name.
Procedure.Copy File	Copies a single file. You must specify both the source and destination file names. If it already exists, the destination file is overwritten without warning. Specify a fully-qualified path and file name.
Procedure.Create Folder	Creates a new file system folder. No error occurs if the folder already exists. Specify a fully-qualified path and folder name.
Procedure.Delete File	Deletes a file. No error occurs if the named file does not exist. Specify a fully-qualified path and file name.

Element	Description
Procedure.Delete Folder	Deletes a folder and its contents. No error occurs if the specified folder does not exist. Specify a fully-qualified path and folder name.
Procedure.File Exists	Returns *True* and executes its child elements if the named file is found in the web server file system. Specify a fully-qualified path and file name. The token `@Procedure.<myProcedureElementID>.rdReturnValue~` will return *True* or *False* depending on file existence.
Procedure.Folder Exists	Returns *True* and executes its child elements if the named folder is found in the web server file system. Specify a fully-qualified path and folder name. The token `@Procedure.<myProcedureElementID>.rdReturnValue~` will return *True* or *False* depending on folder existence.
Procedure.Run Shell Command	Executes an OS command-line command or application and optionally passes in parameters. See the section below for more information.
Procedure.Uncompress File	Expands a compressed .ZIP file. If no destination folder is specified, the files are written to the application's rdDataCache directory. Wild cards are allowed for selectively identifying files to be uncompressed, see below. You must specify a fully-qualified path and compressed file name.
Procedure.Uncompress Folder	Expands a compressed folder and its contents. Maintains the hierarchy of any sub-folders and files within it. You must specify both the source file name and the destination folder name and they must include a fully-qualified path.

Element	Description
Procedure.XML Modifier	Allows you to update an XML file from a Process Task. Information about using this element is available in Procedure.XML Modifier.

You can use tokens, such as @Function.AppPhysicalPath~ and @Data.FileName~, to provide the part or all of the file or folder names required in these attributes. Valid characters that can be used in file and folder names are those allowed by the web server operating system.

The wildcards characters * and ? can be used where mentioned in the descriptions above. For example, to compress all files in a folder, leave the Files To Compress attribute *blank* or enter **.** and to compress only XML files that start with an S, enter *S*.xml*.

Using Procedure.Run Shell Command

The **Procedure.Run Shell Command** element allows you to run OS shell commands or applications from a task and handle their output.

Commands and applications launched using this element run *synchronously* and will *block* the processing of the rest of the task until they complete.

Here are the element's attributes:

Attribute	Description
ID	(Required) Specifies a unique identifier for this element.

Attribute	Description
Error Output Filename	Specifies an method of handling error output from a shell command, as follows: • Leave blank to have error output included in the return values, or • Specify a fully-qualified file path and file name to send error output to a file, or • Specify "NUL" to ignore error output If left blank, the error output can be accessed using the token `@Procedure.<myProcedureID>.ErrorOutput~`.
Filename	Specifies the filename of a shell or application to execute, as follows: • For Windows PowerShell, enter `powershell` and preface Shell Command Parameters with "/C". • For the Windows Command Line, enter `cmd` and preface Shell Command Parameters with "/C". • For Linux, enter the path to the command, such as `/usr/bin/cp`. • For applications, enter the fully-qualified path and filename of the executable file. If *not* located within your Logi application folder, then the account used by web server to run Logi app must have Read & Execute file access permissions for the executable file. • Tokens may be used here.

Shell Command Parameters	Specifies a string of parameters for the shell command. In a command line, this is the typically the part of the command that goes after the actual command name. Begin the string with "/C" if using the Windows Command line. Examples, with parameter string highlighted in yellow:
	
	`powershell /C Test-NetConnection`
	`cmd /C COPY *.* c:\temp /Y`
	`/usr/bin/ps -ef`

Attribute	Description
	Be sure to include any switches needed to suppress any prompts, such as "/Y" in the COPY example above. Otherwise, the task will "hang" indefinitely.
Standard Output Filename	Specifies a fully-qualified path and filename to be used to store the shell command's or application's "standard output". This is the output typically seen when running a shell command from a command prompt. Tokens may be used here. If left blank, standard output can be accessed using `@Procedure.<myProcedureID>.StandardOutput~`.
Timeout	Specifies the length of time, in seconds, to wait for an action to complete before a timeout error occurs. If set to 0, then the task will wait indefinitely until the request completes. Entering a value here is a good idea, to protect against the task "hanging", but be sure to allow enough time for the shell to launch, interpret and execute the command and any parameters, and return a result. You may wish to experiment with different values and you can enter decimal values less than a whole second, such as 0.001, if appropriate.
Working Directory	Specifies the fully-qualified path to the folder to be used as the context of the shell command or application. Tokens may be used here.

The following tokens are available for use with Procedure.Run Shell Command:

Token	Description
@Procedure.*myProcedureID*.Er-	If the element's Error Output Filename attribute has been left blank, this token contains the shell command's or application's error output. If no error has occurred, this token will have no value.

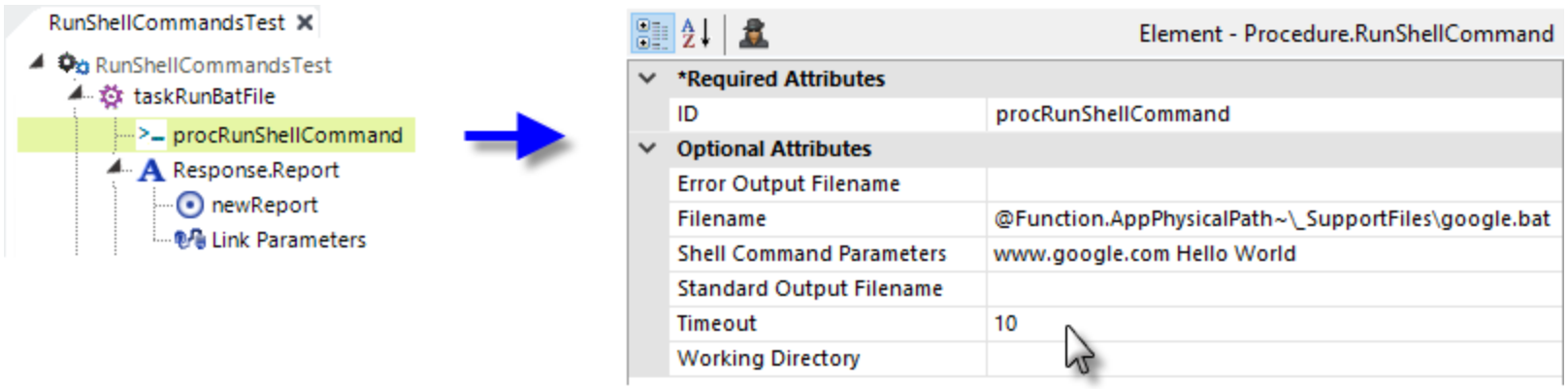
Token	Description
rorOutput~	
@Procedure.*myProcedureID*.ExitCode~	Contains the "exit" or "return" code from the shell command or application, often *0* for success.
@Procedure.*myProcedureID*.StandardOutput~	If the element's Standard Output Filename attribute has been left blank, this token contains the shell command's or application's standard console output. If there is no standard output, this will have no value.
@Procedure.*myProcedureID*.TimedOut~	Contains *True* if a timeout occurred; otherwise contains *False*.

USAGE EXAMPLE: RUN A WINDOWS BATCH FILE

In this example, we'll use Procedure.Run Shell Command to run a simple Windowsbatch file that will "ping" a web site and display some parameters. Here's the batch file contents:

```
ping %1
echo %2 %3
echo
```

As you can see, it expects three parameters.



In the task definition shown above, you can see how the Procedure.Run Shell Command element's attributes are configured. Note the use of a token in the **Filename** value, and that there are three separate parameters, separated by a space, in the **Shell Command Parameters**.

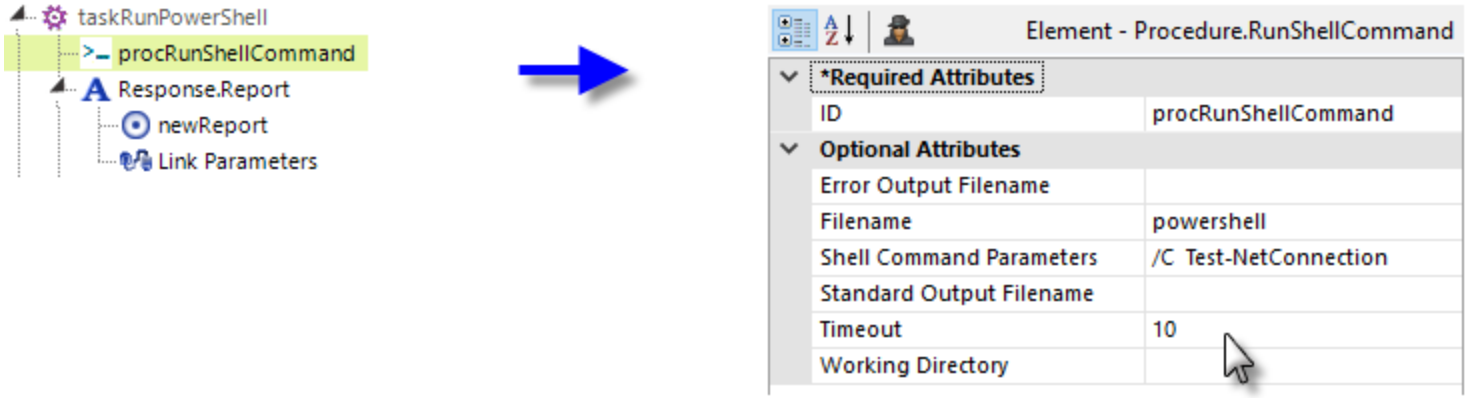
Link Parameters are used in our example to pass the four related Procedure tokens to the response report.

```
Results -

Exit Code:  0

Standard Output:

   c:\windows\system32\inetsrv>ping www.google.com

   Pinging www.google.com [172.217.7.132] with 32 bytes of data:
   Reply from 172.217.7.132: bytes=32 time=2ms TTL=54
   Reply from 172.217.7.132: bytes=32 time=2ms TTL=54
   Reply from 172.217.7.132: bytes=32 time=2ms TTL=54
   Reply from 172.217.7.132: bytes=32 time=2ms TTL=54

   Ping statistics for 172.217.7.132:
       Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
   Approximate round trip times in milli-seconds:
       Minimum = 2ms, Maximum = 2ms, Average = 2ms

   c:\windows\system32\inetsrv>echo Hello World
   Hello World

   c:\windows\system32\inetsrv>echo
   ECHO is on.


Error Output:

Timed Out: False
```

The results are displayed in the response report as shown above.

USAGE EXAMPLE: TEST A NETWORK CONNECTION

In this example, we'll use a Windows PowerShell command to retrieve information about the network:



In the task definition shown above, you can see how the Procedure.Run Shell Command element's attributes are configured. Note the use of the shell invocation in the **Filename** value, and that "/C" is added before the shell command in the **Shell Command Parameters**.

```
Results -

Exit Code:  0

Standard Output:

    ComputerName            : internetbeacon.msedge.net
    RemoteAddress           : 13.111.4.55
    InterfaceAlias          : Ethernet
    SourceAddress           : 199.169.155.5
    PingSucceeded           : True
    PingReplyDetails (RTT)  : 2 ms

Error Output:

Timed Out: False
```

The results are displayed in the response report as shown above.

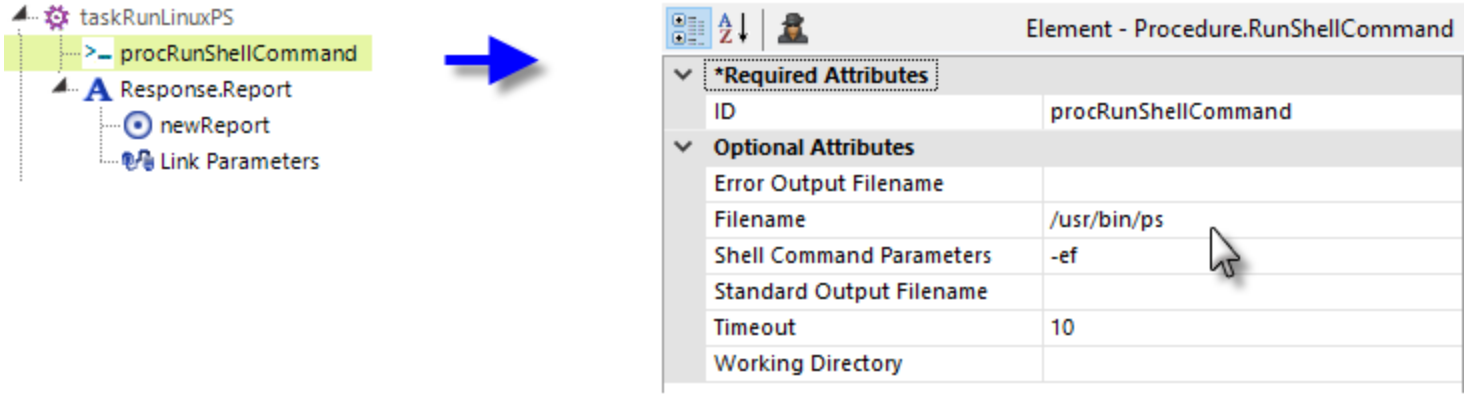
For purposes of illustration, let's make the command invalid by adding "x" to it, so the Shell Command Parameters value is "/C Test-NetConnectionx", and re-run the task:

```
Results -

Exit Code:  1

Standard Output:

Error Output:

    Test-NetConnectionx : The term 'Test-NetConnectionx' is
    not recognized as the name of a cmdlet, function, script file,
    or operable program. Check the spelling of the name, or if a path was
    included, verify that the path is correct and
    try again.
    At line:1 char:1
    + Test-NetConnectionx
    + ~~~~~~~~~~~~~~~~~~~
        + CategoryInfo          : ObjectNotFound: (Test-
    NetConnectionx:String) [], CommandNotFoundException
        + FullyQualifiedErrorId : CommandNotFoundException

Timed Out: False
```

Now we can see that the Exit Code, Standard Output, and Error Output values have changed.

USAGE EXAMPLE: RUN A LINUX COMMAND

In this example, we'll use the Linux "ps" command to display a formattedlist of active processes:



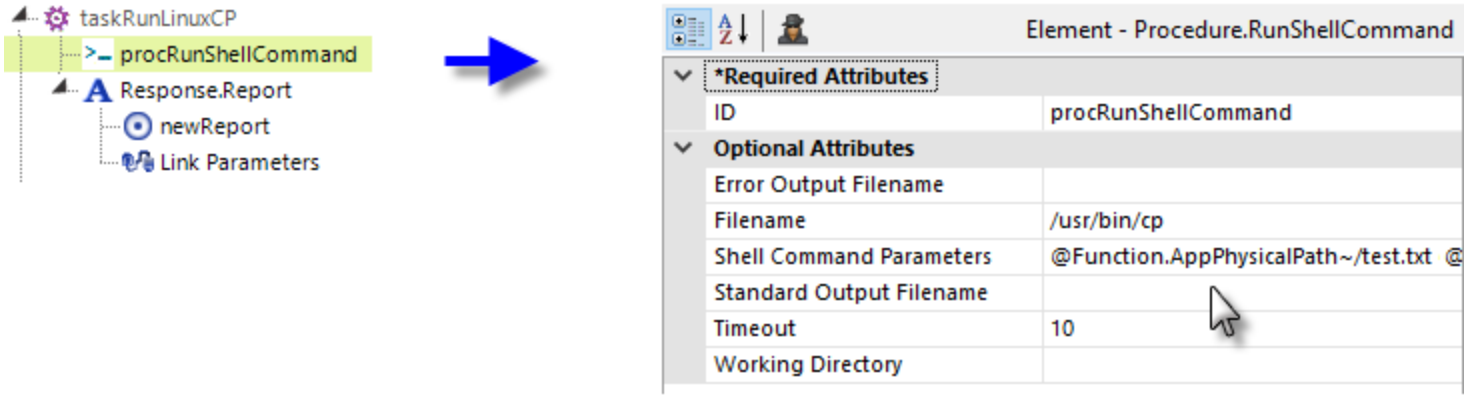
In the task definition shown above, you can see how the Procedure.Run Shell Command element's attributes are configured. Note the use of the Linux "ps" command in the **Filename** value, and the "-ef" flags in the **Shell Command Parameters**.

```
Results -

Exit Code:  0

Standard Output:

  UID    PID    PPID  C  STIME    TTY    TIME         CMD
  root     1       0  0  13:04    ?      00:00:01     /usr/lib/systemd/systemd
  root     2       0  0  13:04    ?      00:00:00     [kthreadd]
  root     4       2  0  13:04    ?      00:00:00     [kworker/0:0H]
  root     5       2  0  13:04    ?      00:00:00     [kworker/u2:0]
  root     6       2  0  13:04    ?      00:00:00     [mm_percpu_wq]

Error Output:

Timed Out: False
```

The results are displayed in the response report as shown above.

USAGE EXAMPLE: RUN A LINUX COMMAND USING TOKENS

In this example, we'll use the Linux "cp" command to copy a file:



In the task definition shown above, you can see how the Procedure.Run Shell Command element's attributes are configured. Note the use tokens in the **Shell Command Parameters** value, which in its entirety is:

```
@Function.AppPhysicalPath~/test.txt @Function.AppPhysicalPath~/test.out -f
```

The ability to run shell commands can be dangerous, so use this element with care.

Exporting from a Task

You can create tasks that include procedures that export reports or data. This is done in the task by running a "hidden instanced" of an existing Report definition and then exporting it or its data.

There is a difference between exporting reports from a Report definition and from a Process task. An export from a Report definition is written out as a *temporaryfile* in your application's `rdDownload` folder and then that file is opened for viewing in the user's browser. The temporary file is automatically deleted later. An export from a Process task is saved as a file in a location, and with a filename, you specify and it's not opened automatically in the browser. It's *not* considered a temporary file, so it's not deleted later automatically; you may have to manage it separately.

So, exporting from a Process task is useful when you need to save a file to the web server for later use, such as:

- Archiving data or reports
- Creating attachments to be sent with e-mails
- Building a custom Data Cache (XML export) that can be used later by your reports

More information about exporting can be found in our export topics.

As shown in the example above, an export procedure such as **Procedure.Export PDF** is added to a task, and its **Filename** attribute is set to the fully-qualified path for the export file, including the file name and extension. The account being used to run the Logi app must have Write permissions to the storage location (usually this is already in place if you're saving to any folder within your application folder), and the @Function token for the application path can be used here, as shown above.

The **Target.PDF** element can be used to specify the **Report Definition File** to be exported. If not specified, the "Current Report" definition - the one that called the task - will be exported. Why have an option? You may have created two similar definitions - one for browsing and one for exporting.

If you specify a table in the Target.PDF element's **Data Table ID** attribute, then just that table's *data* (without headers, footers, etc.) will be exported. See Logi Studio's Information Panel text for more information about other Target element attributes that are specific to different export types.

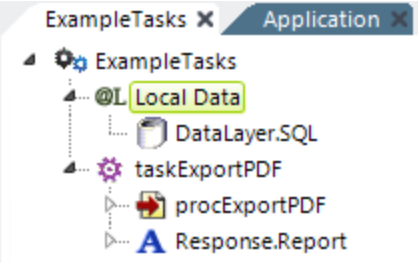
If the original report depended on Request variables to set queries, filtering, etc. then be sure to either set the Target element's **Request Forwarding** attribute to *True*, or to use **Link Parameters**, to ensure that the exported file is similarly configured.

The elements for exports to other formats (Word, Excel, CSV, etc.) operate in a similar fashion.

DevNet includes a sample application that demonstrates exporting using process tasks.

Using Local Data in Tasks

The **Local Data** element is available for use in Process definitions.



It operates in them in the same manner that it does when used in Report definitions and the data its datalayer retrieves can be accessed from any task in the Process definition using an @Local Data token. For a full explanation of this element, see our *Datalayer Introduction* document.

SQL Database Interactions from Tasks

Tasks also provide the ability to interact with SQL databases, including the so-called "write-back" capability required to update databases. Both SQL Queries and Stored Procedures are supported, with special elements, and all valid SQL commands, including INSERT, UPDATE, and DELETE, can be used.

Using SQL Statements

Implementing a direct SQL statement in Process task is easy and tokens can be used to "parameterize" the statement:

In the example shown above, we see a task that includes the **Procedure.SQL** element. The complete SQL Command in the example is:

```
UPDATE Products SET UnitsInStock = @Request.inpHidden~ WHERE ProductID = @Request.ProductID~
```

Any valid SQL statement 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 for this attribute is *False*.

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

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

then with tokens it would be:

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

The **SQL Return Type** attribute specifies the return value type of the SQL operation:

- If *RowsAffected* (the default) is selected, the operation will return the number of rows changed by any UPDATE, INSERT, or DELETE commands. You can access this value later in the task using the special token: `@Procedure.<yourProcedureSQL_ID>.RowsAffected~`.
- If *FirstRow* is selected, the first row of any result set from the SQL operation is returned. You can access the values of the returned row columns later in the task using tokens in this format: `@Procedure.<yourProcedureSQL_ID>.<ColumnName>~`.

The example above shows an UPDATE operation but queries using SELECT statements are, of course, also supported. Values returned by a query using Procedure.SQL are limited to those in the first row, as described above. If you want to return *multiple* rows, use **Procedure.Run DataLayer Rows** and a datalayer, as described in "Running Datalayer and Data Table Rows" on page 386.

The **If Error** element can be used beneath Procedure.SQL to handle errors that may occur. The actual error message text is available in this token: `@Procedure.<yourProcedureSQL_ID>.ErrorMessage`

Procedure.SQL elements can also be used beneath Procedure.Run Data Table Rows, Procedure.Run DataLayer Rows, which are discussed in "Running Datalayer and Data Table Rows" on page 386.

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, write your SQL statement using "placeholder" notation. The exact syntax for this 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.

 There is no need to surround the placeholder with quotes, even if it represents string data. The Logi Engine sees the parameter data type in the element attributes and handles it appropriately.

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

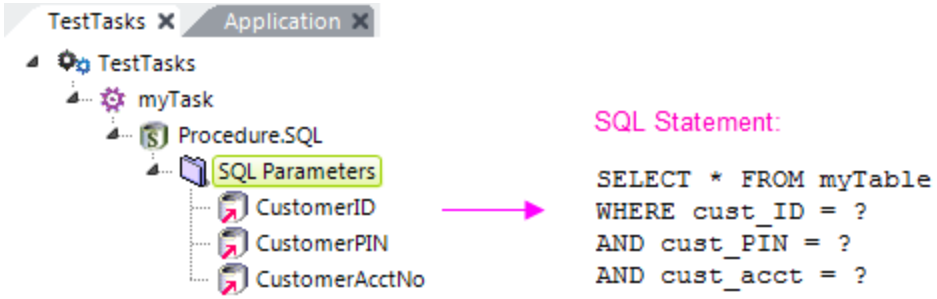
```
SELECT * FROM Customers WHERE State=:state AND City=:city
```

and, as before, the **ID** attribute of each SQL Parameter element must match a placeholder *name*.

However, 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**.

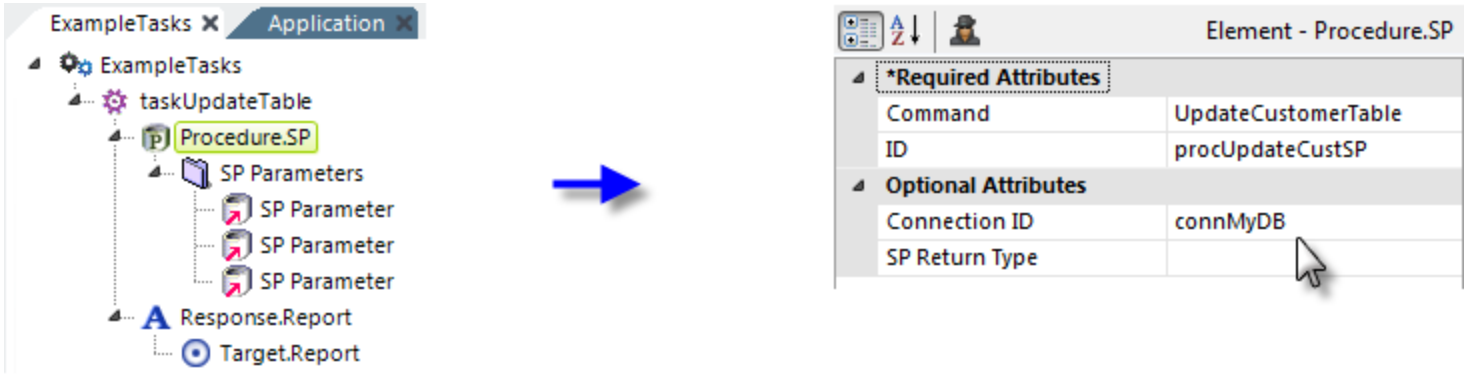
Consult your database documentation for specific placeholder information.

SQL Parameters are especially useful if you're *writing* string data that includes embedded single quotes to the database. The Logi Engine automatically handles escaping the single quotes in the SQL statement sent to the database server.

Using Stored Procedures

We highly recommend the use of stored procedures, especially when updating the database. It's the most secure method and usually offers the best performance. A stored procedure is code, possibly pre-compiled, that resides on your database server and is

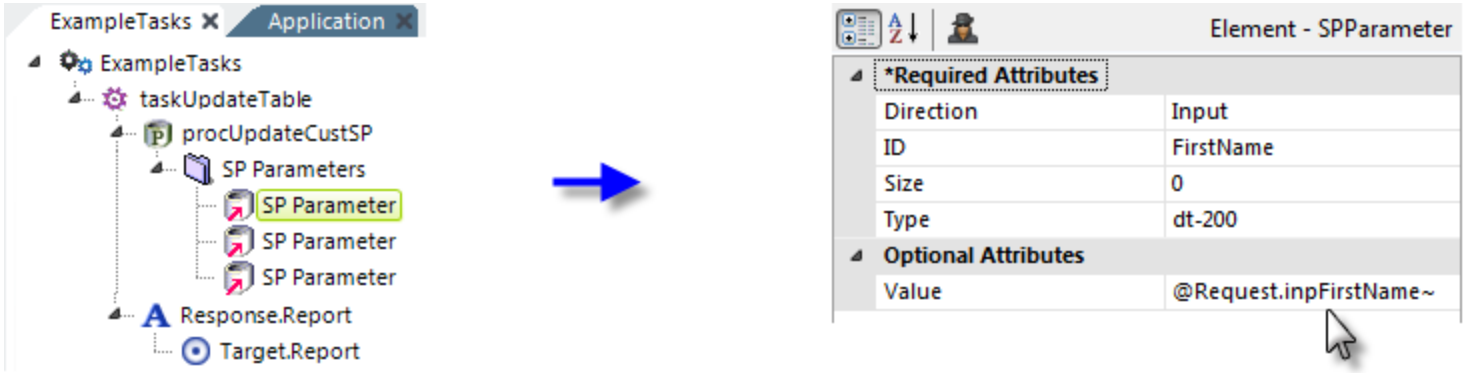
called to perform work. It's usually a collection of standard SQL commands.



In the example task above, a **Procedure.SP** element has been added in order to invoke a stored procedure. Its attributes are set as shown to connect to, and run, the stored procedure. The **Command** attribute includes a pull-down list of all of the stored procedures available through the database connection specified.

Stored procedures are written using the tools that came with your database server; some servers may accept code written in an external text editor such as Notepad. Logi Studio does *not* include an editor for writing stored procedures.

Two other types of elements can be used if the stored procedure requires parameters: the **SP Parameters** element, which is just a container, and one or more **SP Parameter** elements.



The first **SP Parameter** element has been configured, as shown above, as an Input parameter. It will pass the First Name value from a Request variable to the stored procedure. Its **Direction** attribute is set to *Input*, and its **Size** attribute value is set to *0*, which causes it to be adjusted to the correct size automatically at runtime. Other SP Parameters will be configured similarly, to pass additional values.

The relationship between parameters in the task definition and the parameters in the stored procedure is determined solely by their *order*, not by their IDs.

SP Return Values

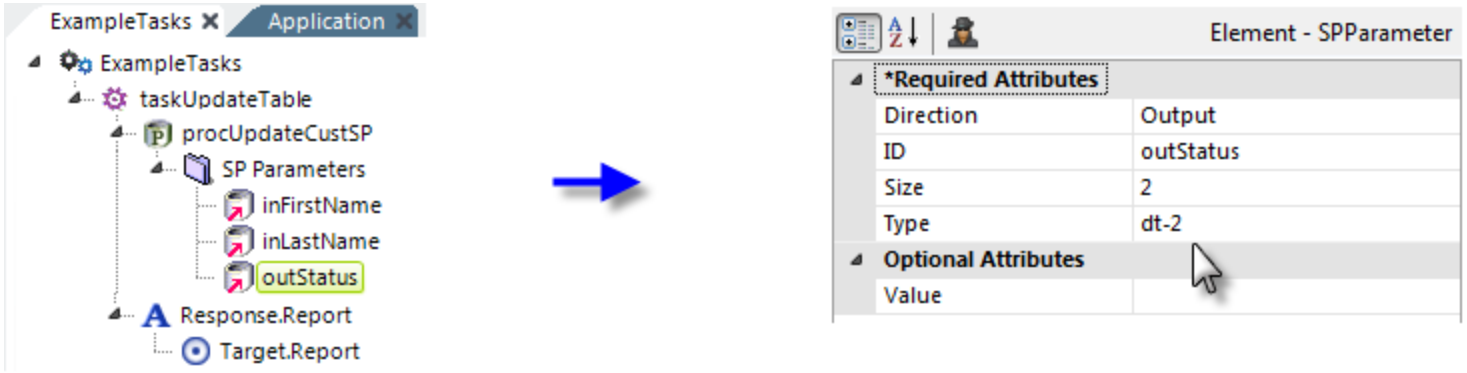
The Procedure.SP return behavior is controlled by its **SP Return Type** attribute, which has two possible values:

When configured as *ReturnValue* (the default) only the value of the first column of the first row of data returned by the stored procedure, if any, is available. Additional columns or rows are ignored. The returned value can be accessed with an @Procedure "rdReturnValue" token, for example: `@Procedure.myProcedureId.rdReturnValue~`

When configured as *FirstRow*, the entire first row of data returned by the stored procedure is available. Access the values of the first row with @Procedure tokens. For example: `@Procedure.myProcedureId.someColumnName~`

Using SP Output Parameters

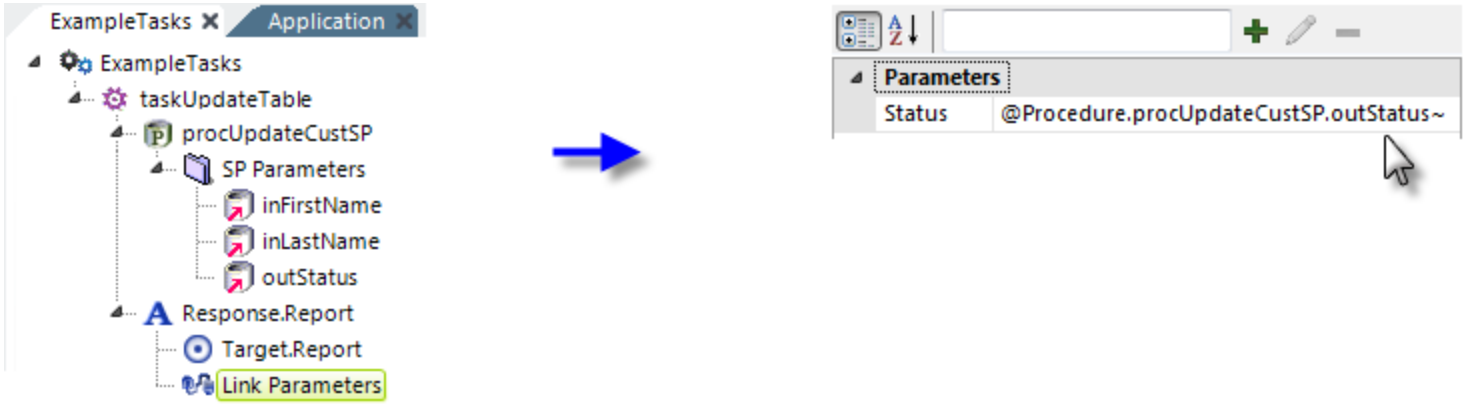
Another mechanism for returning data is to have the stored procedure return *output parameters* and the task can be configured to receive them:



To accomplish this, the example task has now been given an SP Parameter element that's configured as an output parameter. Note that its **Direction** attribute is set to *Output*. The stored procedure will return a value to the task, using this parameter. In this case, the **Size** attribute must be set to the number of bytes for its data type, or the maximum expected to be returned.

Once again, the relationship of parameter elements in the task and the stored procedure's output parameters is determined solely by their *order*, not their IDs.

The value returned in the output parameter is available using a special token, shown below:



A **Link Parameters** element has been added to the task so that we can pass the results back to a report. Its **Status** attribute has been configured to use the special token that holds the value passed to the output parameter element. The token format is:

```
@Procedure.<Procedure.SP Element ID>.<SP Parameter Element ID>~
```

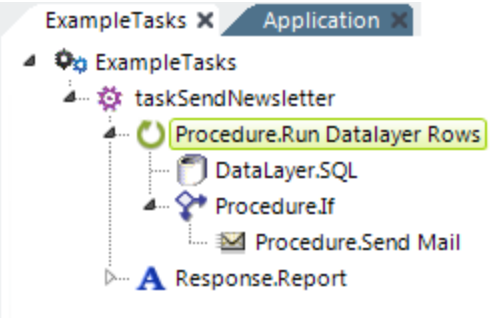
DevNet includes a sample application that demonstrates the use of a Process task to update a table.

Running Datalayer and Data Table Rows

Tasks offer you the ability to retrieve just one row of data, or to iterate through all of the rows in either a datalayer or a Data Table in a Report definition, processing the data in each row individually.

In a manner similar to the Local Data element, the **Procedure.Data** element can be used to run a datalayer; the column values from the first datalayer row (only) can then be accessed in the task using this token: @Procedure.*<Procedure.Data element ID>.<column name>*~.

The examples below show you how to *iterate* through multiple datalayer or Data Table rows :



In this example task, we'd like to look through a contact database table and send out an email newsletter for every record that has an email address. Here's a description of the purpose of each element used:

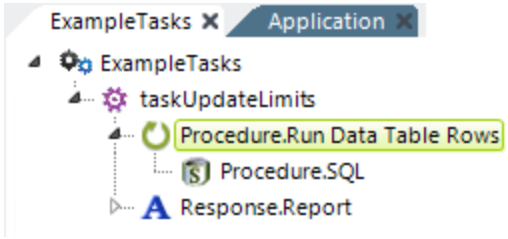
1. The **Procedure.Run Data Layer Rows** element is the parent container that causes the iteration through each row in the datalayer once data has been retrieved.
2. A **DataLayer.SQL** element is used to query the database table, returning all of the records into the datalayer. Other elements beneath the Procedure.Run Data Layer Rows element can access data values in the datalayer using standard @Data tokens.

3. The Procedure.If element is configured so that its child element, Procedure.Send Mail, will run *only* if the email address column in the current datalayer row is not Null.
4. The **Procedure.Send Mail** element uses the token @Data.EmailAddress~ in its **To Email Address** attribute to send out an email for the current datalayer row.
5. The **Response.Report** element, of course, redirects processing somewhere else when the task completes.

Let's look at another example: imagine a report definition that includes a Data Table and one of its columns has an **Input Text** element displayed in every row, like this:

Product ID	Product Name	Unit Price	Stock Limit
1	Chai	$18.00	555
2	Chang	$19.00	24
3	Aniseed Syrup	$10.00	5454
4	Chef Anton's Cajun Seasoning	$22.00	89
5	Chef Anton's Gumbo Mix	$21.35	4
6	Grandma's Boysenberry Spread	$25.00	67
7	Uncle Bob's Organic Dried Pears	$30.00	9259

At runtime, the user can enter numbers to adjust the table values. But, how would you update the database after multiple changes have been made and the page is submitted? That's right - you can do it with a task.



Here's a description of the elements needed:

1. The **Procedure.Run Data Table Rows** element is the parent container that causes the iteration through each row in the Data Table named in its **Data Table ID** attribute. If the attribute is left blank, the first table found in the application that includes user input elements will be used.
2. The **Procedure.SQL** element is used to issue an UPDATE command to the database, using an @Request token to access the value of the Input Text element for each table row.
3. The **Response.Report** element, of course, redirects processing somewhere else when the task completes.

Important Limitations

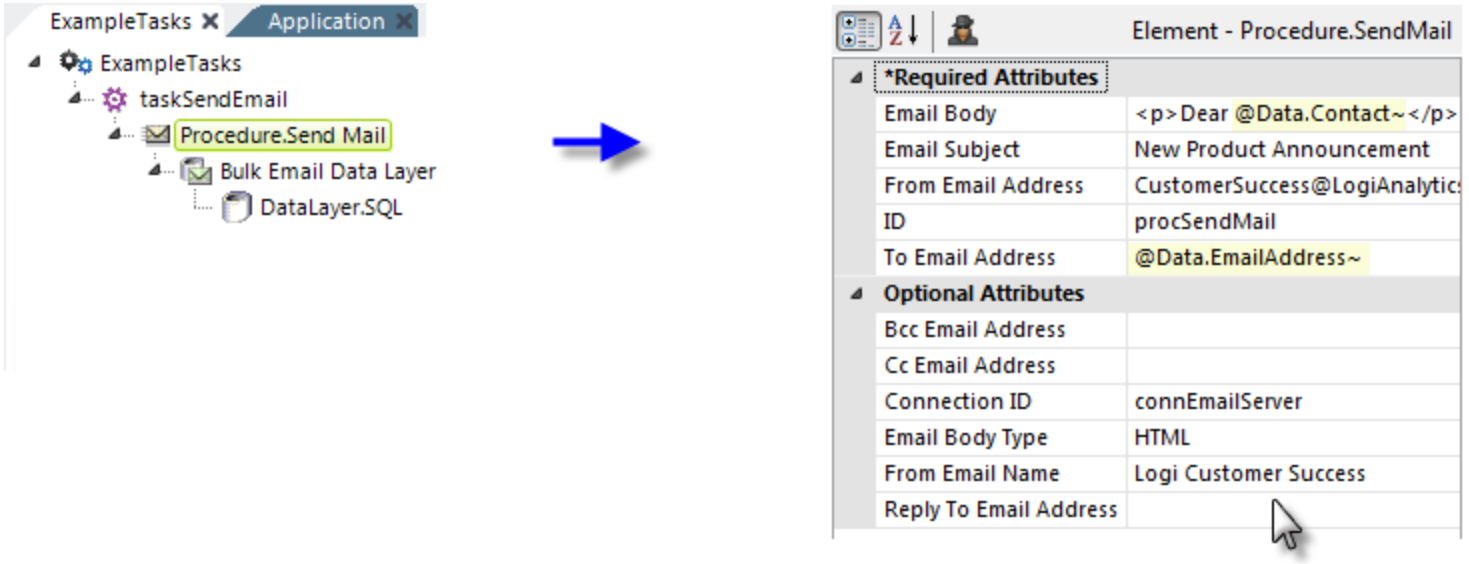
Please note these two important limitations:

- **Interactive Paging** should *not* be used with Data Tables being processed using Procedure.Run Data Table Rows. If possible, use input elements and datalayer filtering elements instead to limit the amount of data in the table.
- **Input check boxes** should *not* be the sole Input element within a Data Table row when using Procedure.Run Data Table Rows. This Input element is unusual in that it does not submit a Request variable value at all if the box is not checked and, when this is the only Input element in the table, it will cause Procedure.Run Data Table Rows to only "see" the rows that are selected, creating an incorrect listing of the rows. To avoid this, you can include an **Input Hidden** element within the table

that stores a unique ID for each row. This will ensure that each row is represented by a Request variable and gets processed by Procedure.Run Data Table Rows.

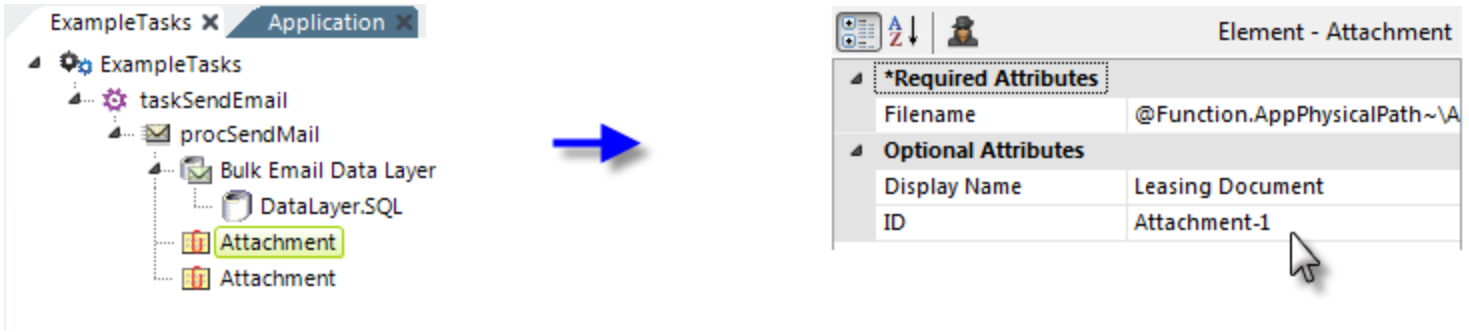
Sending Email from a Task

A process task also gives you the ability to send email messages, separately or in bulk.



An example in "Running Datalayer and Data Table Rows" on page 386 demonstrated one method of sending out a mass mailing, but there's a second method you may want to use instead.

1. Add a **Procedure.Send Mail** element to your task, as shown above, and set its attributes as desired to configure the message to be sent. The actual message (the "Email Body") can be formatted as *Plain Text* or *HTML*. Use @Data tokens to reference the data values that will be coming from your data source.
2. Beneath it, add a **Bulk Email Data Layer** element, which iterates through its child datalayer rows.
3. Beneath it, add an appropriate datalayer element to retrieve the data with the addressing information. If possible, use a query that includes a filter to eliminate records that have Null email addresses.



4. Optionally, add one or more **Attachment** elements beneath the Procedure.Send Mail element. Set its **Filename** attribute to the fully-qualified file path and file name of the file to be attached. The @Function token for your application folder can be used here, as shown above. Set the **Display Name** attribute to the text that you wish to appear identifying the attachment in the recipient's mail reader.

   Keep in mind that your mail server may impose restrictions on the *size* of file attachments.

Procedure.Send Mail element supports TLS/SSL encryption, allowing, for example, GMail to be used as an SMTP server. Set the related **Connection.SMTP** element's **SMTP Authentication Method** attribute to *3* to invoke this, and its **SMTP Port** attribute may need to be set to *587*.

When sending email from a Java application to an older SMTP server, attachments with long file names (> 60 chars.) result in a split name that cannot be clicked to be viewed by the recipient. To solve this, keep attachment names short, or add this JVM option:

```
-Dmail.mime.encodeparameters=false
```

Sending out thousands of emails may have an impact on your network bandwidth and your email server performance, so use caution before "opening the floodgates" and sending out a zillion messages!

DevNet includes a sample application that demonstrates the use of a process task to send email.

Web Service Interactions

The **Procedure.REST** element can be used to interact with a REST-style web service from within a process task. Typically, this would be to send a request that invokes an HTTP PUT or DELETE method, but GET and POST are also supported.

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.

The Procedure.REST element has attributes that are similar to those of DataLayer.REST:

Attribute	Description
ID	(Required) Specifies a unique element ID.
Accept Type	Specifies the type of data expected to be received from a REST request. Options include *application/json* and *application/xml* (the default).
Connection ID	Specifies the ID of a **Connection.REST** element defined in the _Settings definition.
Http Method	Specifies the verb to be sent with a REST request. Standard options include: *DELETE*, *GET*, *POST*, and *PUT,* and you can also enter custom verbs, like *PATCH*, if necessary. The default value is *GET*.
ID	Specifies a unique element ID. You'll need to provide this if you want to use tokens to get a returned code or description.
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*.
Url Path	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` Procedure.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.`

You can reference the web service's response code and description with these Procedure tokens:

- @Procedure.*yourProcedureID*.rdHttpResponseCode~
- @Procedure.*yourProcedureID*.rdHttpResponseDescription~
- @Procedure.idof Restprocess.rdhttpResponseBodyContent~

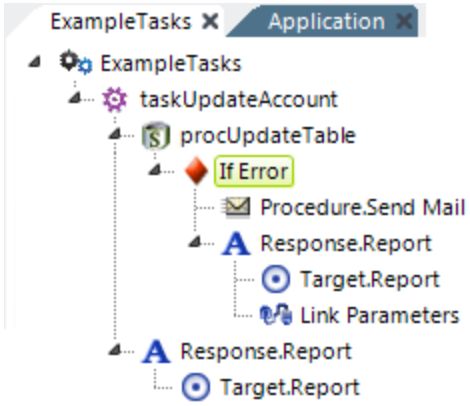
Procedure.REST can also use the **Http Body** child element, which has two attributes:

- **Content Type,** which sets the content type in the request header and defaults to *application/x-www-form-urlencoded*.
- **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.

Error Handling in Tasks

Error handling is an important function for any application and tasks include a special element for this purpose.



The **If Error** element allows you to handle errors that occur when running procedures in a task. The example above shows a typical implementation: if an error occurs when a table update is attempted, an email alert will be sent out and a redirect to a report page will occur (possibly back to the page that called the task). The actual error text can be passed to that report page, using a **Link Parameters** element, as a Request variable.

The *placement* of the If Error element is important. Errors will "bubble up" through procedures until an If Error element is found, and if not found, up through tasks and the definition itself. You have the choice of placing your error handling elements in different locations to handle different situations. Without an If Error element, the application will return an error page if an error occurs.

You may use multiple If Error elements in a task, if desired. As mentioned in the previous paragraph, you can access the error message text. This is done using this special token: `@Procedure.yourProcedureID.ErrorMessage~`

Because the element ID of the procedure is part of the token, it limits the scope of the error message availability. You can also use: `@Function.LastErrorMessage~` if you prefer, and its scope is application-wide.

You also have the ability to expose detailed error information from the response content of the REST call. Access the detailed message by adding the token `@Procedure.idof Restprocess.rdhttpResponseBodyContent~`to the BodyContent field in your session parameters.

The If Error element has an **Error Filter** attribute, which allows individual If Error elements to handle different errors. Set the Error Filter attribute value to any text found within the error message. If the text is found, the If Error element will handle the error. If not, the error will be ignored and the error will "bubble up". This allows you to target specific errors, and to react to them differently, if desired. If you use this approach, you'll probably want to include a final If Error element *without* any Error Filter text, to catch any errors not specifically handled by your other handlers.

The Error Filter also allows you to *ignore* errors, by specifying their error message text and then placing *no* child elements beneath the If Error element. When that error occurs, it will be considered "handled" and processing will continue with the next element after the failed procedure element.

 The If Error element only recognizes errors that occur *in the process task* and will not react to those that occur in a report definition run by the task. For example, if a task uses Procedure.Export CSV to export data from a report definition to CSV and an error occurs when the report is run, If Error will not see it. In this case, you can use **If Data Error** in the report definition to provide alternate data when the report is exported, then **Procedure.Run DataLayer Rows** to read the exported file, test its data, and take appropriate action.

Format Data

Many elements have a **Format** attribute that lets you specify how its data will be formatted. The most common of these, for example, is the **Label** element. Applying formatting to a Label caption is especially useful for dates, times, and certain types of numbers.

The following topics describe the available options for formatting data:

About Formatting

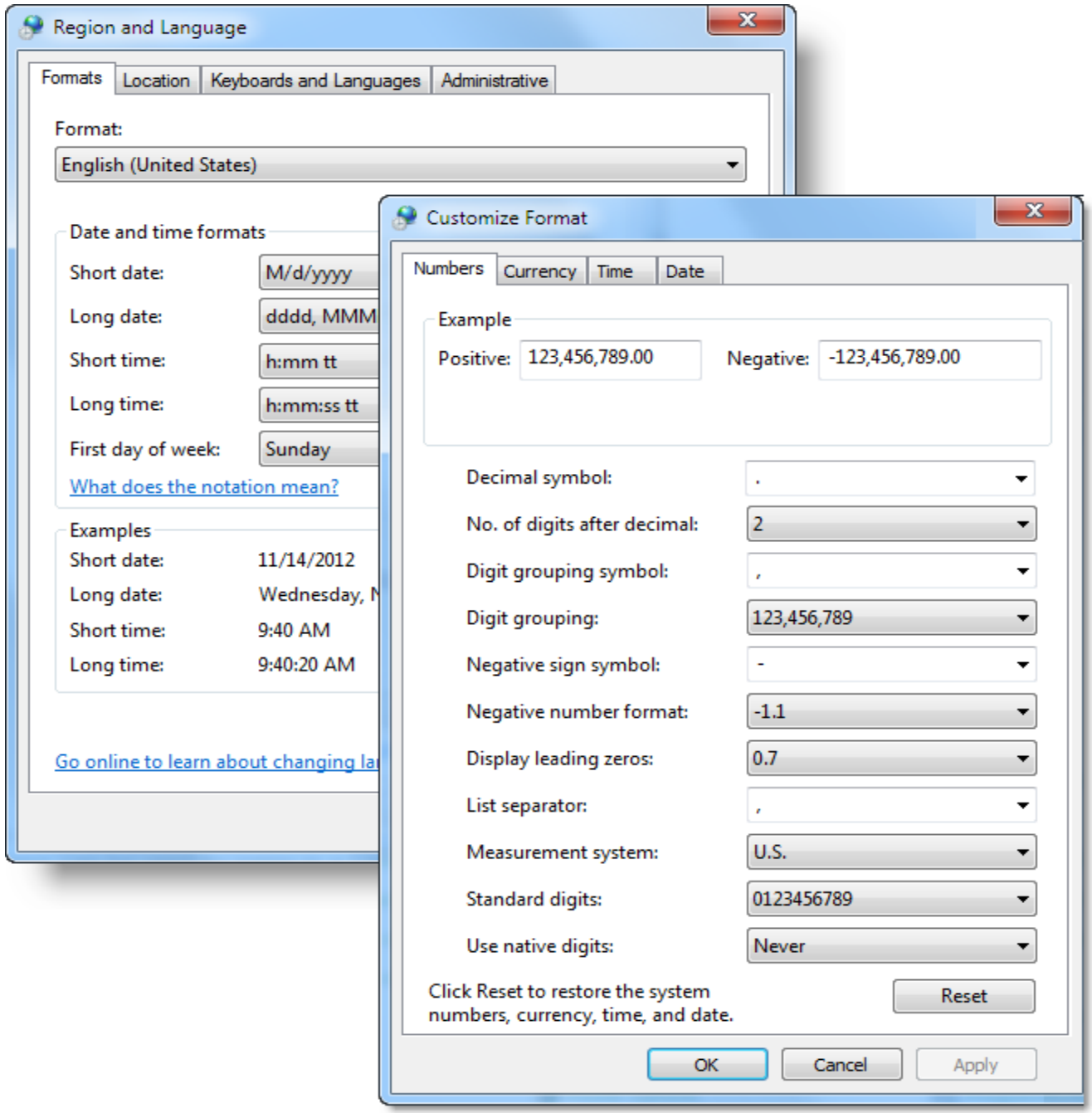
The following elements have a **Format** attribute, which can be set to format the appearance of their data:

• Analysis Chart Column • Analysis Filter Column • Analysis Grid Column • Animated Map Color Spectrum Legend • Caption Style • Chart Grid Column • Color Spectrum Legend • Crosstab Comparison • Crosstab Table Value Columns • Data Labels	• Heatmap Group • Heatmap Label • Heatmap Label Style • Input Combo List • Input Date • Input Number • Input Telephone • Input Text • Input Text Area • Input Time	• Marker Label Style • MDX Calculated Measure • PDF Form Field • Polygon Color Spectrum Legend • Quicktip Row • Schedule • Side Label Style • Stack Labels • SubCaption Style • Word Form Field

• Data Scale • Drill To Column • Drillthrough Column • Formatted Column • Gauge Label	• Label • Label Scale • Label Style • Legend	• XOLAP Calculated Measure • XOLAP Level • XOLAP Measure • XOLAP Member Property

Logi Studio is not always context-sensitive when it comes to the options available for Format attributes, so it may be possible to select formatting options that are not appropriate for the data.

Many of the formatting types available are dependent on the **Regional** settings in the web server's operating system.


Region and Language
Formats
Location
Keyboards and Languages
Administrative
Format:
English (United States)
Date and time formats
Short date:
M/d/yyyy
Long date:
dddd, MMM
Short time:
h:mm tt
Long time:
h:mm:ss tt
First day of week:
Sunday
What does the notation mean?
Examples
Short date:
11/14/2012
Long date:
Wednesday,
Short time:
9:40 AM
Long time:
9:40:20 AM
Go online to learn about changing la
Customize Format
Numbers
Currency
Time
Date
Example
Positive:
123,456,789.00
Negative:
-123,456,789.00
Decimal symbol:
.
No. of digits after decimal:
2
Digit grouping symbol:
,
Digit grouping:
123,456,789
Negative sign symbol:
-
Negative number format:
-1.1
Display leading zeros:
0.7
List separator:
,
Measurement system:
U.S.
Standard digits:
0123456789
Use native digits:
Never
Click Reset to restore the system numbers, currency, time, and date.
Reset
OK
Cancel
Apply

For example, as shown above, **Regional** and **Language** options are available on Windows-based web servers through the Control Panel. The "Currency", "Short date" and "Long date" formats defined here can be used by name in a Format attribute.

Standard Formats

The following standard formats are available, from a pull-down list, in Studio for elements' Format attribute value:

Type	Description
<	Converts all characters to lower-case.
>	Converts all characters to upper-case.
Expanded Spaces	Preserves space characters that are normally collapsed by the web browser. *This formatting will not be applied if used with content that includes more than one @Data token.*
Preserve Line Feeds	Preserves line breaks (CTRL-ENTER). *This formatting will not be applied if used with content that includes more than one @Data token.*
HTML	Preserves HTML tags, embedding them rather than encoding them. Text appearance and alignment is affected by the tags; they are not seen in the text.
General Date	Display a date and/or time. For real numbers, display a date and time, for example, 4/3/93 05:34 PM. If there is no fractional part, display only a date, for example, 4/3/93. If there is no integer part, display time only, for example, 05:34 PM. Date display is determined by the web server's settings.
Long Date	Display a date according to the web server's Long Date format.
Medium Date	Display a date using the Medium Date format appropriate for the language version of the host application on the web server.

Type	Description
Short Date	Display a date using web servers's Short Date format.
Long Time	Display time using the web server's Long Time format; includes hours, minutes, seconds.
Medium Time	Display time in 12-hour format using hours and minutes and the AM/PM designator.
Short Time	Display time using the 24-hour format, for example, 17:45.
yyyy/MM/dd	A custom date/time format defined by the developer using the components described in "Custom Date and Time Format" on page 406.
hh:mm	Display time as hours and minutes, with leading zeroes. Example: 01:08
yyyy/MM/dd hh:m-m:ss	Display date and time, with leading zeroes. Example: 2007/01/01 01:08:02
General Number	Display number with no thousands separator.
Currency	Display number with currency symbol, thousands separators (if appropriate), and two digits to the right of the decimal separator. The symbols and characters used are based on the end-user's browser culture settings or computer system settings. For example, United States = "en-us" = $1,000.00, United Kingdom = "en-gb" = £ 1.000,00.
Fixed	Display at least one digit to the left and two digits to the right of the decimal separator.
Standard	Display number with thousands separator, at least one digit to the left and two digits to the right of

Type	Description
	the decimal separator.
Percent	Display number multiplied by 100 with a percent sign (**%**) appended to the right; always display two digits to the right of the decimal separator.
Scientific	Use standard scientific notation.
Timespan	Displays decimal numbers as a time span, formatted as d:hh:mm:ss.
mp	Formats numbers by applying the appropriate "metric prefixes" (giga-, mega-, kilo-, etc.). Example: "1,234,567" formatted with "$#.00mp" produces "$1.23M". More information about metric prefixes can be found here. Customization of the mp format can be accomplished using the Globalization element's Metric Prefix String attribute. For more information, see *Using the Globalization Element*.
mps3	Identical to the mp format, but allows rounding to three significant digits. Example: A value of 123456, with format mps3, returns 123K.
###,###,##0.00	Display number with thousands separator for every three digits to left, at least one digit to the left and two digits to the right of the decimal separator.
Yes/No	Display **No** if number is 0; otherwise, display **Yes**.
True/False	Display **False** if number is 0; otherwise, display **True**.
On/Off	Display **Off** if number is 0; otherwise, display **On**.

The **HTML** format, when used with a Label caption, allows you to embed HTML code in the report. This be used to accomplish things as simple as making part of a line bold-faced using <B> tags, or as complicated as including scripts within <SCRIPT> tags for various purposes. When using the HTML format, ensure that you do not include a <BODY> tag that might confuse the Logi Server Engine when it generates its HTML output.

- The standard numeric formats, such as Currency, may cause *rounding*, and this may affect aggregations and summaries of the data.
- Use the standard date/time and numeric formats to improve performance with large reports.

Custom Date and Time Format

Developers can enter custom date and time formats by typing them in directly.

For example, one of the standard date formats is "yyyy/MM/dd". Developers can directly enter other combinations of valid formatting characters, such as "MM/dd/yyyy" to obtain the desired format. Note that format values are case-sensitive and are *not* entered with quotation marks. Here are the valid date and time formatting characters:

Chars	Description
:	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by the client OS locale setting.
/	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by the client OS locale setting.
%	Used to indicate that the following character should be read as a single-letter format without regard to any trailing letters. Also used to indicate that a single-letter format is read as a user-defined format. See below for further details.
d	Day as a number, without a leading zero: *1*. Use %d if this is the only character in your user-defined numeric format.
dd	Day of month number, with leading zero: *04*
ddd	Three-letter abbreviation of day name: *Wed*

Chars	Description
dddd	Full day name: *Thursday*
M	Full month name and day combination: *July 22*
MM	Month number: *07*
MMM	Three-letter month name abbreviation: *Jul*
MMMM	Full month name: *September*
gg	The period/era: *A.D.* Note that this is not available in **Format Label** and **Format Data** elements.
qq	Quarter number: *1, 2, 3, or 4.* Note that this is not available in **Format Label** and **Format Data** elements.
y	Year number (0-9), without leading zeros. Use **%y** if this is the only character in your user-defined numeric format.
yy	Two-digit year, with a leading zero, but without century: *09*
yyyy	Four-digit year: *2009*
h	Hour as a number, without a leading zero, using the 12-hour clock: *1:15:15 PM.* Use %h if this is the only character in your user-defined numeric format.
hh	Hour as a number, with a leading zero, using the 12-hour clock: *04*

Chars	Description
H	Hour as a number, without a leading zero, using the 24-hour clock: *13:15:15 PM.* Use %H if this is the only character in your user-defined numeric format.
HH	Hour as a number, with a leading zero, using the 24-hour clock: *01*
m	Minute as a number, without leading zeros: *12:1:15*. Use %m if this is the only character in your user-defined numeric format.
mm	Minute as a number, with leading zeros: *12:01:15*
s	Second as a number, without leading zeros: *12:1:5*. Use %m if this is the only character in your user-defined numeric format.
ss	Second as a number, with a leading zero: *12:1:05*
f	Fractions of a second. For example, ff will display hundredths of seconds, whereas ffff will display ten-thousandths of seconds. You may use up to seven f symbols in your user-defined format. Use **%f** if this is the only character in your user-defined numeric format.
T	Uses the 12-hour clock and displays an uppercase `A` for any hour before noon; displays an uppercase `P` for any hour between noon and 11:59 P.M.
t	Hour and minute with AM/PM designator: *10:32 AM*
tt	AM or PM designator alone: *PM*

Chars	Description
z	Timezone offset as a number, without a leading zero: *-8*. Use %z if this is the only character in your user-defined numeric format.
zz	Timezone offset as a number, with a leading zero: *-08*
zzz	Full timezone offset as a number, with a leading zero: *-08:00*

Visit the MSDN Library User Defined Date/Time Formats page for more information.

Custom Numeric Formats

Developers can enter custom numeric formats by typing them in directly. For example, one of the standard date formats is "###,###,##0.00". Developers can directly enter other combinations of valid formatting characters, such as "$#,##0" to obtain the desired format.

Here are the valid numeric formatting characters:

Chars	Description
None	Displays the number with no formatting.
0	Digit placeholder. Displays a digit or a zero. If the expression has a digit in the position where the zero appears in the format string, display it; otherwise, displays a zero in that position. If the number has fewer digits than there are zeros (on either side of the decimal) in the format expression, displays leading or trailing zeros. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, rounds the number to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, displays the extra digits without modification.
#	Digit placeholder. Displays a digit or nothing. If the expression has a digit in the position where the **#** character appears in the format string, displays it; otherwise, displays nothing in that position. This symbol works like the **0** digit placeholder, except that leading and trailing zeros aren't displayed if the number has fewer digits than there are **#** characters on either side of the decimal separator in the format expression.

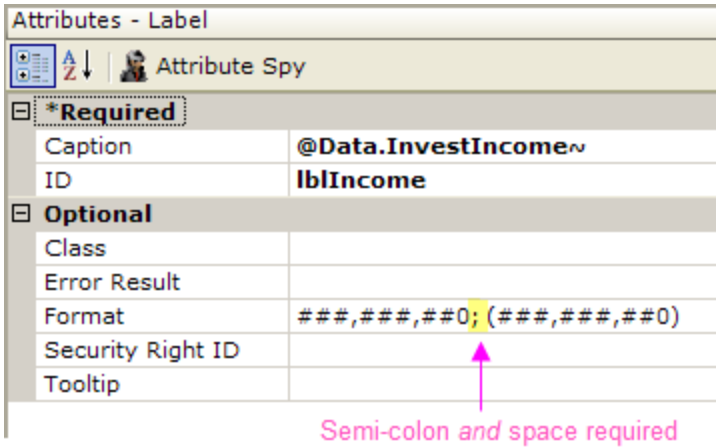
Chars	Description
.	Decimal placeholder. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only **#** characters to the left of this symbol; numbers smaller than 1 begin with a decimal separator. To display a leading zero displayed with fractional numbers, use zero as the first digit placeholder to the left of the decimal separator. In some locales, a comma is used as the decimal separator. The actual character used as a decimal placeholder in the formatted output depends on the number format recognized by your system. Thus, you should use the period as the decimal placeholder in your formats even if you are in a locale that uses a comma as a decimal placeholder. The formatted string will appear in the format correct for the locale.
%	Percent placeholder. Multiplies the expression by 100. The percent character (**%**) is inserted in the position where it appears in the format string.
,	Thousand separator. The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator. Standard use of the thousand separator is specified if the format contains a thousand separator surrounded by digit placeholders (**0** or **#**). A thousand separator immediately to the left of the decimal separator (whether or not a decimal is specified) or as the rightmost character in the string means "scale the number by dividing it by 1,000, rounding as needed." For example, you can use the format string `"##0,."` to represent 100 million as 100,000. Numbers smaller than 1,000 but greater or equal to 500 are displayed as `1`, and numbers smaller than 500 are displayed as `0`. Two adjacent thousand separators in this position scale by a factor of 1 million, and an additional factor of 1,000 for each additional separator. Multiple separators in any position other than immediately to the left of the decimal separator or the rightmost position in the string are treated simply as specifying the use of a thousand separator. In some locales, a period is used

Chars	Description
	as a thousand separator. The actual character used as the thousand separator in the formatted output depends on the Number Format recognized by your system. Thus, you should use the comma as the thousand separator in your formats even if you are in a locale that uses a period as a thousand separator. The formatted string will appear in the format correct for the locale.
E- E+ e- e+	Scientific format. If the format expression contains at least one digit placeholder (**0** or **#**) to the left of **E-**, **E+**, **e-**, or **e+**, the number is displayed in scientific format and `E` or `e` is inserted between the number and its exponent. The number of digit placeholders to the left determines the number of digits in the exponent. Use **E-** or **e-** to place a minus sign next to negative exponents. Use **E+** or **e+** to place a minus sign next to negative exponents and a plus sign next to positive exponents. You must also include digit placeholders to the right of this symbol to get correct formatting.
- + $ ( )	Literal characters. These characters are displayed exactly as typed in the format string. To display a character other than one of those listed, precede it with a backslash (****) or enclose it in double quotation marks (" ").
\	Displays the next character in the format string. To display a character that has special meaning as a literal character, precede it with a backslash (****). The backslash itself isn't displayed. Using a backslash is the same as enclosing the next character in double quotation marks. To display a backslash, use two backslashes (****). Examples of characters that can't be displayed as literal characters are the date-formatting and time-formatting characters (**a**, **c**, **d**, **h**, **m**, **n**, **p**, **q**, **s**, **t**, **w**, **y**, **/**, and **:**), the numeric-formatting characters (**#**, **0**, **%**, **E**, **e**, comma, and period), and the string-formatting characters (**@**, **&**, **<**, **>**, and **!**).

Visit the MicrosoftCustom Numeric Formats page for more information.

Positive and Negative Number Formatting

To construct a custom format for currency or numbers that treats positive and negative numbers differently, use a semi-colon and a space in the Format attribute to delineate two formats:



For example, the Format value shown above will show positive numbers with a comma as a thousands separator. It will show negative numbers in the same way but also enclose them in parenthesis. Note the semi-colon *and* space which separate the positive and negative formats.

You can also use the **Conditional Class** element, beneath any Label element that displays numeric data, to display negative numbers in a different color than positive numbers.

Formatted Column Element

A special element, the **Formatted Column** element, allows developer to format data right in a datalayer. The element formats data from another column and places it in a **new column** in the datalayer. All of the formatting options previously discussed here are available in this element and using the Formatted Column can produce report performance improvements.

For more information, see *The Formatted Column*.

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.