

Ignition 7.9 User Manual — Reporting and SQL Bridge

Generated 2026-10-12 from
<https://www.docs.inductiveautomation.com/docs/7.9/>

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

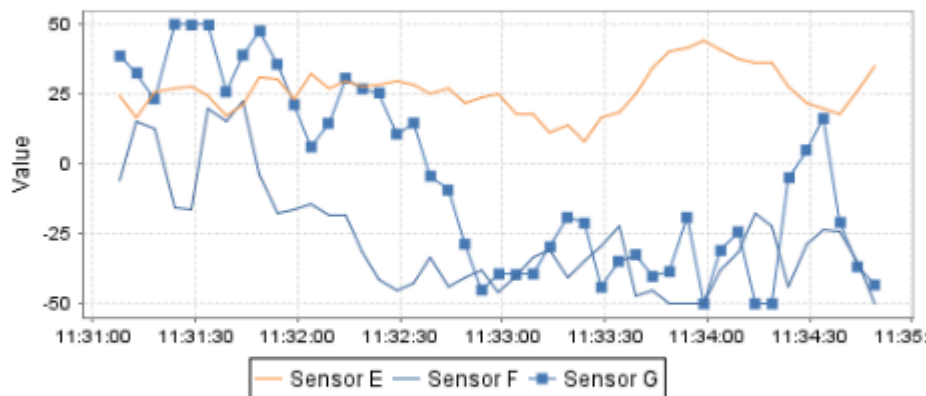
Reporting

Welcome to Ignition Reporting!

Ignition Reporting is a module for creating dynamic reports! The Reporting Module allows you to create your own custom reports on the fly or generate them based on a schedule. Data is introduced through Ignition, providing access to any SQL database!

Ignition Reporting makes creating professional reports easy with a rich library including: images, graphs, tables, and a variety of basic shape tools. The Report Scheduler allows automatic report generation at any time you'd like, and automated distribution means it gets where you need it without lifting a finger. Access to live reports is available through the web based Ignition runtime, a Java application, providing authenticated users access from anywhere, all based on networking standards that your IT department can support. Reports are printer friendly and can easily be exported to a variety of formats including .PDF! Here are some common uses of dynamic reports:

- Production Management
- Efficiency Monitoring
- Downtime Tracking
- Statistical Process Control
- Quality Assurance
- Overall Equipment Effectiveness (OEE) Management
- Historical Data Analysis



Index	Sensor E	Sensor F	Sensor G
1	24.23	-5.53	38.64
2	16.57	14.95	32.55
3	25.91	12.88	23.36
4	27.21	-15.81	50
5	27.53	-16.42	49.75
6	24.69	19.99	49.78

Benefits

Ignition Reporting enables managers to increase productivity, decrease waste, reduce costs, and increase quality with existing resources by providing a view of their manufacturing process. Managers often save time by automating reporting processes that were once done by hand. Often valuable man-hours that went into creating spreadsheets or reports can be recovered! These reports are trivial to manage since they are generated on the fly from existing SQL database data.

Features

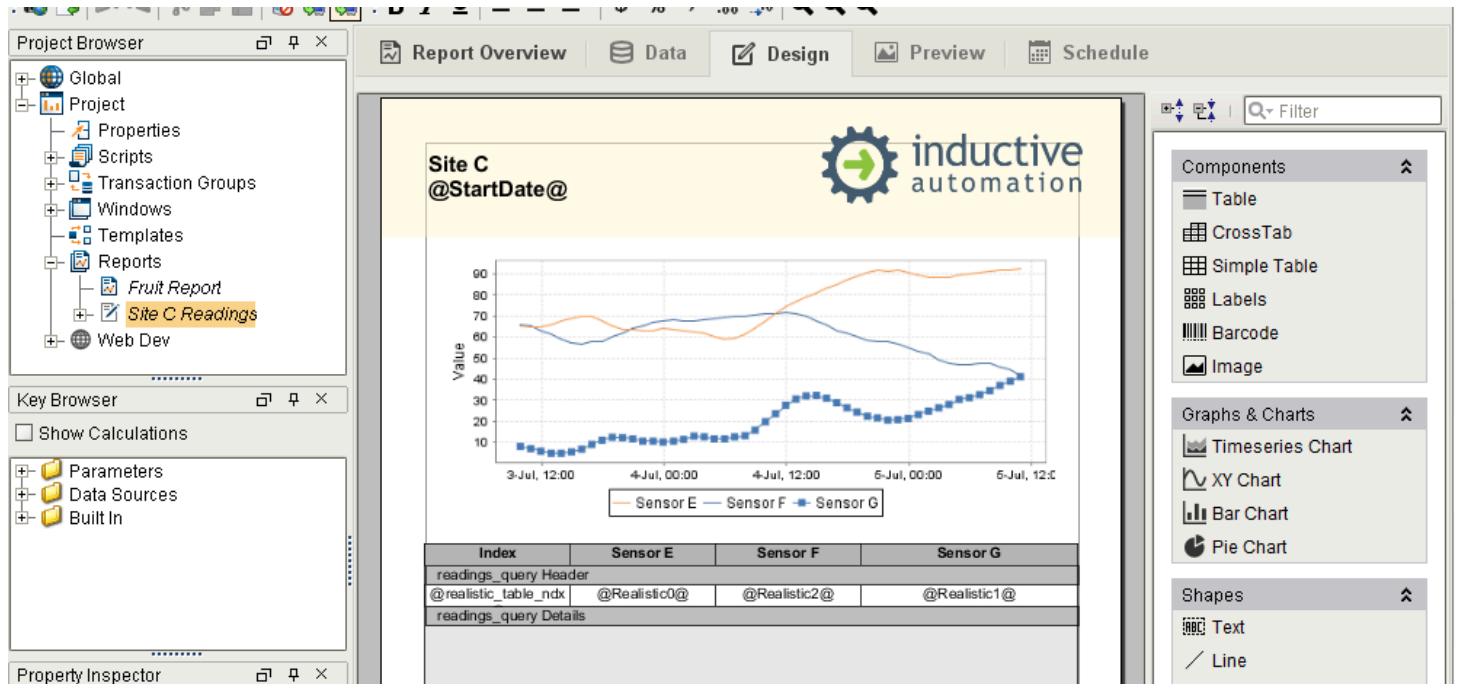
Here are some of the other innovative features of the Reporting Module.

- Report Engine
- Report Workspace
- Reports as Project-level Resources
- Powerful Data Collection Utilities
- Drag-and-Drop Query Builder
- Charting Components
- 2D Barcode Generation
- Powerful Scheduled Report generation

- Familiar Property Editing
- Scheduled Distribution Actions to Email, Save, FTP and More!
- Scripting Capabilities

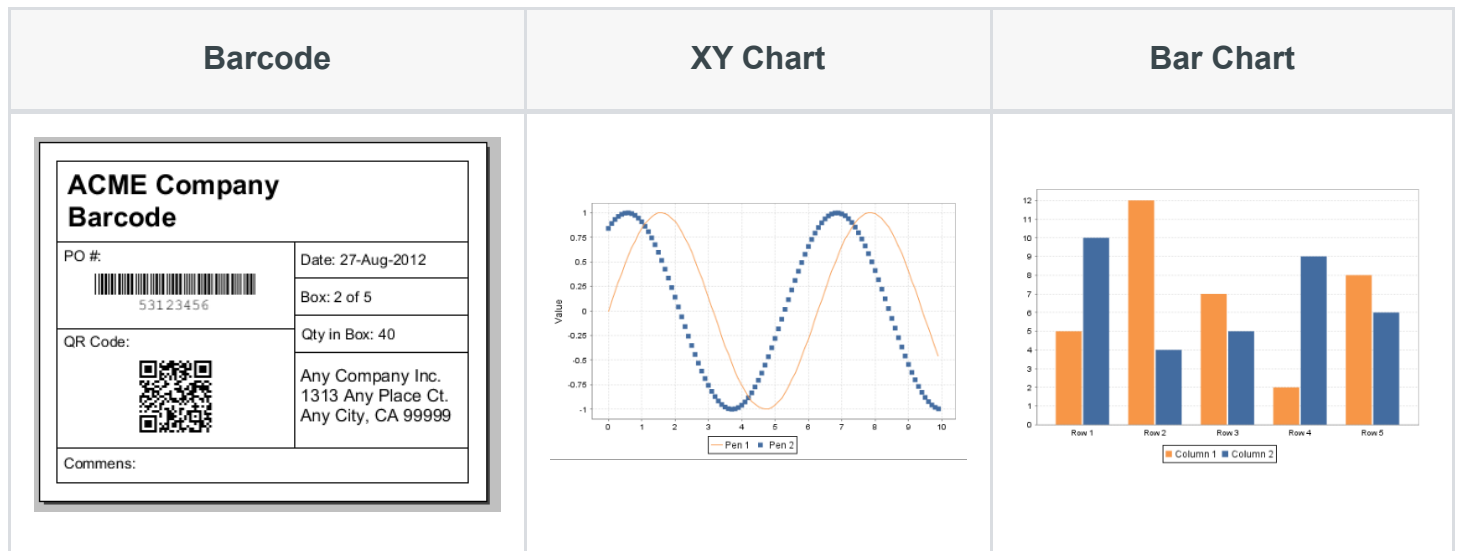
Intuitive Report Design

The **Report Designer** makes report generation feel just like designing a Vision window, and includes intuitive layout and drawing tools. Tricks and learned-behaviors from Vision windows are easily transferable to reports.



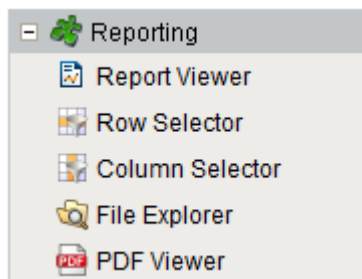
Powerful Components

A host of report-specific components unleash a wealth of possibilities as shown below. **Tables** can dynamically add pages to account for varying amounts of data, or change appearance based on certain values. Beautiful charts can visualize data. There are even Label and Barcode components so you can create shipping and product labels.



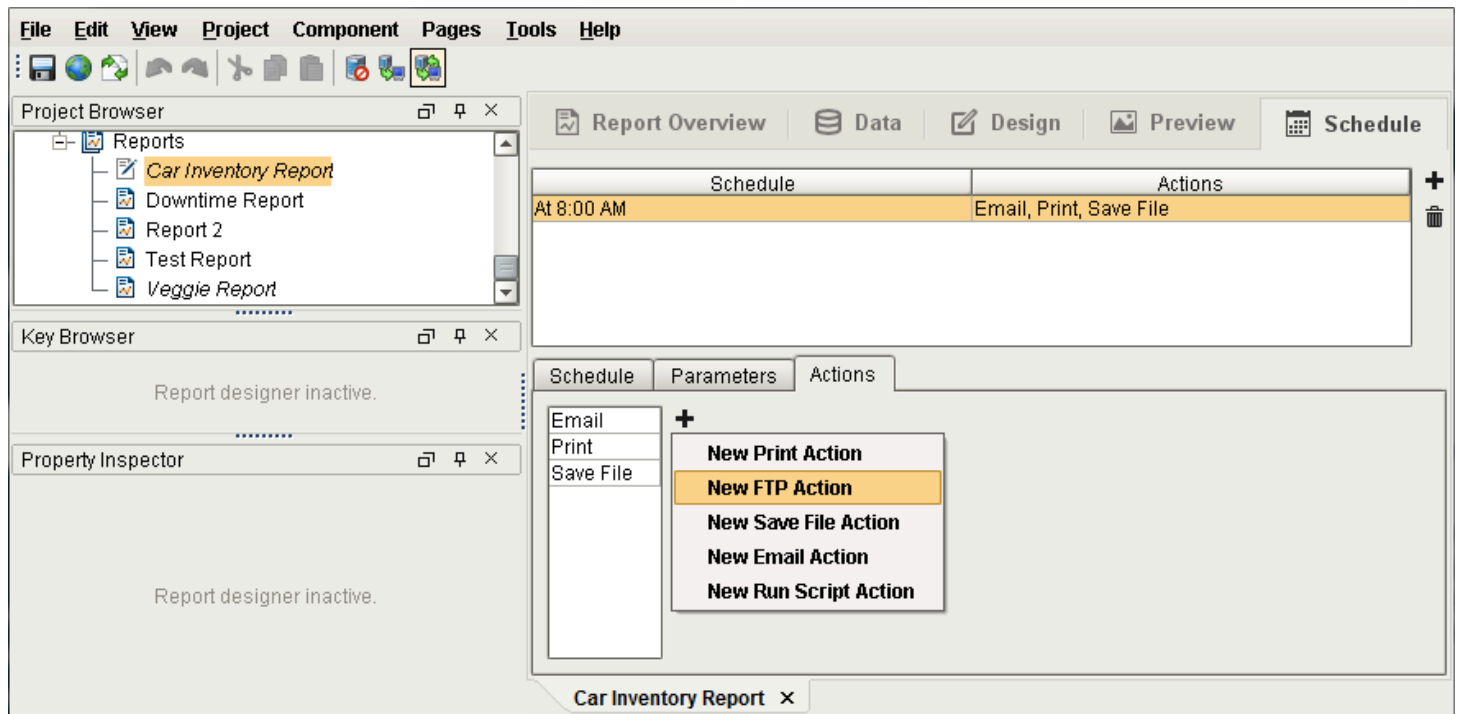
Reporting Module Components that can be used with the Vision Module

The Reporting Module provides several components that can be used with the Vision module. The **Row Selector** and **Column Selector** components aid with dataset manipulation, while the **File Explorer** and **PDF File Viewer** components allow clients to access files outside of Ignition. There is even a **Report Viewer** component so reports can be viewed directly from the client.



Scheduled Report Execution

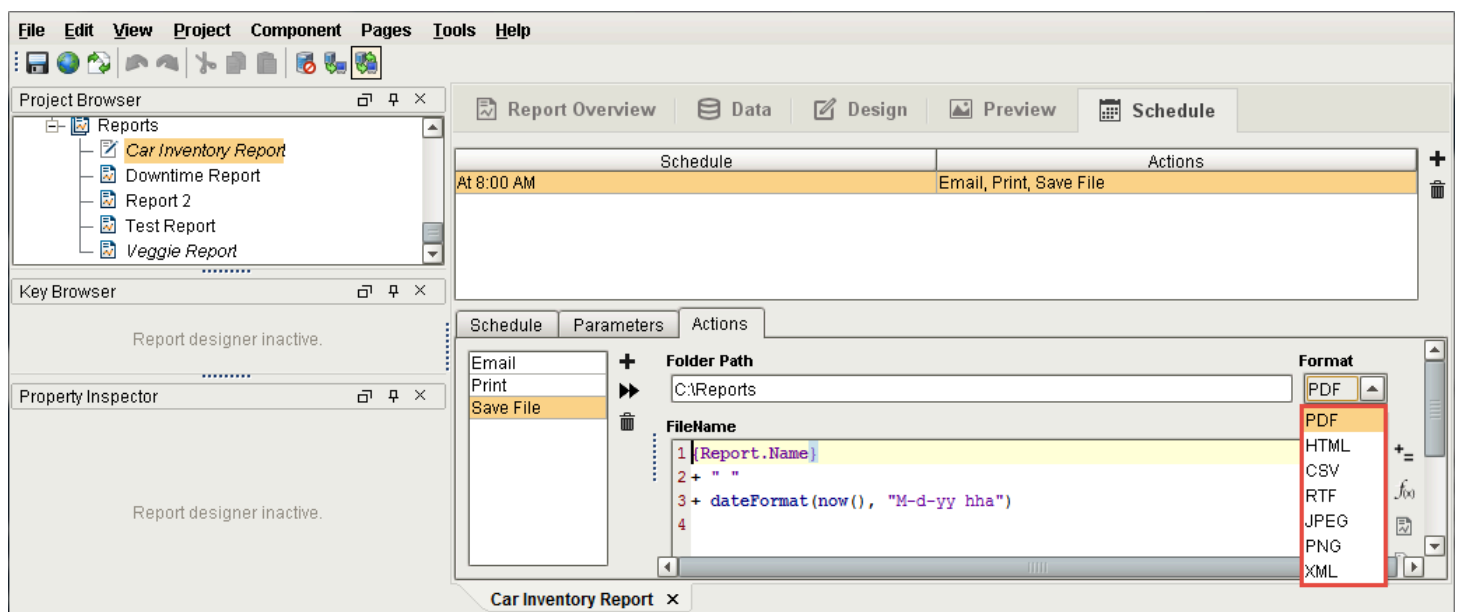
The **Schedule Engine** allows you to schedule when reports run, and what to do with the report when it's complete such as Email, Print, or Save as a File!



Supports Multiple File Formats

Reports can be generated in the following formats: CSV, HTML, JPEG, PDF, PNG, RTF, and XML.

New in 7.9.5 As of 7.9.5, reports may now be exported using the XLS format.



Using the Report Module

On the Vision Module

- A report that you create with the Reporting Module can be displayed using the Vision component either in the Designer or the Client.

- You can print and save a report through scripting. You can also right click on the component to print or save which is very useful in the Client. To learn more, refer to the [Report Viewer](#) in the Appendix.

On the Trial Version

- All Report functionality is available.
- There is no limit on the number of reports or how you can view or distribute them.
- The only limitation is the watermark on each page.

Legacy Reports

To take advantage of Ignition's powerful platform, we introduced a new version of the Reporting Module. You may see reports created by an older Reporting Module version in your project. If you need to modify an existing report, you'll still have access to the same customizer that you always had, double-clicking on them to open up their editor. They will continue to work as they always have without any modification. To learn more about converting reports created by an older version of the Reporting Module, refer to the section on [Converting Legacy Reports](#).

We recommend that you convert your older reports to the latest version to take advantage of the many features of the Reporting Module.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Reporting Basics

Reporting Overview

Ignition takes advantage of its powerful platform to make designing, creating, and distributing Reports easier than ever. Reports run on the Gateway, and have that same look and feel as the Vision Module. Reports even have their own workspace with some of their own components, and powerful data collection utilities. We know that Reports are increasingly important in modern automation projects and the Reporting Module makes it easy for you to create engaging, informative reports that are easily tailored to fit your data. We look forward to seeing what you can build!

The screenshot displays the Ignition Reporting Module interface. The main workspace shows a report titled "Tank Report" with the subtitle "January Compliance Report". The report includes a header section with a table of data and a large image of a tropical landscape. The table has the following structure:

Humidity	Temperature	Level	Time
@ambientHum@	@ambientTemp@	@receiverLevel@	@t_stamp@

The interface also features a Project Browser on the left showing the report structure, a Tag Browser with a list of tags, a Key Browser, and a Property Inspector at the bottom. The right sidebar contains a Components palette with options like Table, CrossTab, Simple Table, Labels, Barcode, and Image, as well as a Shapes palette with options like Text, Line, Rectangle, Ellipse, Star, Polygon, and Pencil. The status bar at the bottom indicates "Project saved. (824 ms)" and "100% 134 / 455 mb".

Reporting Interface



INDUCTIVE
UNIVERSITY

Reporting Interface

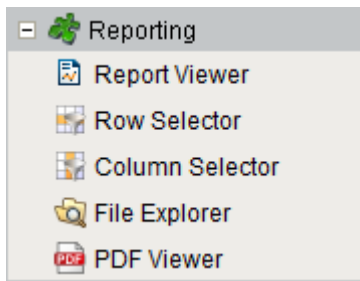
[Watch the video](#)

The **Reporting interface** was designed with the same look and feel as the Vision Module. Once you install the Report Module, you'll be able to open the Ignition Designer and see a Reports resource folder. Since Reports are a Project-Level Resource, it's easy to create a new report whether it's used in a window or not. With a right-click on the **Reports** node in the Project Browser and the option to create a new report is a click away.

From the first informative panel to full previews of the report you are creating, the Reports workflow will significantly cut the time it takes to design, edit, and distribute reports. We believe this Report workflow will be understandable to the new developer, yet powerful enough for a seasoned analyst. Make no mistake, reports have a lot of moving parts, but fear not, you'll be able to leverage these tools to create impressive reports faster than ever!

Reporting Components

Included with the Reporting module are many unique vision components, all of which are added into a new **Reporting** section of the Component Palette. The most important component is the Report Viewer, which allows you to view your reports within a Vision Client. The others, such as the **PDF Viewer and Row Selector**, all enhance the capabilities of your Clients.



Scripting in the Report Module

The Report Module allows you to use scripting in several ways to extend the existing functionality. You can use some or all of these to extend the flexibility of Ignition's Report module. If the existing builders aren't enough, Ignition offers scripting for you to create exactly the functionality you need.

- **Script Datasource:** There is a Script Data Source type that allows you to create your data set in any way you like. You can alter other Data Sources, combine results, do complex calculations and more.
- **Scheduling:** There is a Run Script action in the Report Scheduling system. If the existing print/email/save/FTP Actions aren't flexible enough, create a script to do exactly what you want with a generated report.
- **Scripting Functions:** The Report Module adds additional functions to the list of `system.report.*` functions available throughout Ignition. You can use these functions to execute and/or distribute reports on demand.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Report Designer Interface

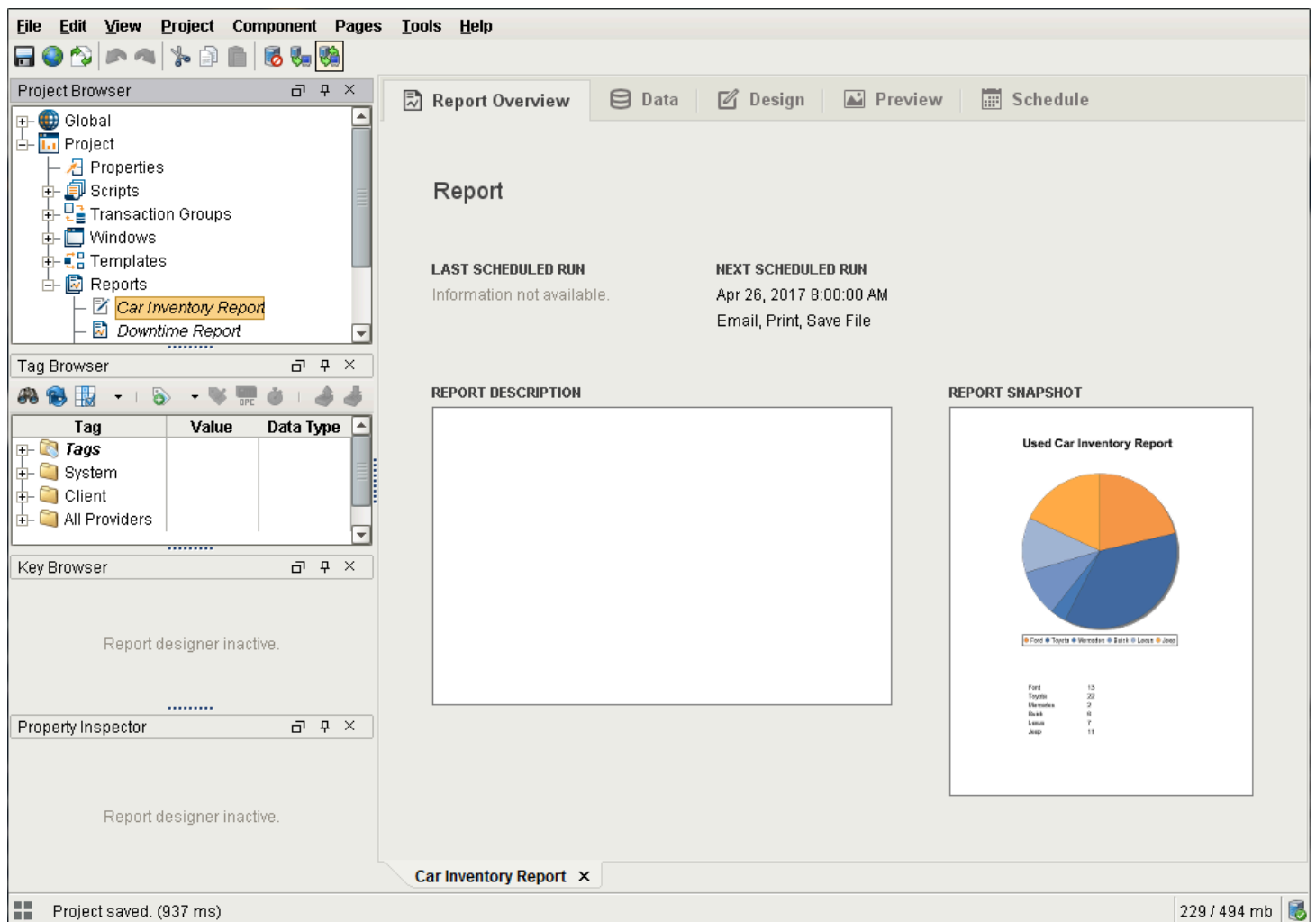
Reports Workspace and Workflow

Built from the ground-up to be intuitive and familiar, the Reporting workspace has a logical workflow which makes it easy for anyone to create meaningful reports. The reporting workspace has five panels which make up the workflow of a report: Report Overview, Data, Design, Preview, and Schedule panels.

Let's take a look at each panel of the Reporting workflow.

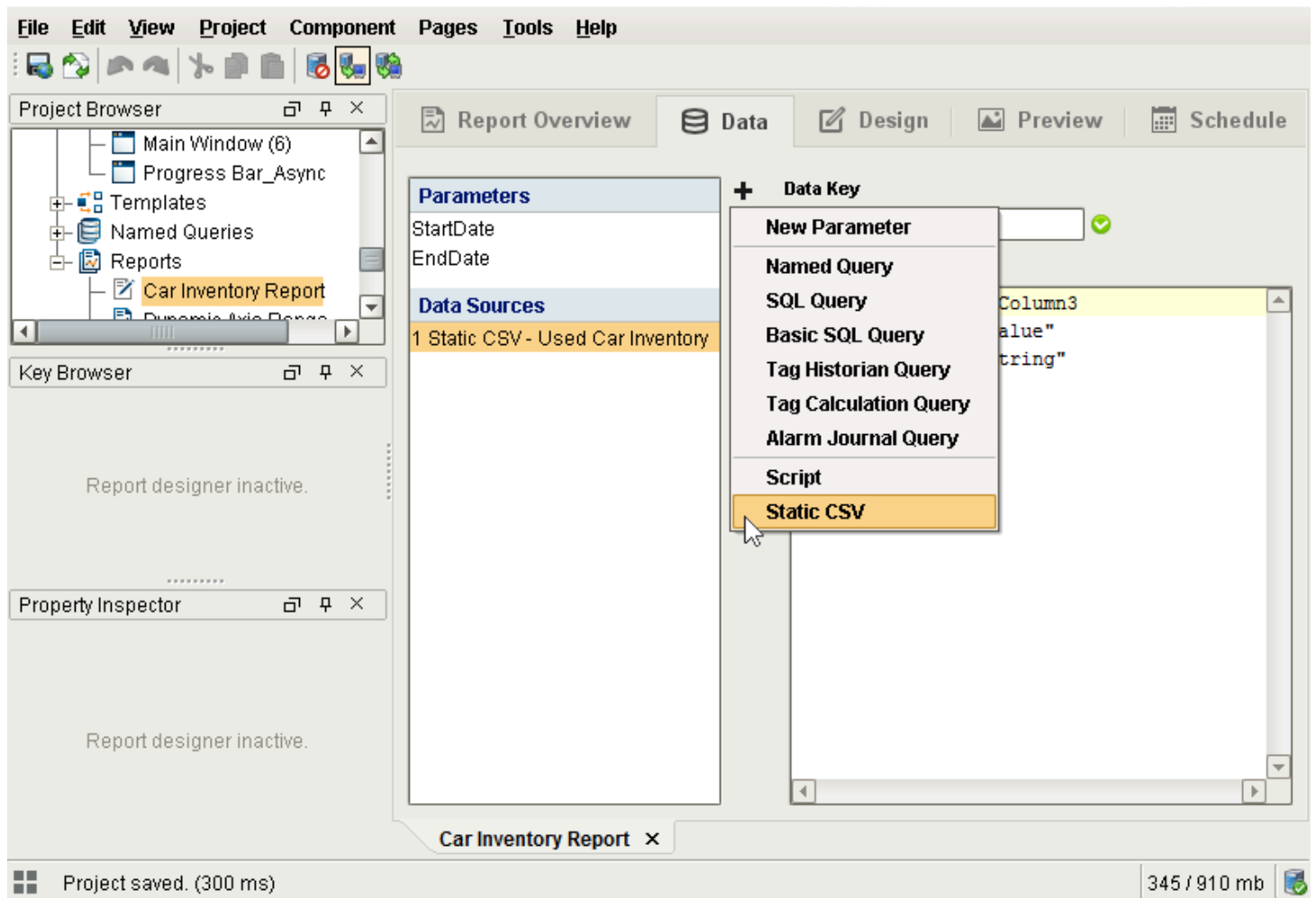
Report Overview Panel

The Report Overview panel should be fairly self-explanatory – it provides some valuable quick-glance information about your report. There is space to add notes, which is a good place to provide background information and context about the report. The Overview will also show a thumbnail of the most recent report, its last execution time, and the next scheduled execution. The Report Snapshot is generated per visit to the Preview Panel.



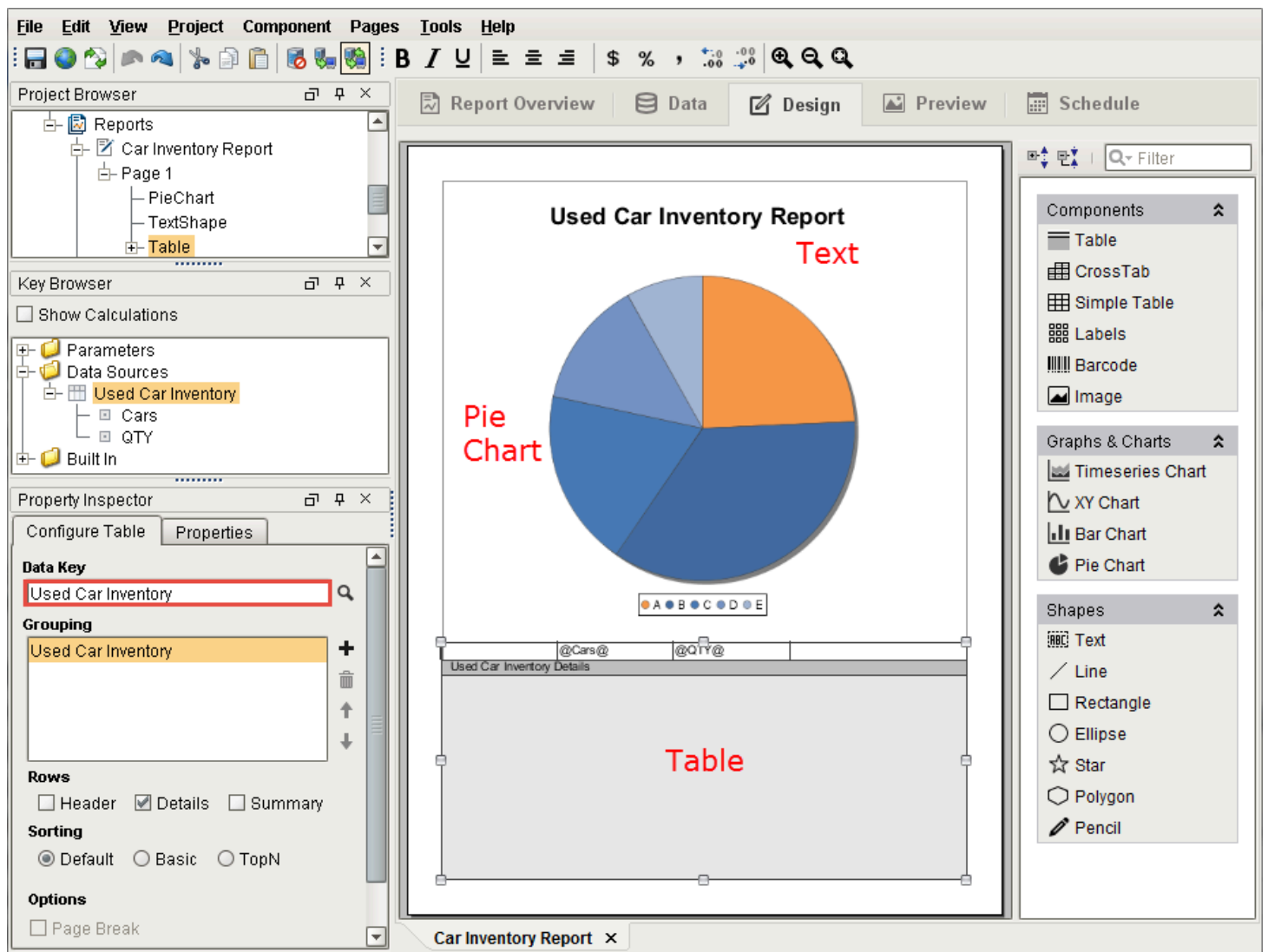
Data Panel

The **Data panel** is the first step for configuring a new report. You'll notice the list on the left that has labels for Parameters and Data Sources, as well as some buttons. On this page you can generate data that you want to make available on your report. We'll dive more into Parameters and Data Sources, but for now go ahead and click the (+) icon. You'll see a list of built-in Data Sources and Parameters. Selecting any one of them will open the configuration screen for that Data Source type and allow you to start configuring the data you'll supply to your report.



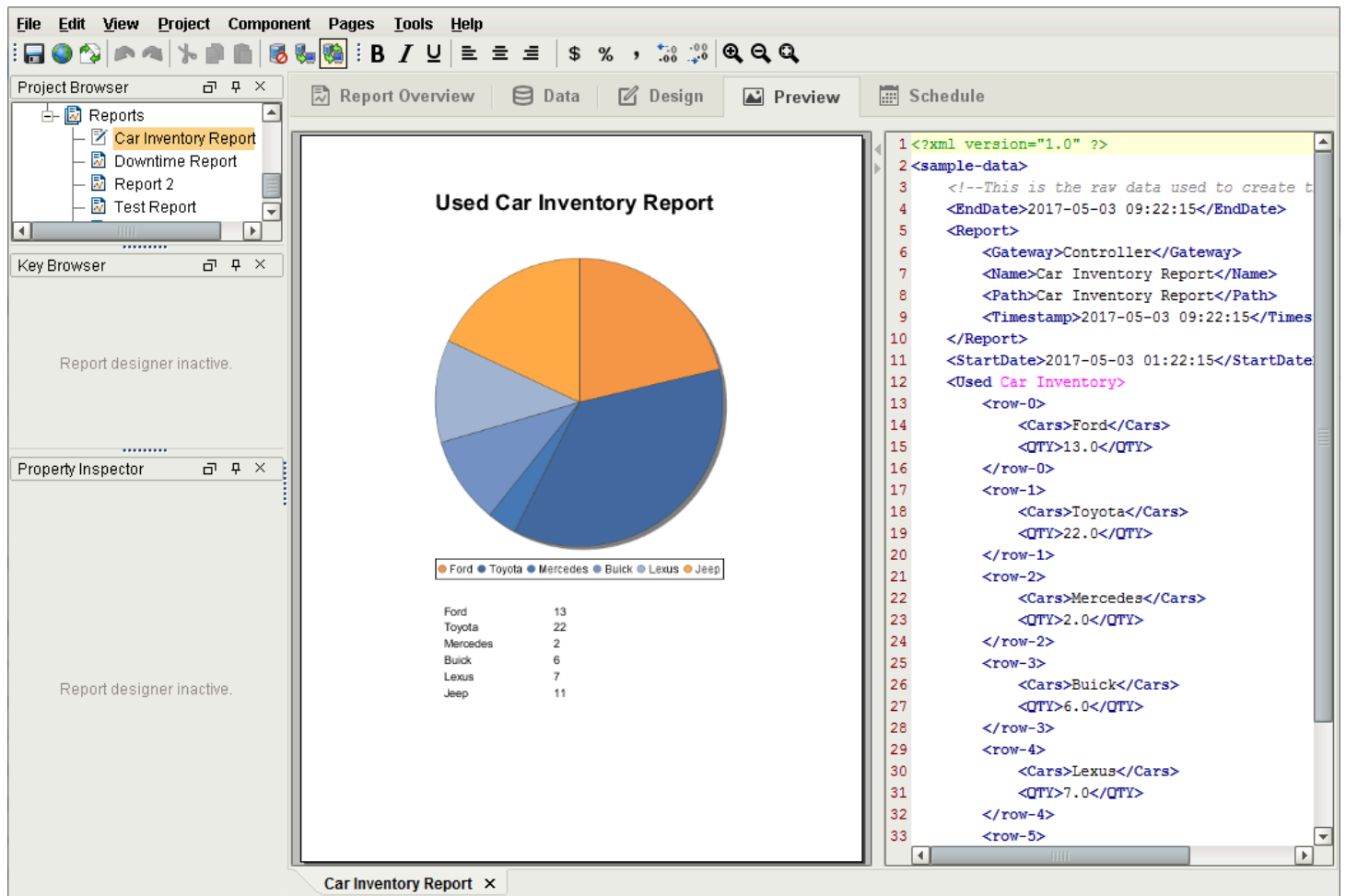
Design Panel

If you've ever designed a Vision Window, the **Design panel** should make you feel right at home. Starting from the top-left – the structure of a Report is now represented in the **Project Browser** tree, allowing easy visualization of the tree of elements on your report, but don't stop there! Grab a component off the vision-like reporting palette on the far right and add it to the page. For complex components, there are configuration panels in the **Property Inspector** to supplement the Property Inspector table that all the design objects have. In place of the Tag Browser, there is a **Key Browser** that gives you an easy way to add the Data Sources and Parameters that you configured in the Data panel to the Report.



Preview Panel

The Preview panel, while not one of our three design steps, is a huge help while building your report. This panel provides quick visual feedback on your report. Not only does it provide you an instant example of what your report looks like, but it also gives you the ability to view the actual data (on the right) that is being sent to your report! Having a snapshot of your data can be incredibly helpful when trying to figure out why your report doesn't appear as you expect it to. Whether you have an error in your query, or a bad Data Key in a component, you'll be able to figure out the structure of the data your report is getting and quickly find ways to solve any problems.



Schedule Panel

Scheduling a report to run is built-in and easy to use. The Schedule panel is an incredibly powerful way to automate the execution and delivery of your reports. Whether you have extremely long running queries that you don't want tying up a window, need to deliver to multiple file servers via FTP, or want to simply email to an existing roster, this panel provides you the means to do it. It's broken up into two major areas. On top, you find a simple table of schedules that have been established. To add a schedule, click the (+) icon. With a schedule added, you can now choose when and how that scheduled Action will occur. Our schedules use the common Chron format, but don't worry if you aren't familiar with it. The preset options and combo-box configurability allow you to easily create just about any schedule you can imagine.

Once you selected your schedule, you can add parameters that may affect your Actions. Actions are incredibly powerful and allow you to choose how you want to distribute or act on your finished reports. Whether saving a file locally, posting to an FTP server, or emailing, you can get your report where it needs to be automatically. If our normal distribution Actions somehow fall short, you can choose to act on the finished report by triggering a script! We've also given you a convenient way of immediately executing the Action.

File Edit View Project Component Pages Tools Help

Project Browser

- Global
- Project
 - Properties
 - Scripts
 - Transaction Groups
 - Windows
 - Templates
 - Reports

Key Browser

Report designer inactive.

Property Inspector

Report designer inactive.

Report Overview Data Design Preview Schedule

Schedule	Actions
At 8:00 AM	Email, Print, Save File

Schedule Parameters Actions

Email

- Print
- Save File

From Address

IgnitionReportServer@myDomain.com

Subject

1 {Report.Name}

Attachment Filename

1 {Report.Name}
2 + " - "
3 + dateFormat(now(), "M-d-yy ha")
4 + ".pdf"
5

Body

1 "Report attached"

Mail Server

SMTP

Format

PDF

Retries

1

Recipients Source

User Roles

User Source

default

Recipient Roles

Administrator

Veggie Report Car Inventory Report x Report

Project saved. (210 ms)

360 / 494 mb

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Using the Report Viewer Component

Report Viewer Overview

The Report Viewer component provides an easy way to view reports in Vision windows. Simply go to the Property Editor of the Report Viewer, and choose the report you want to view from the **Report Path** property dropdown list. Parameters added during report creation are provided as properties in the Report Viewer, and will override any default values set in the Report Resource. Scroll down to the bottom of the Property Editor and you'll see your report parameters available as properties. The Report Viewer allows you to bind your report parameters to other components on your window.

Report Viewer Properties

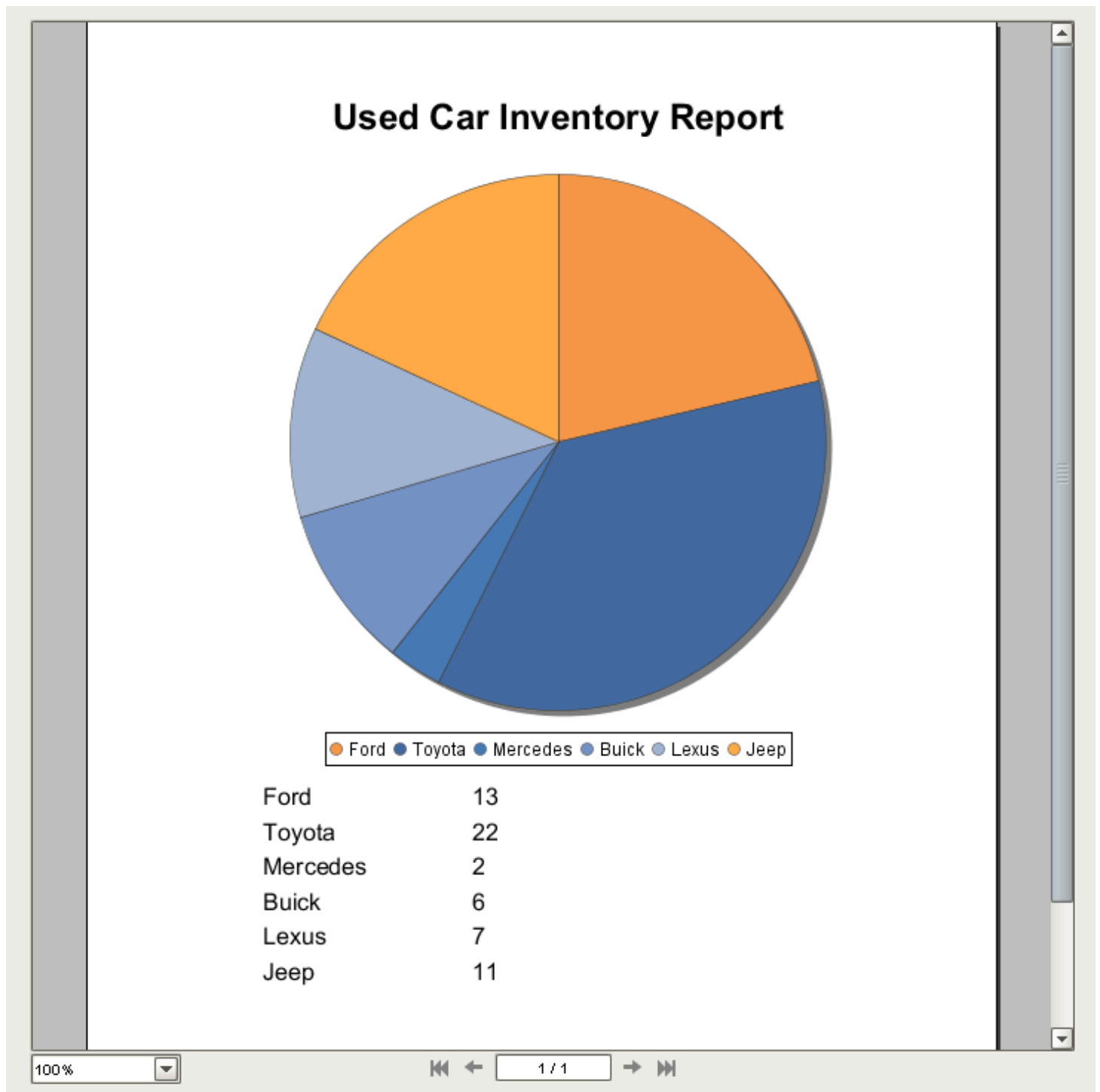
While you're in the Property Editor, take a moment to scroll through all the Report Viewer properties. You'll notice that you can customize the visual style of your report by adding page number, change the zoom factor, and the background colors of the report. More information on the [Report Viewer's properties](#) can be found in the the Appendix.

Using the Report Viewer

Once you have your reports designed, you're ready to view them in a Vision window using the Report Viewer component. (You need to have a report created before you can view it using the Report Viewer. For information on how to design a report, refer to [Report Design](#)).

- In the **Designer**, drag a **Report Viewer** component to a window.
- With the Report Viewer selected, go to the **Property Editor** and set the **Report Path** property to display any of your existing reports from the dropdown list.
- Scroll down to the bottom of the Property Editor, and you'll see **Report Parameters**. This sample report is using the **Start Date** and **End Date** parameters. (These are the default parameters that are available in the Data panel of your report). Choose the date and time for both Start and End Dates using the calendar next to each property, and the information in the report will update automatically.

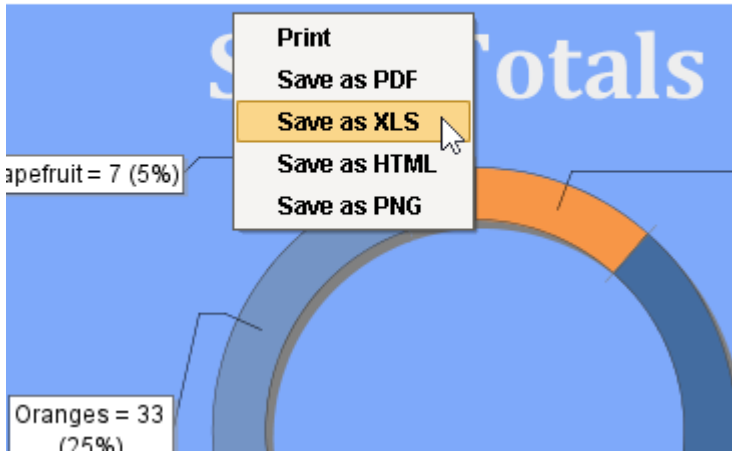
- You can even bind the Start and End dates to run daily at the same time (i.e., 9am to 5pm). If the time is outside of the 9am-5pm window, then you can use the calendar to select the time. You can even bind a parameter to another component on a window.



Save and Print a Report

In **Preview Mode** and from the **Client**, you can right click on the report and print it. You can even save the report to PDF, HTML, or PNG format.

New in 7.9.5 As of 7.9.5, the Report Viewer component can save the report as an XLS document.



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

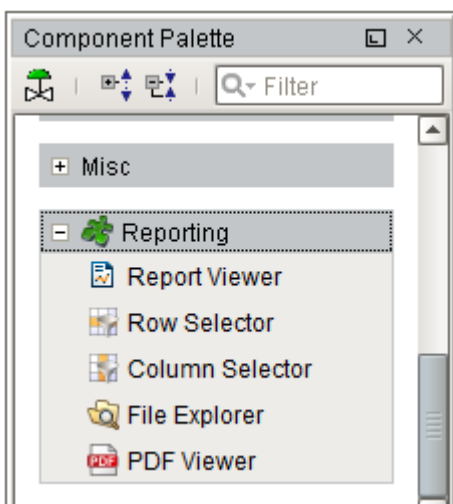
For documentation on the latest stable release, see the [latest version](#) (8.3).

Vision Reporting Components

The Reporting Module comes with some additional components that can be used with the Vision components to help you configure and filter your data, and view reports. These extra components are found in the Component Palette under Reporting: Report Viewer, Row Selector, Column Selector, File Explorer, and PDF Viewer.

NOTE

These components will not be visible on the Component Palette unless the Reporting Module is installed.



Each of these components are described below along with examples of how they can be used and how to configure them.

Reporting Module Components

Row Selector

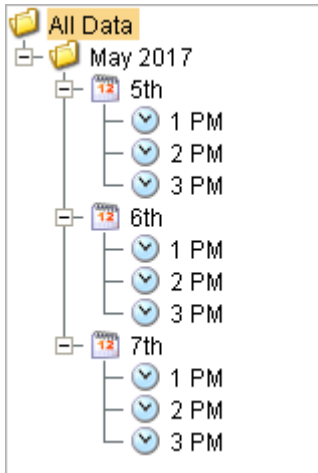


INDUCTIVE
UNIVERSITY

Row Selector

[Watch the video](#)

The **Row Selector** component allows users to filter a dataset based on unique values of one or more columns. Each level in the sorting tree is based on these properties. The user will see a dynamically generated expandable tree that groups their data by any number of choices. As a user clicks down the tree, objects bound to the dataset will indicate the filtered data.



A common way to filter the data is by time. For example, you can feed the Row Selector an input dataset that represents a large time range, and have it break it up by Year, Month, and Day. Then you can power a report with the output dataset that lets the user dynamically create reports for any time range. When configuring the Row Selector for the first time, you'll notice some default Date filters in the Row Selector Tree Customizer to help you quickly configure and filter raw data by time. If you don't want to filter your data by time, then simply delete the default filters and create your own.

See the Appendix for more information on the **Row Selector**.

Row Selector Examples

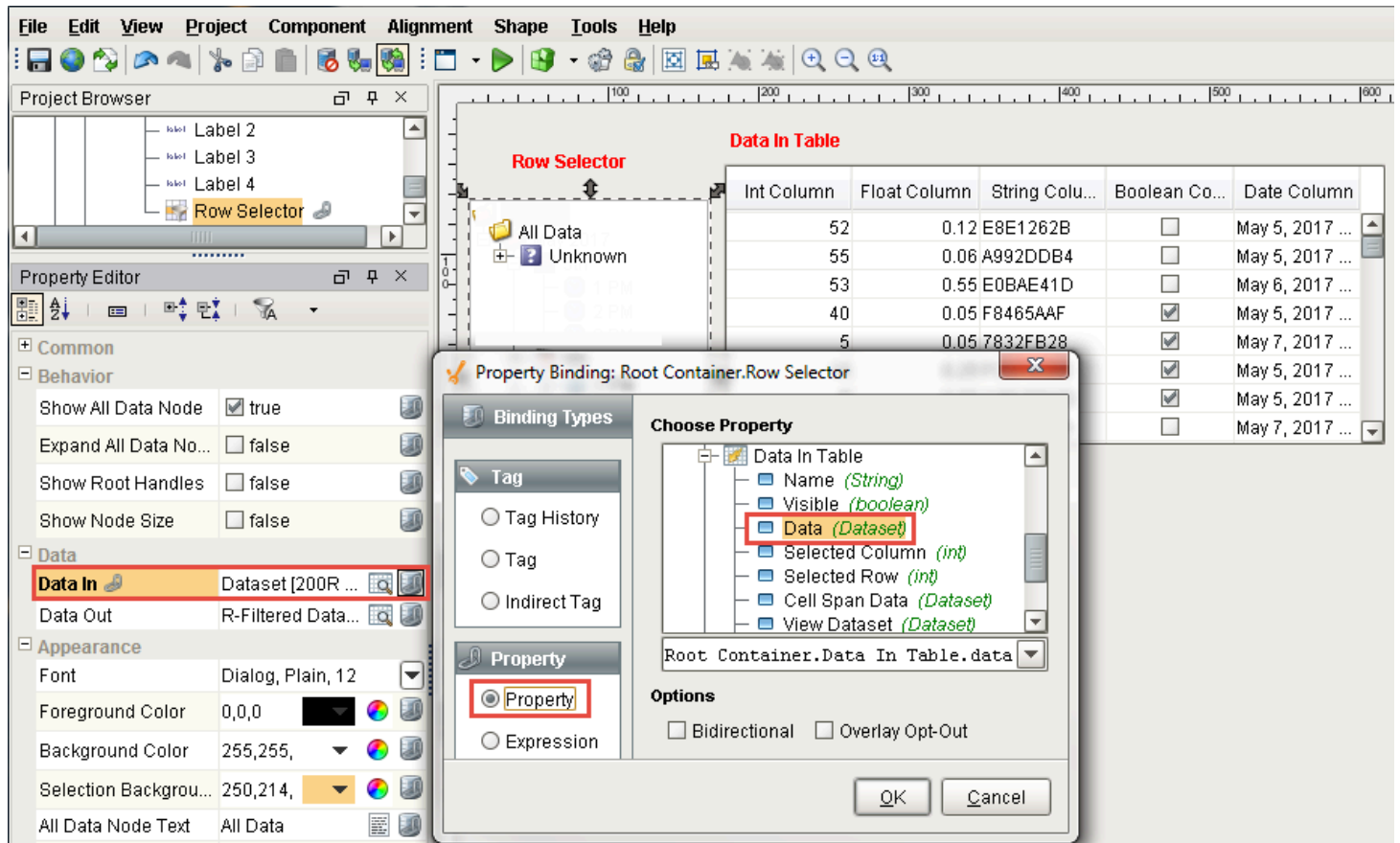
A Row Selector has two important properties: **Data In** and **Data Out**. The Row Selector component filters the data in the **Data In** property and pushes the filtered result to the **Data Out** property.

Let's configure a Row Selector to filter on some raw data. (This example uses a Power Table component which gives you the option of using some sample test data, or you can create your own data).

- Drag a Power Table component on to your window and change the **Name** of the Power Table to "**Data In Table**". Scroll down the Property Editor and toggle the **TestData** property. This will populate the **Data** property with some test data. Alternatively, you could manually populate the **Data** property using either the Dataset Viewer or by creating a binding on the property.

- Drag a Row Selector on to your window. With the Row Selector selected, click the binding icon on the Row Selector's **Data In** property, select the **Property** binding type and bind it to the **Data** property in the **Data In Table**, and click **OK**.

Data In Table

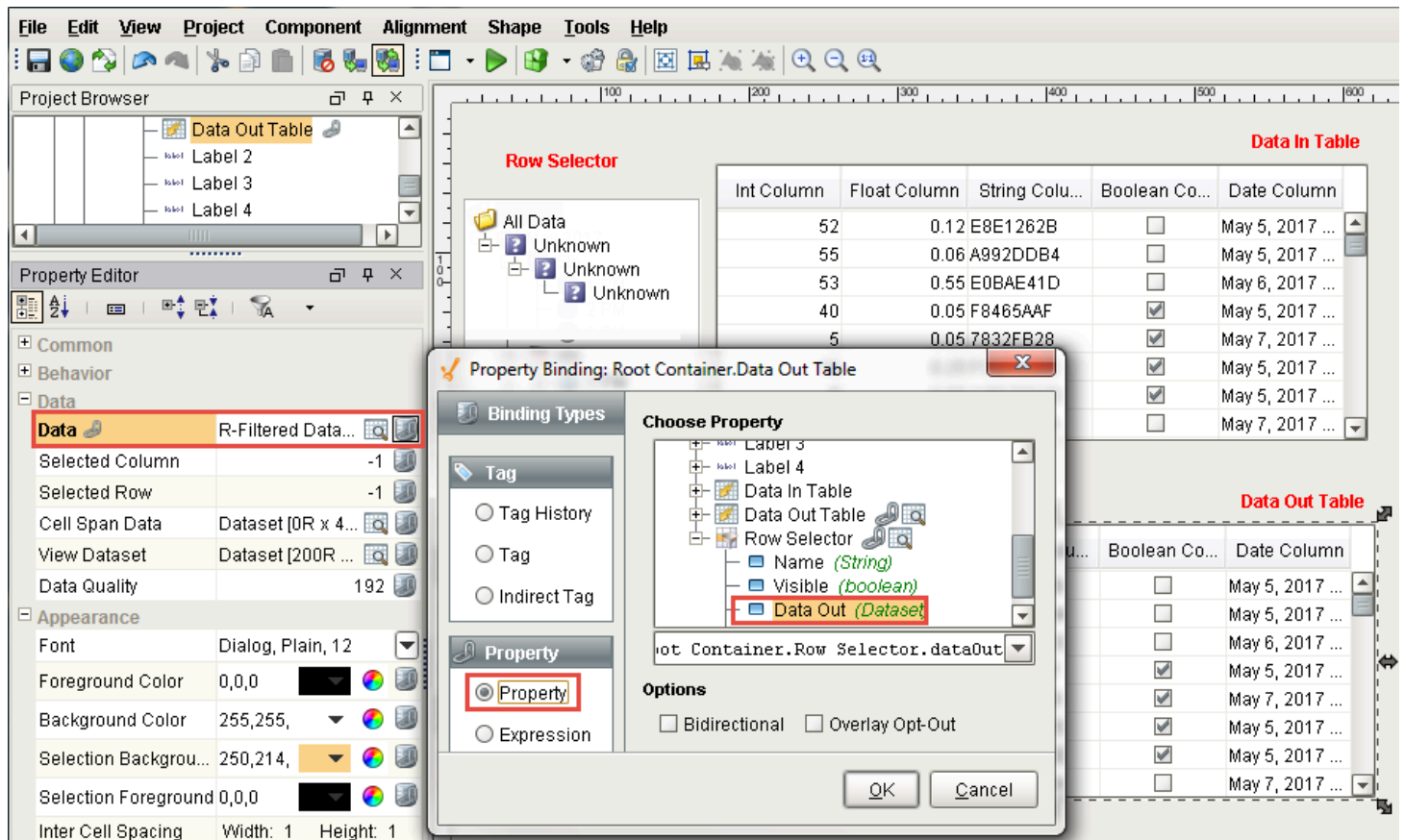


- Drag another Power Table component on to your window, change the Name to "**Data Out Table**", and bind the **Data** property to the **Row Selector Data Out** property, and click **OK**.

i NOTE

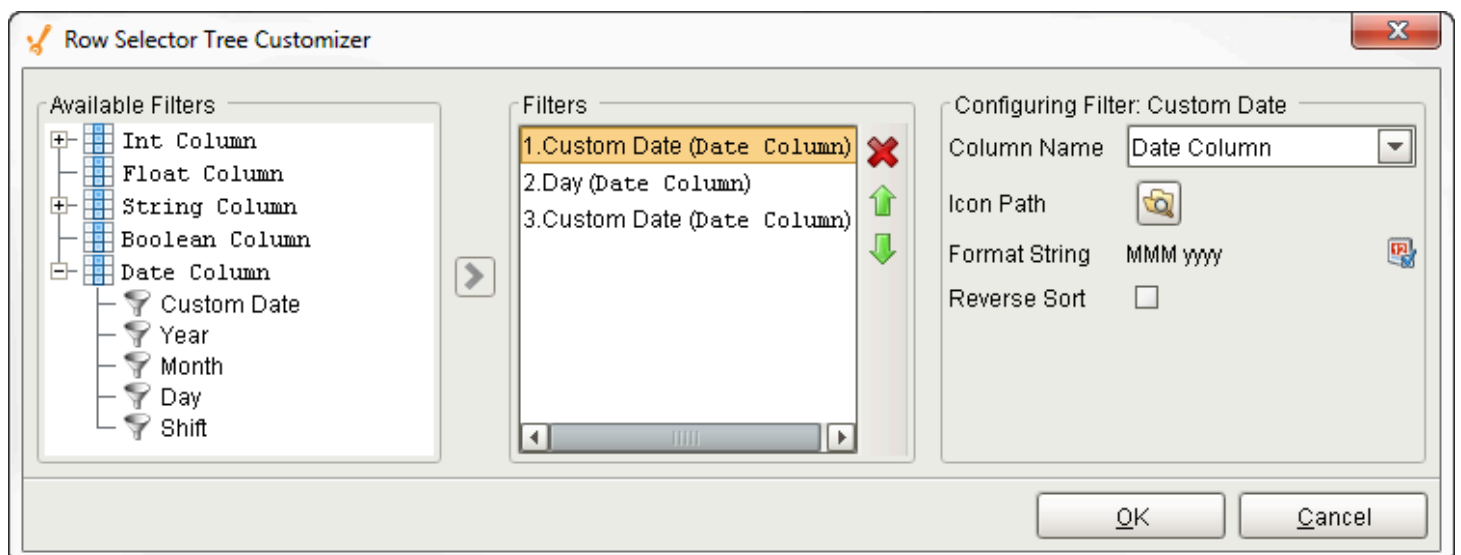
Don't worry about the 'Unknown' nodes in the Row Selector component, as shown in the example below. The Row Selector has three default Date filters which haven't been configured for your data yet. You will need to reconfigure them, or delete them and configure your own unique set of filters. Once configured, the 'Unknown' nodes will disappear.

Data Out Table



- Now let's configure your filters. Right click the Row Selector component and scroll down to **Customizers > Row Selector Tree Customizer**. The Row Selector Tree Customizer provides three default Date filters. Here you can customize the parameters of each Date filter or choose another filter type that is more appropriate for your dataset. Each column on the Data In Table will be listed in the Available Filters tree, and the types of filters available to each column depends on the datatype of the column. After configuring your filters, click **OK**. All your filter nodes will be displayed in the Row Selector.

This example uses the **Date** column to filter on the Month, Day, and Time combination.



- In **Preview Mode**, select Month, Day, or Time to filter on. The filtered results are displayed in the **Data Out Table**: 3 PM on May 6th 2017 is selected in the Row Selector, so the Data Out Table only contains records that match that datetime.

NOTE

When designing your report window, it's not necessary to display the Data In Table only the Data Out Table. Furthermore, you don't necessarily need a component to house the data: the Data In property on the Row Selector could simply retrieve the raw data with a binding.

Row Selector

Data In Table

Int Column	Float Column	String Column	Boolean Column	Date Column
56	0.59	65BB0F26	☐	May 5, 2017 2:20 PM
51	0.16	4C10B535	☒	May 5, 2017 2:20 PM
75	0.42	C1CA7FF4	☒	May 5, 2017 2:20 PM
31	0.7	2D614D34	☐	May 5, 2017 2:20 PM
26	0.79	5670D3BD	☒	May 5, 2017 2:20 PM
27	0.13	E87B9471	☒	May 5, 2017 2:20 PM
84	0.27	069A39B1	☒	May 5, 2017 2:20 PM
76	0.1	0FB708FE	☐	May 5, 2017 2:20 PM

Data Out Table

Int Column	Float Column	String Column	Boolean Column	Date Column
28	0.75	693E3C10	☒	May 6, 2017 3:20 PM
52	0.29	A88EC1E8	☐	May 6, 2017 3:20 PM
82	0.03	FE0145FA	☒	May 6, 2017 3:20 PM

Here are a few more Row Selector examples:

- A **Line Graph** bound to a Row Selector - Set up grouping to be first by month and year, then day, then hour, like the example above. Clicking on a month and year will dynamically update the graph for that time period. Further clicking to a specific day or hour will re-filter the graph for that period.
- A **Report Viewer** bound to a Row Selector - Grouping by department (String) would allow selection by department, automatically regenerating the Report on selection.
- An "Alarm History" Table bound to a Row Selector - This could first be broken down severity level (Integer), then broken into "Alarm Acknowledged" / "Not Acknowledged" (Boolean based). Clicking "Severity 3" would filter the table to all Severity 3 alarms. Selecting "Unacknowledged" would then filter the table to Unacknowledged alarms of Severity 3.

Column Selector

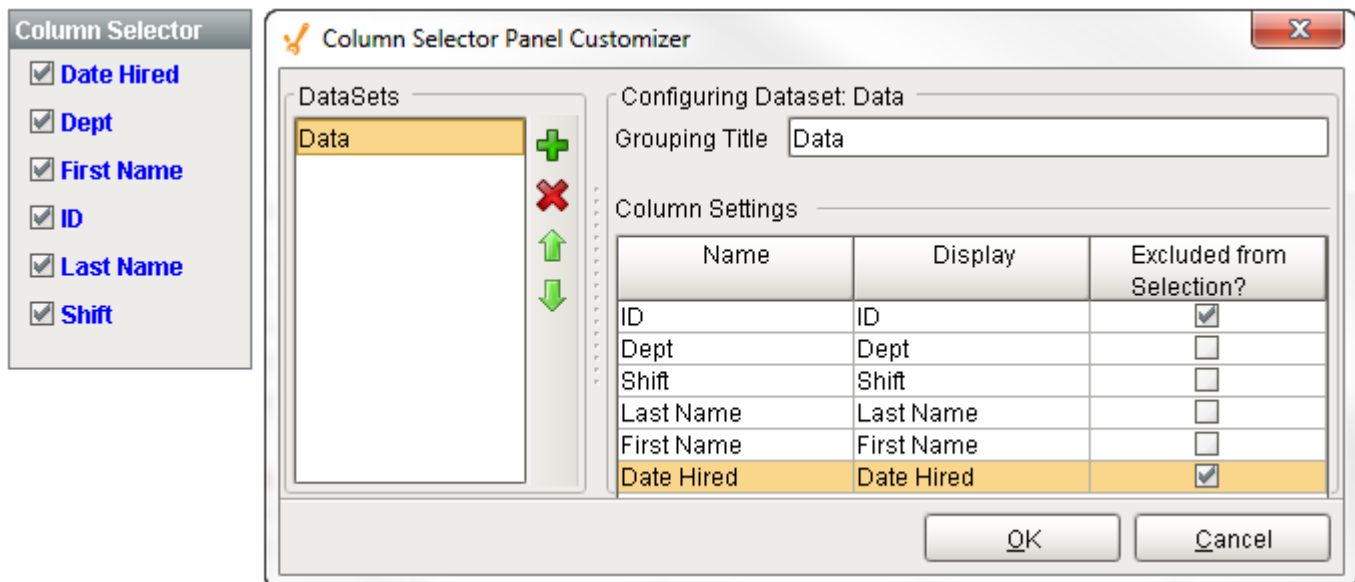


INDUCTIVE
UNIVERSITY

Column Selector

[Watch the video](#)

The Column Selector is similar to the Row Selector except that instead of filtering rows, it filters out entire columns from the output dataset. Each column from the input dataset is shown as a checkbox and allows users to show or hide variables in the datasets via the checkboxes, then outputs the resulting dataset. The Column Selector allows users to choose which columns in a dataset they wish to use. If an object is bound to the Column Selector it will update itself whenever a user checks or unchecks a column. This allows users to dynamically show or hide Table columns, "pens" on a graph, data in a [Report Viewer](#), or any other component set up to use a dataset.



See the Appendix for more information on the [Column Selector](#).

Column Selector Example

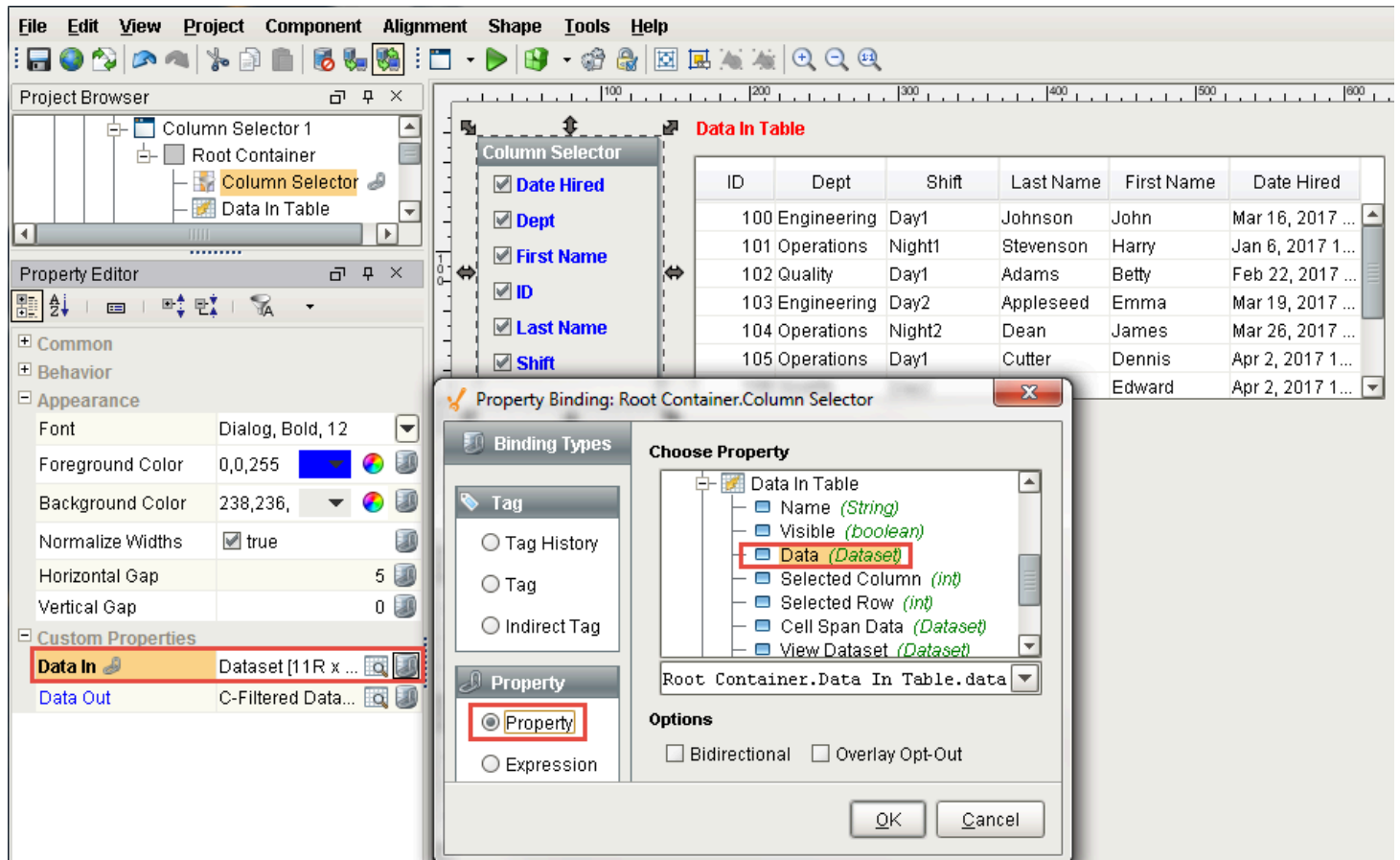
Just like the Row Selector, the Column Selector also has the same two important properties: **Data In** and **Data Out**. The Column Selector filters the data in the **Data In** property and pushes the filtered results to the **Data Out** property.

Let's configure a Column Selector to filter on some columns of employee data. (You can use a Power Table component which gives you the option of using some test data, or creating your own data like we did in the example below).

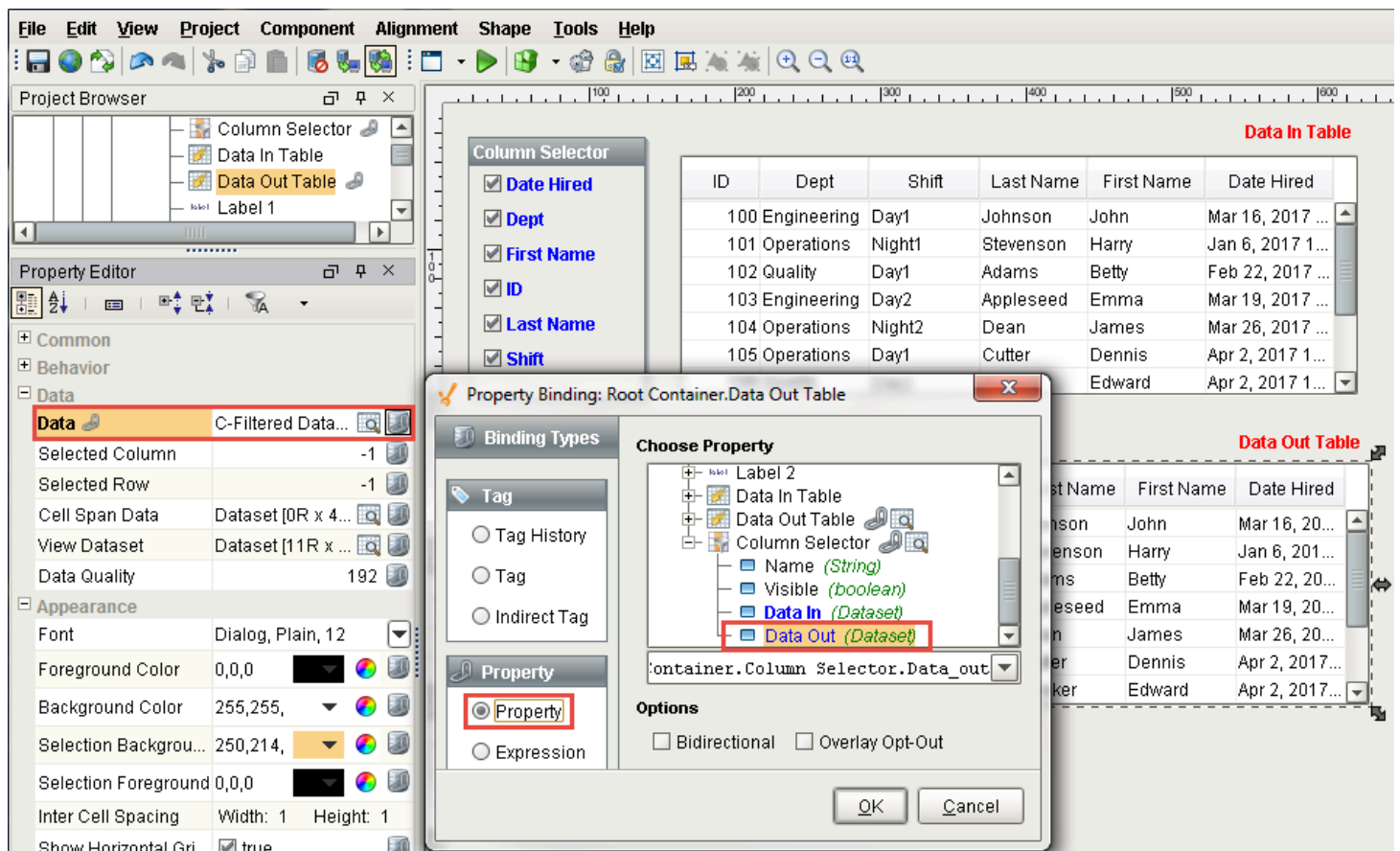
- Drag a Power Table component onto your window and change the **Name** of the Power Table to "**Data In Table**". Scroll down the Property Editor and toggle the **TestData** property. This will

populate the **Data** property with some test data. Alternatively, you could manually populate the **Data** property using either the Dataset Viewer or by creating a binding on the property.

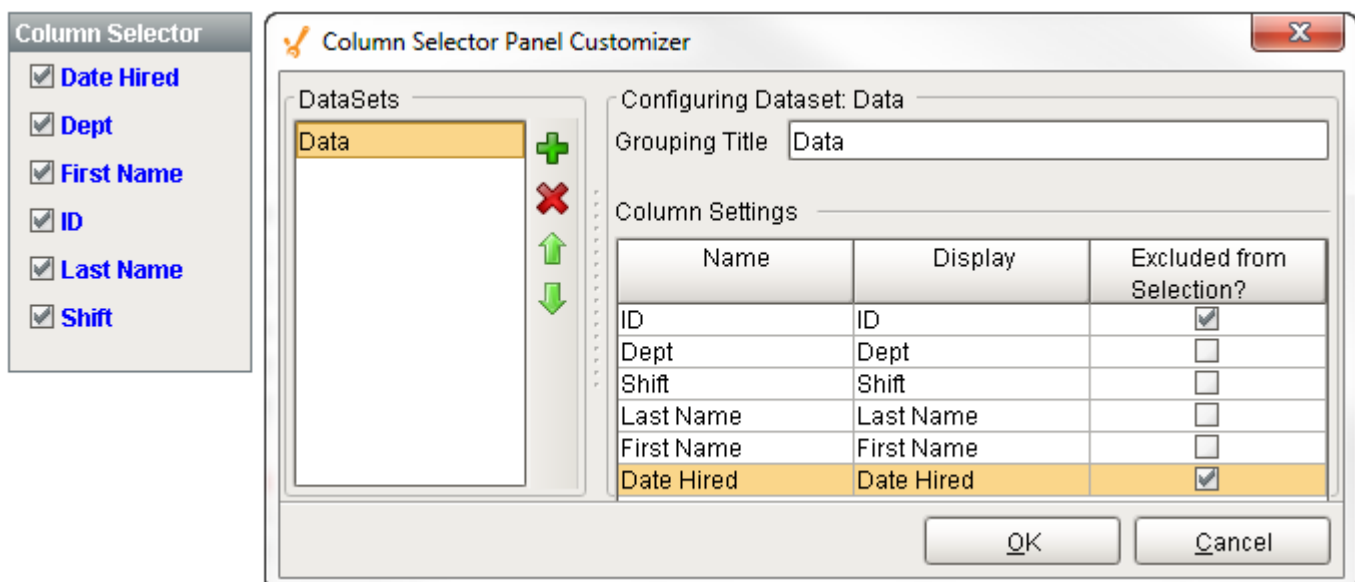
- Drag a Column Selector on to your window. With the Column Selector selected, click the binding icon on the **Data In** property at the bottom of the Property Editor, select the **Property** binding type and bind it to the **Data** property in the **Data In Table**, and click **OK**. The Column Selector will display a list of all the columns that are available in the dataset, and the Data In Table displays all the raw data.



- Drag another Power Table component on to your window, name this component **Data Out Table**, bind the **Data** property of this Power Table to the **Column Selector Data Out** property, and click **OK**. All the columns and raw data will be displayed in the Data Out Table.

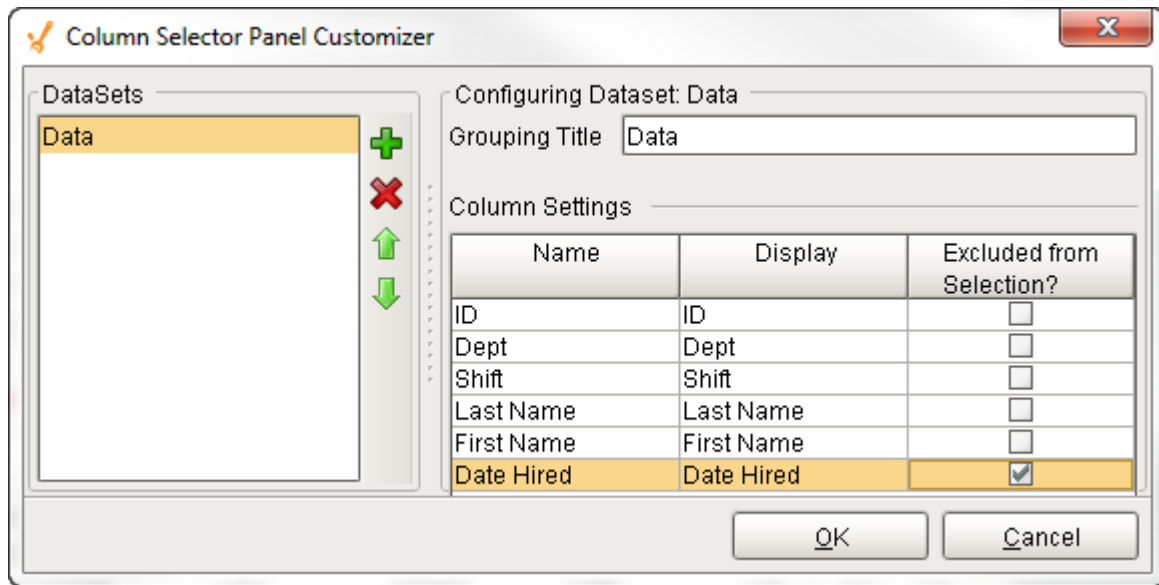


- In **Preview Mode**, you can hide specific columns by unchecking the box in the Column Selector. Conversely, if you want to bring a column into view, simply check the box. In this example the Date Hired and Shift columns were unchecked. You'll notice that these columns are omitted from the Data Out Table.



- The Column Selector component also has a way to hide columns that you don't want the user to deselect. In the **Designer**, right click the **Column Selector** and scroll down to **Customizers > Column Selector Customizer**. Click on **Data DataSet**, and you'll see a list of all the columns in the DataSet. If you check any of the columns under the **Excluded from Selection**, they will no longer show up on the Column Selector. It's important to note, that the Excluded from Selection

doesn't filter out the column, it simply determines if you are able to filter the data on that column or not.



PDF Viewer

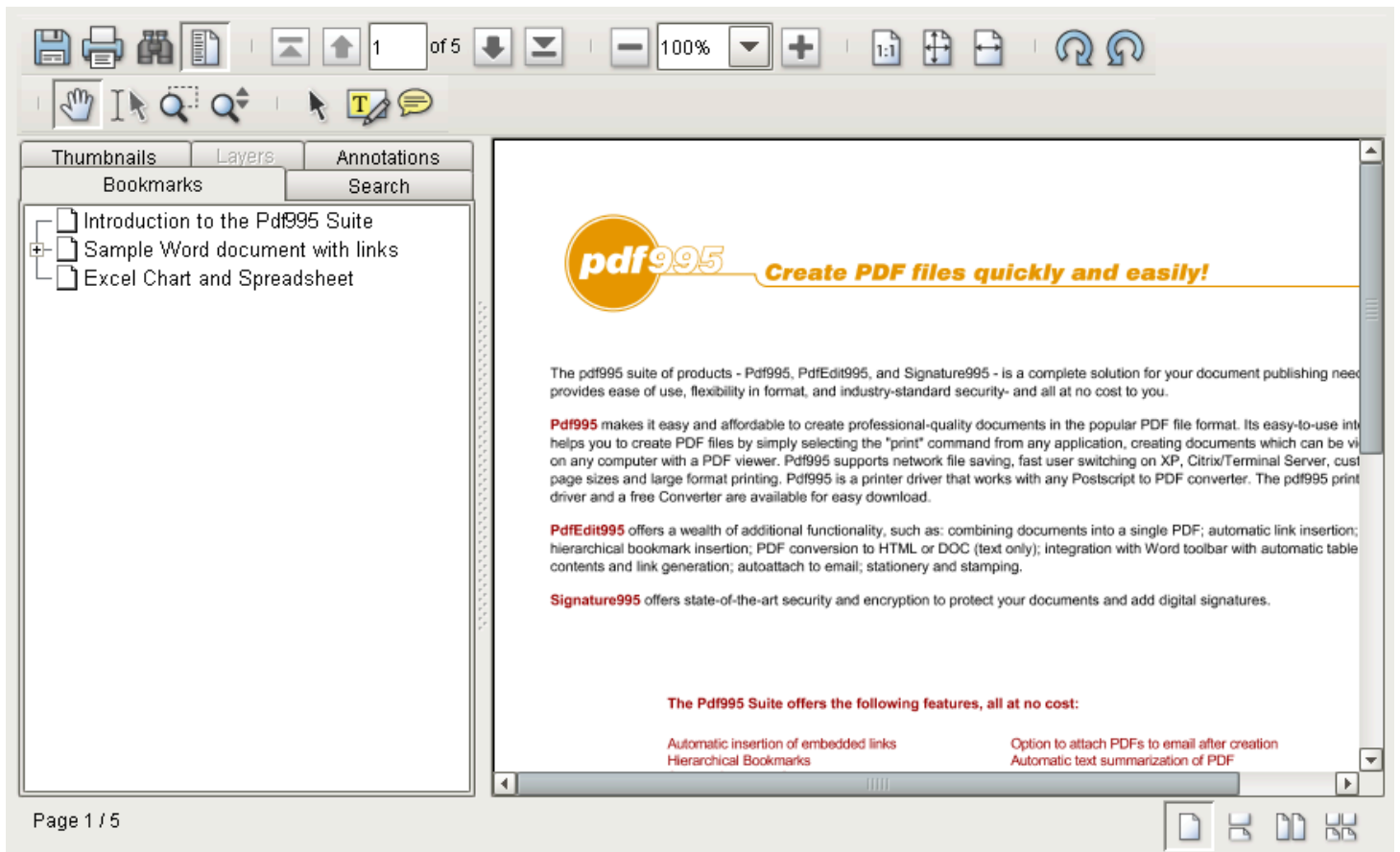


INDUCTIVE
UNIVERSITY

PDF Viewer

[Watch the video](#)

The **PDF Viewer** allows you to take PDF files and embed them into your Ignition Client. This can allow you to view PDF documents without having to leave the Ignition Client, perfect when running a client at full screen, or to view any reports that were saved in PDF format.



File Explorer and PDF Viewer

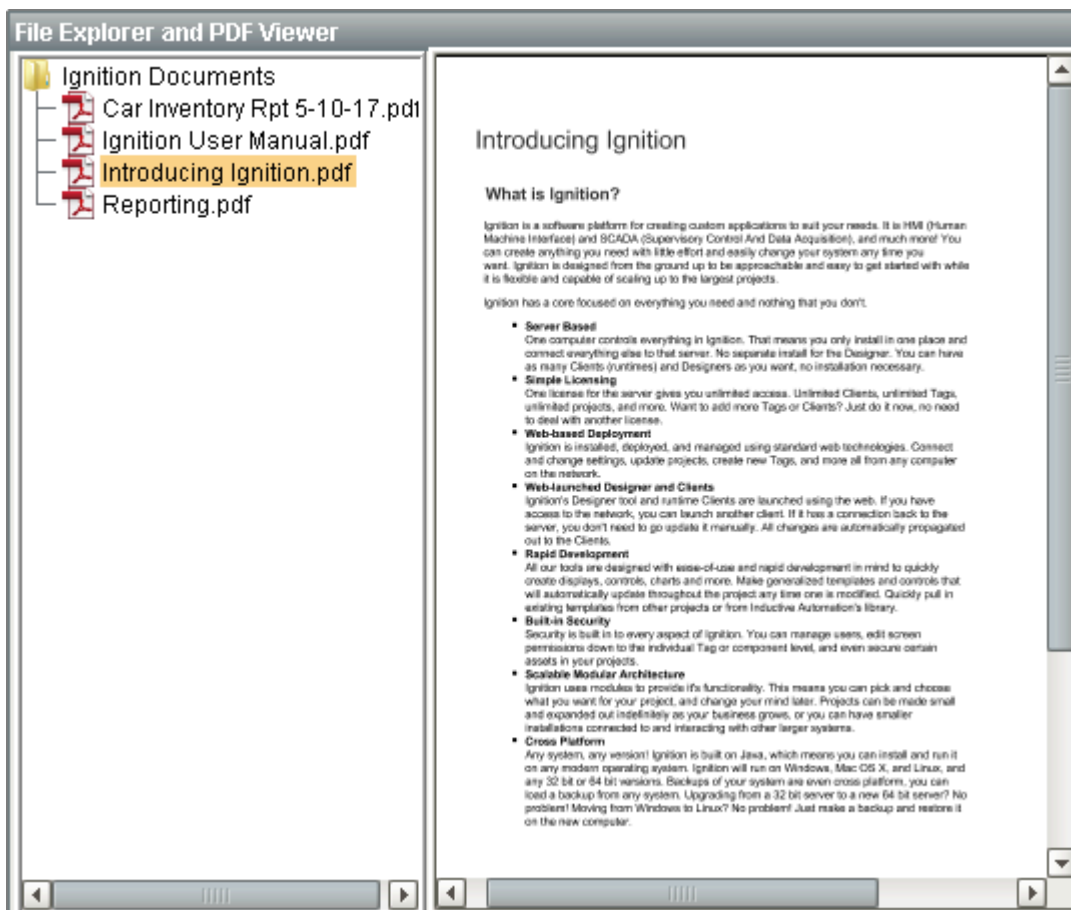


INDUCTIVE
UNIVERSITY

File Explorer and PDF Viewer

[Watch the video](#)

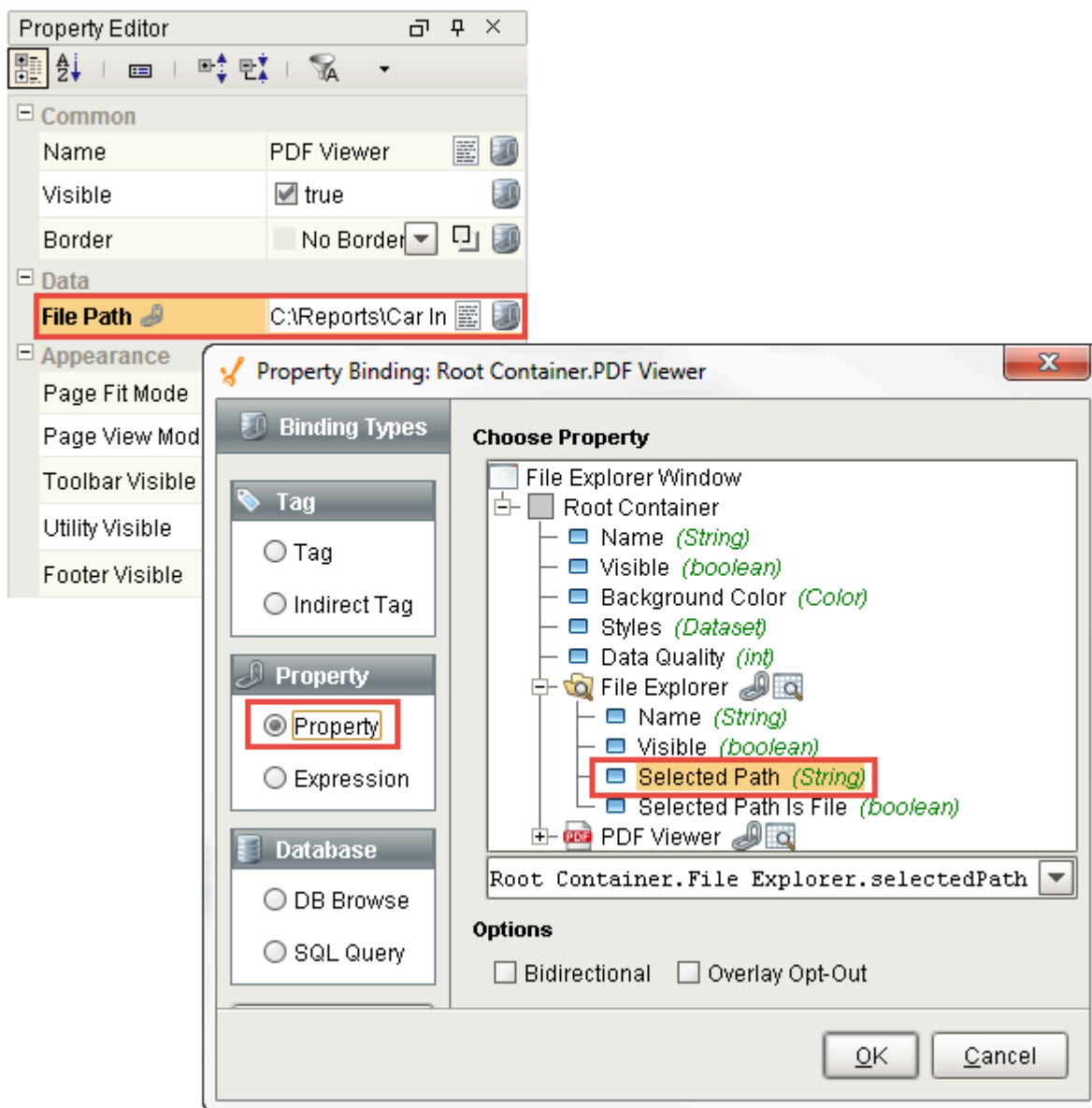
The **File Explorer** component displays a file system tree structure that allows users to navigate around various folders. It can be rooted to any folder including shared network folders, and can filter file types by their file extension like 'pdf.' The File Explorer is typically used in conjunction with the PDF Viewer component in order to create a PDF viewing window. This is very useful for viewing documents from within a project.



File Explorer and PDF Viewer Example

Let's setup the File Explorer and the PDF Viewer to create a window to view a PDF documents.

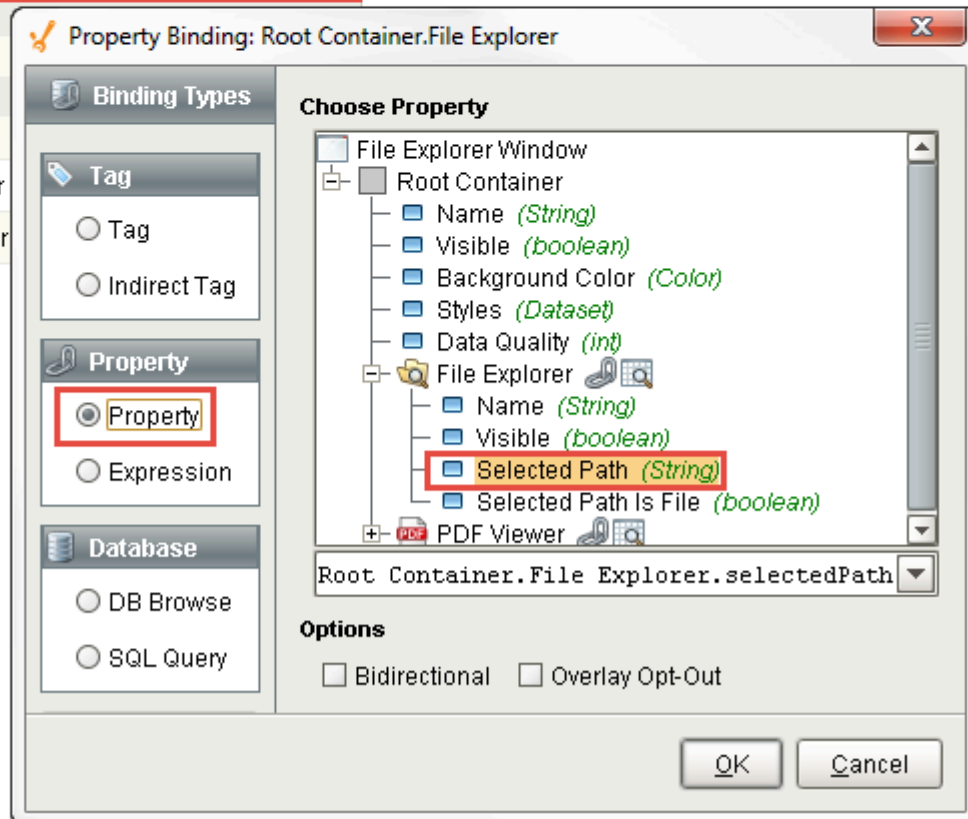
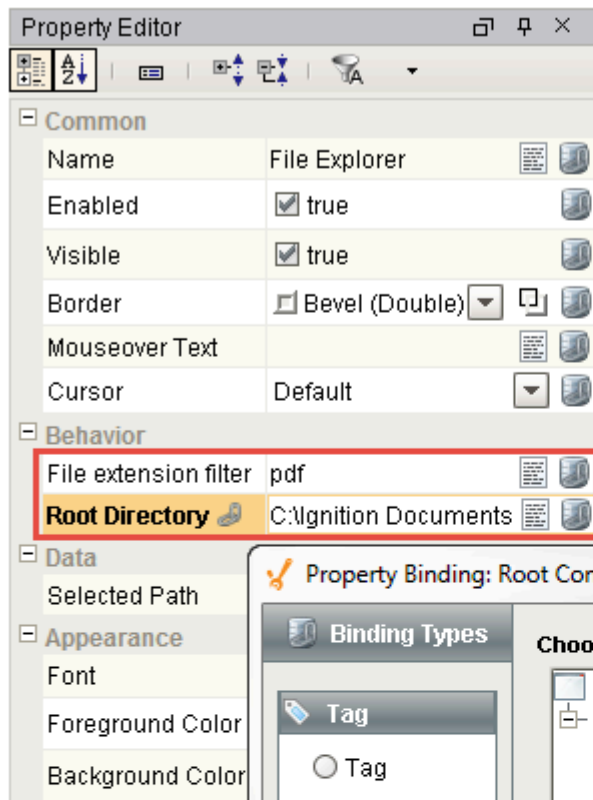
- In Designer, drag a **File Explorer** component and **PDF Viewer** component on to a window and place them side by side.
- Select the **PDF Viewer** component and Bind its File Path property to **File Explorer's** Selected Path property. Select the **Property** binding type, and drill down to the **File Explorer Selected Path** property.



- In the **Property Editor** of the **File Explorer** component, set the Root Directory property to any folder. You can either type in the folder path or bind the root directory as shown in the image below. The Root Directory restricts which directories are accessible on the component. This is typically used to restrict access to a particular folder so that the user doesn't have access to the entire file system. All folders nested within the Root Directory can be accessed on the component, so keep this in mind if you do not want user to see specific folders within the rooted directory.

NOTE

If you set the File Explorer's Root Directory to a network folder, all clients will be able to access documents within all folders in that network folder).



- To filter for only PDF file types, enter 'pdf' (without quotes) in the **File Extension Filter** property, otherwise, all file types will be displayed inside the Root Directory.
- In **Preview Mode**, click on one of the reports in your Root Directory. The **PDF Viewer** works by passing the file path of the file you selected to the PDF Viewer. You'll also notice several PDF Viewer properties that drive the appearance of the PDF Viewer: Page Fit Mode, Hide or Show Toolbar, Utility Bar, Highlight, Select, Save, Print, and more. Note: Clicking on the Save icon in the runtime saves a copy of the report to the **client** computer, not the Ignition Gateway.

File Edit View Project Component Alignment Shape Tools Help

Project Browser

- File Explorer Window
 - Root Container
 - File Explorer
 - PDF Viewer

Tag Browser

Tag	Value	Data Type
Tags		
System		
Client		
All Providers		

Property Editor

Common

Name: File Explorer

Enabled: ☒ true

Visible: ☒ true

Border: ☐ Other Border

Mouseover Text:

Cursor: Default

Behavior

File extension filter: pdf

Root Directory: C:\Reports

Data

Selected Path: C:\Reports\Car Invent

Appearance

Font: Dialog, Plain, 12

Foreground Color: 0,0,0

Background Color: 255,255,255

Reports

- Car Inventory Report 5-11-17 0
- Car Inventory Report 5-12-17 0
- Car Inventory Report 5-16-17 0
- Car Inventory Report 5-17-17 0
- Car Inventory Rpt 5-3-17.pdf
- Car Inventory Rpt 5-4-17.pdf
- Car Inventory Rpt 5-5-17.pdf
- Veggie Rpt 4-24-17.pdf

Used Car Inventory Report

PDF Viewer

Legend: Ford (orange), Toyota (dark blue), Mercedes (medium blue), Buick (light blue), Lexus (very light blue), Jeep (yellow-orange)

Brand	Count
Ford	13
Toyota	22
Mercedes	2
Buick	6
Lexus	7
Jeep	11

Page 1 / 1

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Report Data



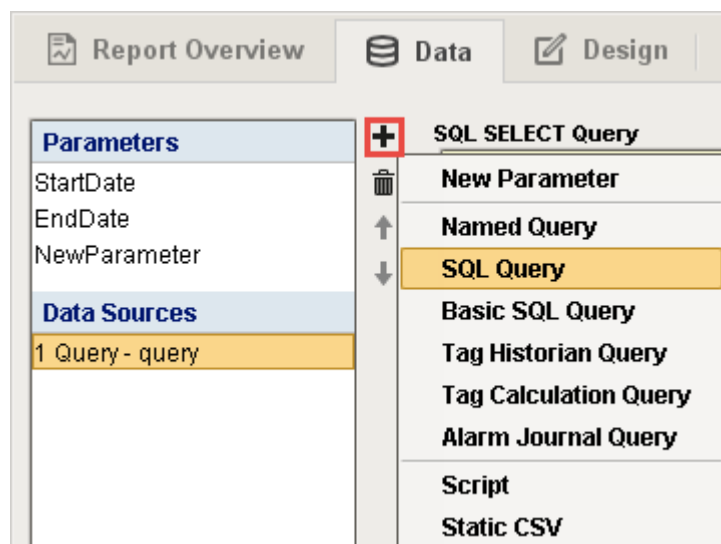
INDUCTIVE
UNIVERSITY

Report Data

[Watch the video](#)

Data

The most critical part of any report is data. Without data, there is nothing to report! In the **Reporting Module**, all data is collected as either a Parameter or a Data Source, and it is all configured in the Data tab of the report. Clicking on the plus (+) button allows you to add new sources of data, bringing up a menu so you can choose the type of data to add. If a Parameter or Data Source is highlighted, clicking the trashcan (🗑️) button will delete that particular source of data.

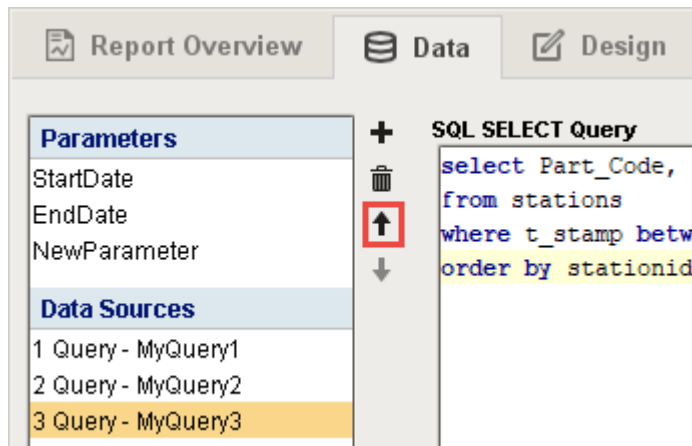


Order Matters

The order of the Parameters and Data Sources is important. In some cases, a parameter or data source may have the capability to reference the results of another type, such as a Parameter referencing the value of another Parameter. Parameters and Data Sources may **ONLY** reference other types of data that are listed vertically above them in the list. In short, the bottom-most Data

Source may reference all other parameters and data sources, while the top-most Parameter may not reference any other parameter or data source.

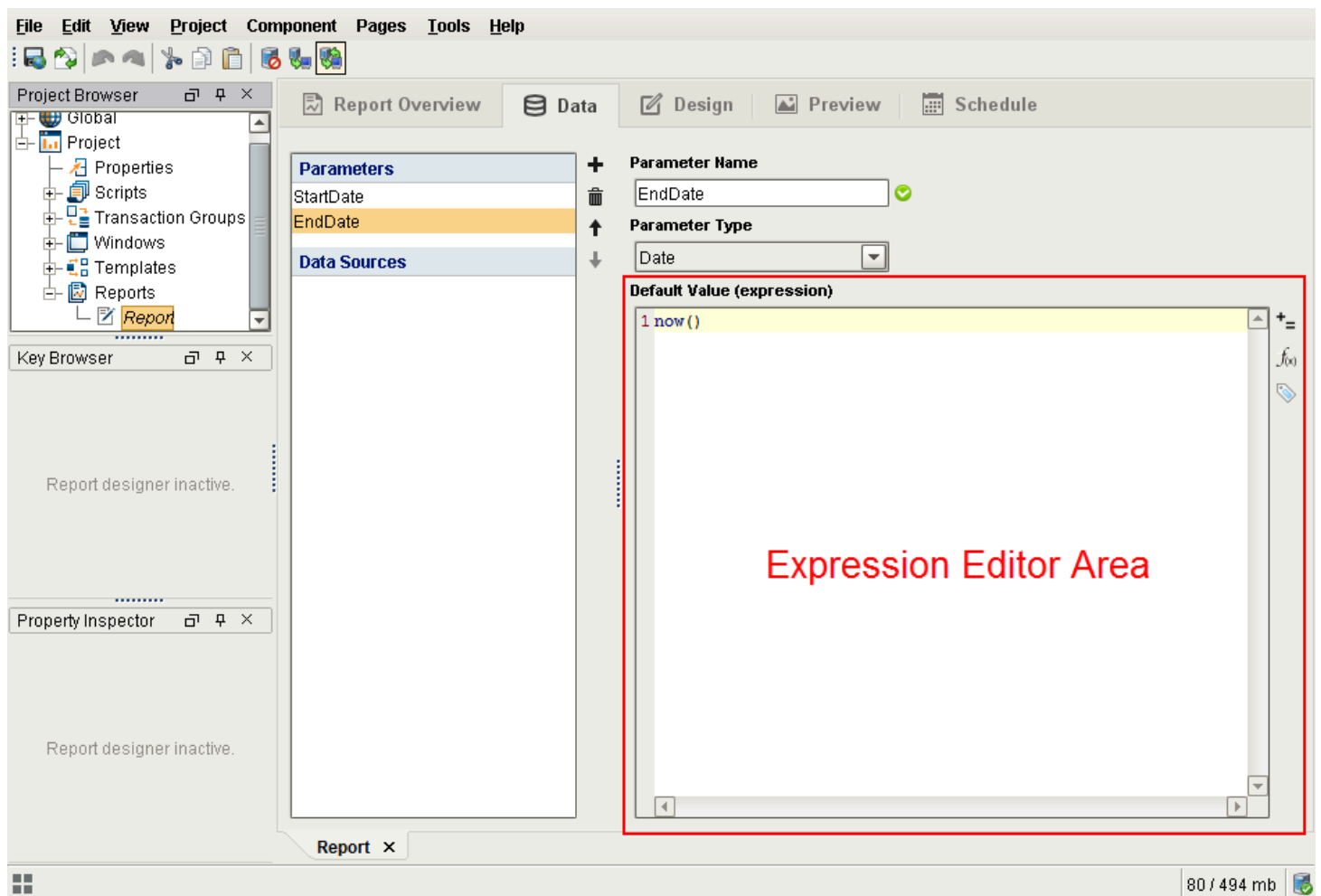
Because of this, it is possible to reorder them using the up (↑) and down (↓) arrow buttons.



Parameters

Parameters are the way you get dynamic data into your charts. These properties are exposed on the **Report Viewer** component and can be bound to other components to allow your users to select what report data they want to see. You can use these to set up dates, time ranges, area selectors, titles, and anything else you want in your report.

By default, StartDate and EndDate properties are automatically created for you because they are almost always used to filter report data. You can, of course, delete them if you don't need them.



Data Sources

Data Sources are the primary means of getting data out of where it lives and into a report. There are eight Data Sources included in the Reporting Module: six Query types, a Script data source and Static CSV. Each data source offers a unique method of collecting (or amending in the case of Script Source) data. Aside from the Script datasource, all will require you to specify a **Data Key**. The Data Key is a unique identifier that represents the top-level of the data collected through this source – think of it as the label for the data source, or as the parent node of your source's collected 'data tree'. When your report is generated, the data collected under this identifier is passed to the reporting engine, which utilizes this identifier to appropriately place your data in the final report.

Here is a complete list of all the Data Source types:

- **Named Query**: A pre-configured query that runs as a prepared statement. If your Gateway is already using Named Queries to display data, then you can easily add those queries to your report.
- **SQL Query Data Source**: a straightforward query that allows parameters to be inserted with question marks (?) and has a graphical Query Builder.
- **Basic SQL Query**: a simplified version of the SQL Query that supports references directly in the query using the brace characters .

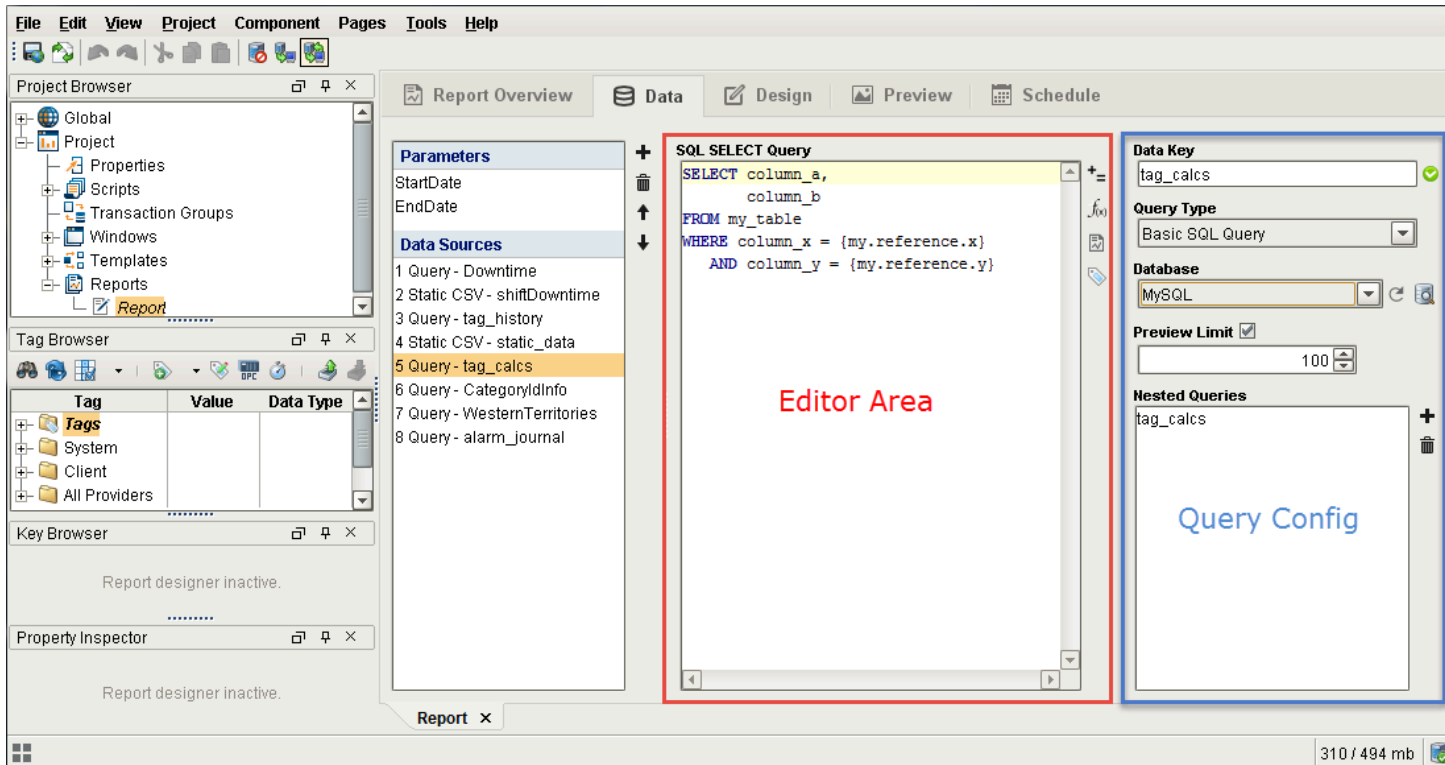
- **Tag Historian Query**: the same Tag History query builder you are familiar with from property bindings.
- **Tag Calculation Query**: similar to the Tag Historian query, but this one allows calculations to be performed on the resulting data (min, average, duration on, count off, etc.).
- **Alarm Journal Query**: the same alarm journal query builder you are familiar with from the Functions property binding.
- **Script**: a blank script that you can use to create a dataset in any way you like.
- **Static CSV**: copy and paste static CSV data directly as an easy way to test your reports.

DEVELOPER TIP

Give your Data Keys readable names that say something about the table or data they represent. This will make it easier to build meaningful reports!

The Query data source types share some common features in the UI, and they all return what is essentially a Dataset. If you take a look at the image below, you'll see we outlined two main areas of the Basic SQL Query data source type. By default, the largest part of any Query panel is taken up by the central Editor Area. This space is tailored for each query type. On the right side of the Data panel, we have a "Query Configuration" area. In the screenshot, you'll see there are options for the *Data Key*, *Query Type*, *Database*, *Preview Limit* and *Nested Queries*. Of these, *Data Key*, *Query Type* and *Nested Queries* are shared among all Queries, while *Preview Limits* are conveniently available in SQL query types. Here is some brief information on each type:

- **Data Key**- The identifier we will reference when we design our reports. The Data Key needs to be unique, can not contain special characters (spaces, underscores, dashes are allowed), and must start with a letter. We added a convenient validator to help detect name collisions.
- **Query Type** - gives you a convenient place to change the type of query.
- **Nested Queries** - All of the query-based Data Source types can have **nested queries** in them. That is, for each row of returned data, a second query can be run based on the results in that row. You can have multiple sub queries for each row, and nest queries as deep as you want. This is particularly useful when you have data organized into runs and want to see historical data for each run.
- **Preview Limit** - Allow you to choose to limit the number of rows returned from a query when a report is being generated in the Preview Panel, and when data is sampled for the purposes of generating Data Keys in the Designer. If supported by your database, this makes it much easier to craft and preview deep query structures without the overhead and wait-times of long-running queries.



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Report Parameters

Parameters

As we covered in the overview, the Data Panel is pretty simple at first glance. When creating a new report, there is an empty list of Data Sources as well as pre-made Parameters. **Parameters** are Ignition **Expressions**. These parameters resolve at report execution time and provide a convenient way to dynamically specify content for your report. These expressions are expected to resolve to specific types.

The Parameter Types available in the ComboBox will be familiar to anyone who has used Expressions, and include:

1. **Date**
2. **String**
3. **Long**
4. **Double**
5. **Boolean**
6. **Dataset**

Adding Parameters function similar to custom properties on Vision windows and components. Their value can be shown directly on your report, or be referenced by data sources and other parameters. Parameters are given a default value when first created, but this value can be overridden once the report runs.



INDUCTIVE
UNIVERSITY

Parameters

[Watch the video](#)

What are Parameters Useful For?

Parameters share some similarities to Custom Properties in Vision components in that they allow an Ignition user to specify dynamic data to be used in reports at runtime. For example, a manufacturer

may want to generate reports for the active production lines every Monday morning, but you don't know ahead of time which production lines will be running. We can create a parameter called `activeProductionLines` and use an expression or Tag value to determine which lines were active, and use the parameter in a SQL Query datasource to only include data from active lines.

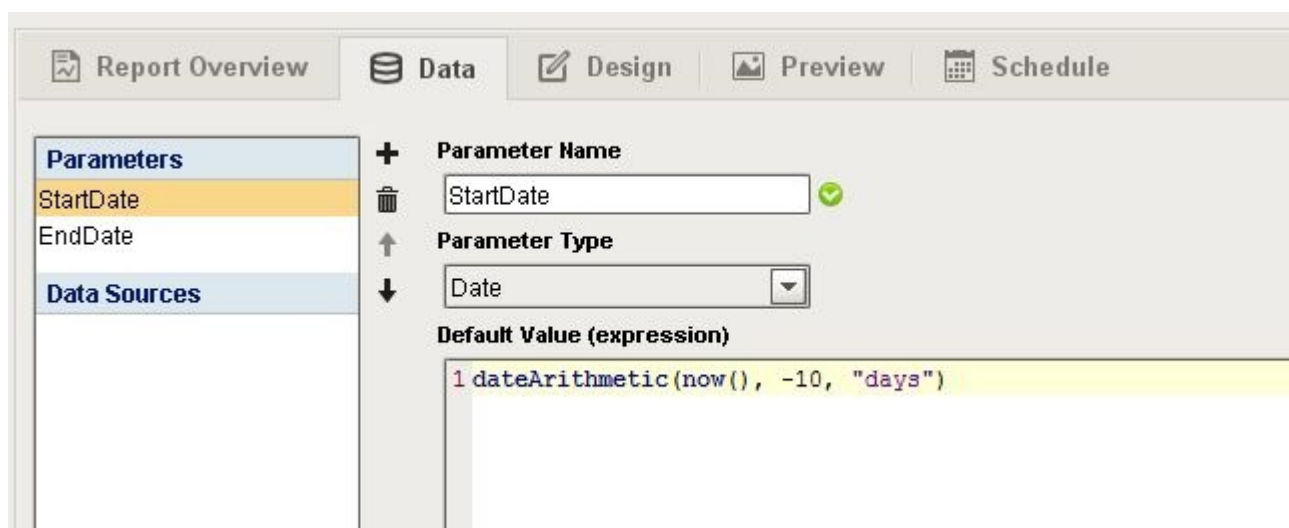
Parameters can be declared with an empty (null) value. In this event, it can be fed values from the Vision Report Viewer component, or in the Report Scheduling panel. The Report Viewer can use parameters to bind to data through Ignition Bindings. If a parameter is supplied a default value in the Report Data panel (we call these default parameters), that value will be automatically be supplied to the Report Viewer component. If a parameter has no default, the Report Viewer and Preview Panel will notify the user that a parameter is undefined. It is important to note that both Scheduled Report and Vision component parameters take precedence over default parameters, so any default parameter can easily be overridden while also providing a baseline value.

NOTE

Default values for parameters are evaluated in the gateway, not by clients. As such, the expressions have a gateway scope.

Default Starting Parameters

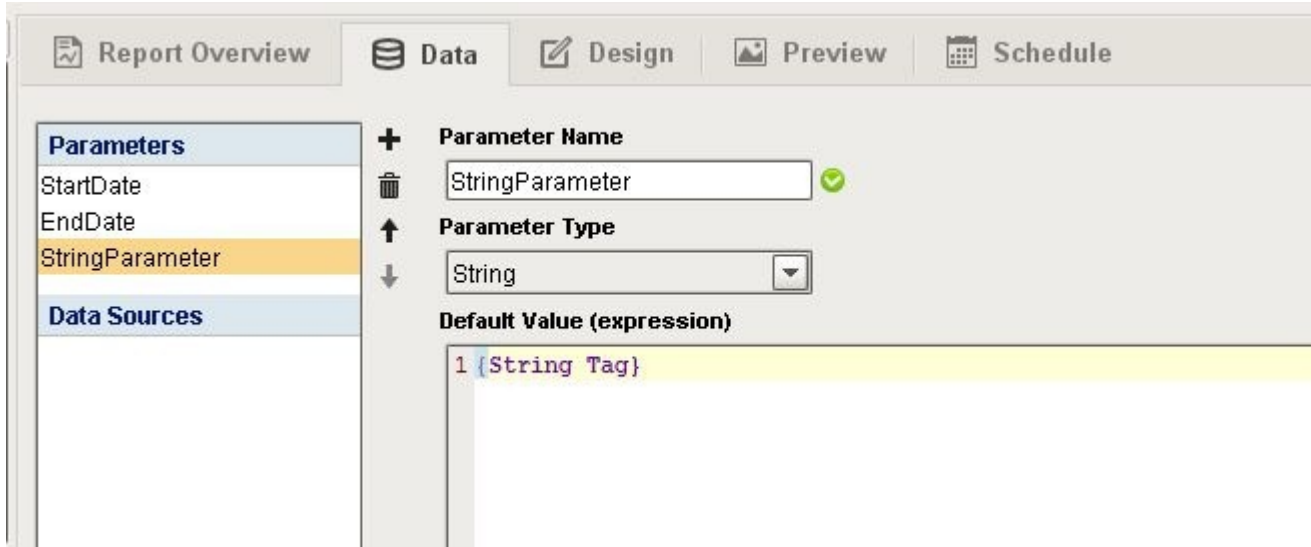
When you first create a report, you start with two parameters: **StartDate** and **EndDate**. EndDate will have a default value of `'now()'`, while StartDate will have a default value of `'dateArithmetic(now(), -8, "hr")'`. Notice how I changed the default value of the StartDate to be a difference of 10 days instead of 8 hours by modifying the Default Value expression. You can also delete both StartDate and EndDate parameters if you do not need date parameters in your report.



Creating New Parameters

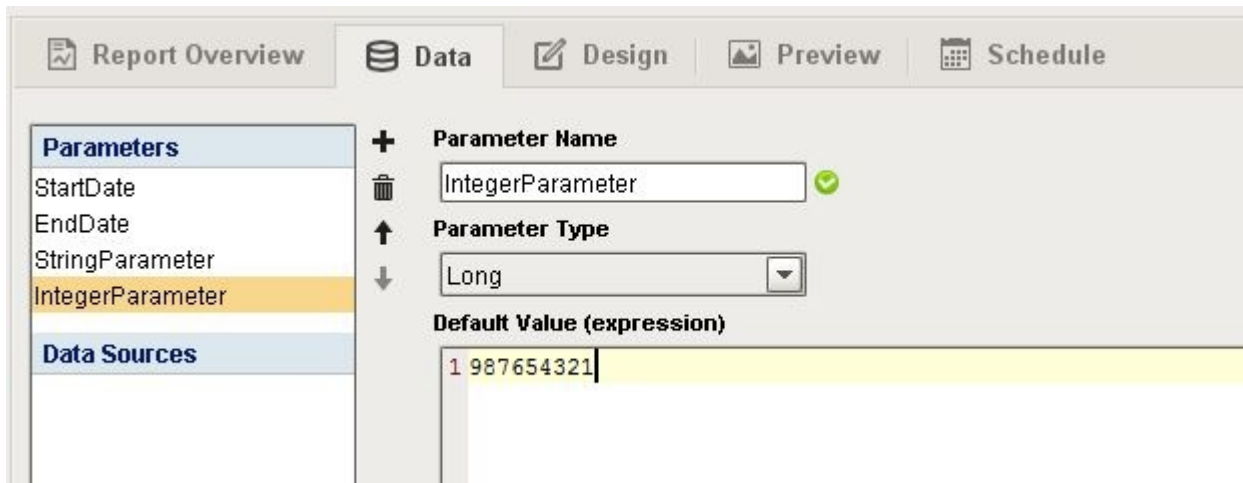
New parameters may be added by pressing the **+** button.

Once created, a default value should be given to the parameter. In addition to expressions, you can also have your default value reference a Tag using the  button on the right of the expression area.



The screenshot shows the 'Data' tab of a report designer. On the left, a sidebar lists 'Parameters' (StartDate, EndDate, StringParameter) and 'Data Sources'. The 'StringParameter' is selected. The main area shows the configuration for this parameter: 'Parameter Name' is 'StringParameter', 'Parameter Type' is 'String', and the 'Default Value (expression)' is '1 {String Tag}'. A green checkmark is next to the parameter name.

You can also use a literal value. Normal Expression languages syntax applies, so **dates** and **strings** must be wrapped in quotation marks, but **numerical** values can be written without quotes.



The screenshot shows the 'Data' tab of a report designer. On the left, a sidebar lists 'Parameters' (StartDate, EndDate, StringParameter, IntegerParameter) and 'Data Sources'. The 'IntegerParameter' is selected. The main area shows the configuration for this parameter: 'Parameter Name' is 'IntegerParameter', 'Parameter Type' is 'Long', and the 'Default Value (expression)' is '1 987654321'. A green checkmark is next to the parameter name.

Finally, you can reference other parameters. In the example below, **BoolParameter** is referencing the value of **IntegerParameter**.

NOTE

When referencing other parameters, you must type the name of the parameter exactly within a set of curly braces , including capitalization. Also, you can only reference parameters that are above the current parameter. Thus, while **BoolParameter** can reference the **IntegerParameter**, the **IntegerParameter** may not reference **BoolParameter**.

Report Overview

Data

Design

Preview

Schedule

Parameters

StartDate

EndDate

StringParameter

IntegerParameter

BoolParameter

Data Sources

Parameter Name

BoolParameter

Parameter Type

Boolean

Default Value (expression)

1 if ({IntegerParameter} = 1000, 1, 0)

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Named Query Data Source

New in 7.9.4



INDUCTIVE
UNIVERSITY

Named Queries in Reports

[Watch the video](#)

Using the Named Query Data Source

Using the Named Query datasource is simple to configure provided you already have a **Named Query** created, and works in the same way that other Named Query bindings work. The Named Query report datasource will execute the selected Named Query on the Gateway, and can use the **Report Parameters** as it's parameters.

This data source supports **Nested Queries**.

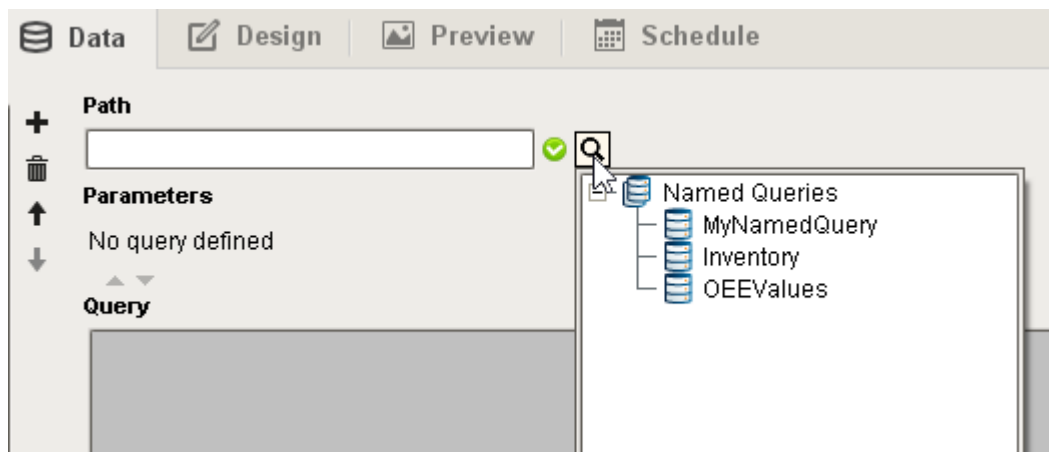
The screenshot shows the configuration window for a Named Query Data Source. On the left, a sidebar contains 'Parameters' (with 'BayNumber' listed) and 'Data Sources' (with '1 Query - named_query' selected). The main area is divided into sections: 'Path' with a text field 'My First Query' and a search icon; 'Parameters' with a table; 'Query' with a text area containing a SQL query; 'Data Key' with a text field 'named_query'; 'Query Type' with a dropdown menu set to 'Named Query'; 'Preview Limit' with a checked checkbox and a value of 100; and 'Nested Queries' with a list containing 'named_query'.

Type	Name	Data Type	Value
Value	BayNum	Int1	{BayNumber}

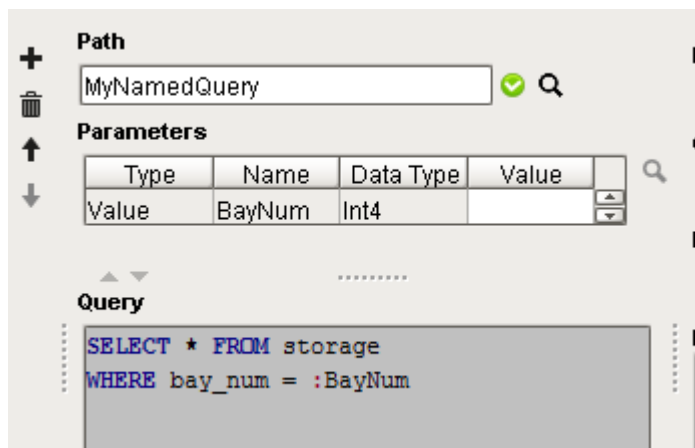
```
SELECT * FROM storage WHERE bay_num = :BayNum
```

Configuring a Named Query Data Source

First select a Named Query by clicking the 🔍 button next to the Path field.



In the resulting list, select one of the configured named queries. The Named Query parameters will appear, along with a preview of the query.



One-by-one, **select** each parameter, and then click the  button next to the **Parameters** table. This provides a popup of all report parameters that are available on the report. Select a Report parameter that will provide a value to the Named Query parameter. Once every parameter has a value, check the results in the Preview Panel.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

SQL Query Data Source



INDUCTIVE
UNIVERSITY

SQL Query

[Watch the video](#)

Using the SQL Query Datasource

The **SQL Query** Data Source allows you to craft parameterized queries that run as a prepared statement. As Prepared Statements, these queries are more resistant to SQL injection offering additional security over basic queries.

The **SQL Query** type looks very similar to the **Basic SQL Query** type. It has a large text area where you can enter in a SQL select query. On the right, you have the option to rename the query, choose what database to run this query against, and add in a [Nested Query](#).

The screenshot shows the Ignition configuration interface for an SQL Query Data Source. At the top, there are five tabs: 'Report Overview', 'Data' (which is selected), 'Design', 'Preview', and 'Schedule'. On the left side, there are two expandable sections: 'Parameters' containing 'StartDate' and 'EndDate', and 'Data Sources' containing '1 Query - query'. The main central area is titled 'SQL SELECT Query' and contains a large, empty text box for writing the SQL query. On the right side, there are several configuration options: 'Data Key' with a text input field containing 'query' and a green checkmark; 'Query Type' with a dropdown menu set to 'SQL Query'; 'Database' with a dropdown menu set to '<Default>' and a refresh icon; 'SQL Query Builder' with a 'Show Builder' button and a dropdown menu set to 'Universal'; 'Preview Limit' with a checked checkbox and a numeric input field set to '100'; and 'Nested Queries' with a list containing 'query' and a plus icon to add more.

Parameters in SQL Query

Because the SQL Query Data Source runs as a prepared statement, passing parameters into this query type works a little differently.

Instead of placing a parameter within characters like the Basic SQL Query, we place a **?** where we would like to pass in a parameter. Doing this will actually create a new text area below the query area for your parameter. This smaller text area is where you can pass a parameter into the query. You can pass in Tag values or Report Parameters, and since the parameter area allows expressions, you can use expressions to create any value you like from a combination of Tags and Report Parameters.

You can add as many of these to a query as needed, with new parameter areas popping up underneath as they get added. To help keep track of what parameter corresponds to which **"?"**, when entering in a value into the parameter area, its associated **"?"** will highlight to show you which parameter you are currently working on.

Parameters

StartDate
EndDate
Station

Data Sources

1 Query - query

+

SQL SELECT Query

SELECT * FROM equipment
WHERE station = 2

Parameter 1

1 {Station}

Parameter 1

1 {Acme/Site1/Bay 4/Temperature} + {TempOffset}

Parameter 1

1 "STATION " + {StationID}

The screenshot displays the SQL Query Builder interface. At the top, a text area contains the following SQL query:

```
SELECT * FROM equipment
WHERE station = ? and
stationname = ? and
stationid = ? and
stationtype = ?
```

Below the query, there are four parameter input fields labeled "Parameter 1", "Parameter 2", "Parameter 3", and "Parameter 4". Each field contains the value "1". To the right of each input field is a set of controls: a "+" sign, an equals sign, a function icon (f(x)), a table icon, and a tag icon.

Crafting Queries with the Query Builder

The SQL Query data source includes the powerful SQL Query Builder tool. The Query Builder is a powerful Drag-and-Drop query building GUI that allows you to make complex queries from your connected databases. While a basic understanding of SQL helps make the most of the Query Builder tool, most people will have no problem creating effective queries after a brief tutorial. The Query Builder is a third party tool that we brought into the Reporting Module. We go into detail on how to use it on the [Query Builder](#) page, but you can also check out the [Active Query Builder's documentation](#) for additional information.

Using the builder

To activate the Query Builder in your **SQL Query** datasource type, start by selecting the SQL Syntax version from the drop down menu beneath the SQL Query Builder button. If your Database type isn't available (or you aren't sure), you can get most of the general functionality by selecting the **Universal** option. Then push the SQL Query Builder button to show the [Query Builder](#).

The screenshot shows a button labeled "SQL Query Builder" with a wrench icon and the text "Show Builder".

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Basic SQL Query



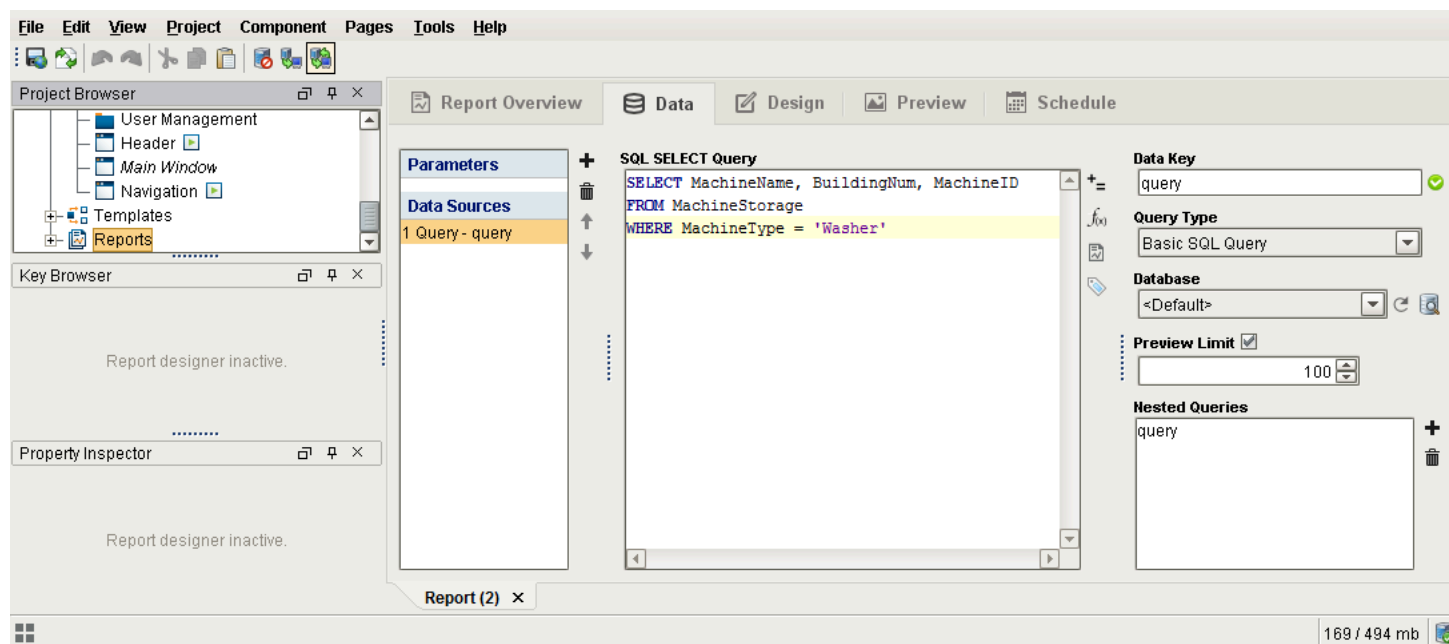
INDUCTIVE
UNIVERSITY

Basic SQL Query

[Watch the video](#)

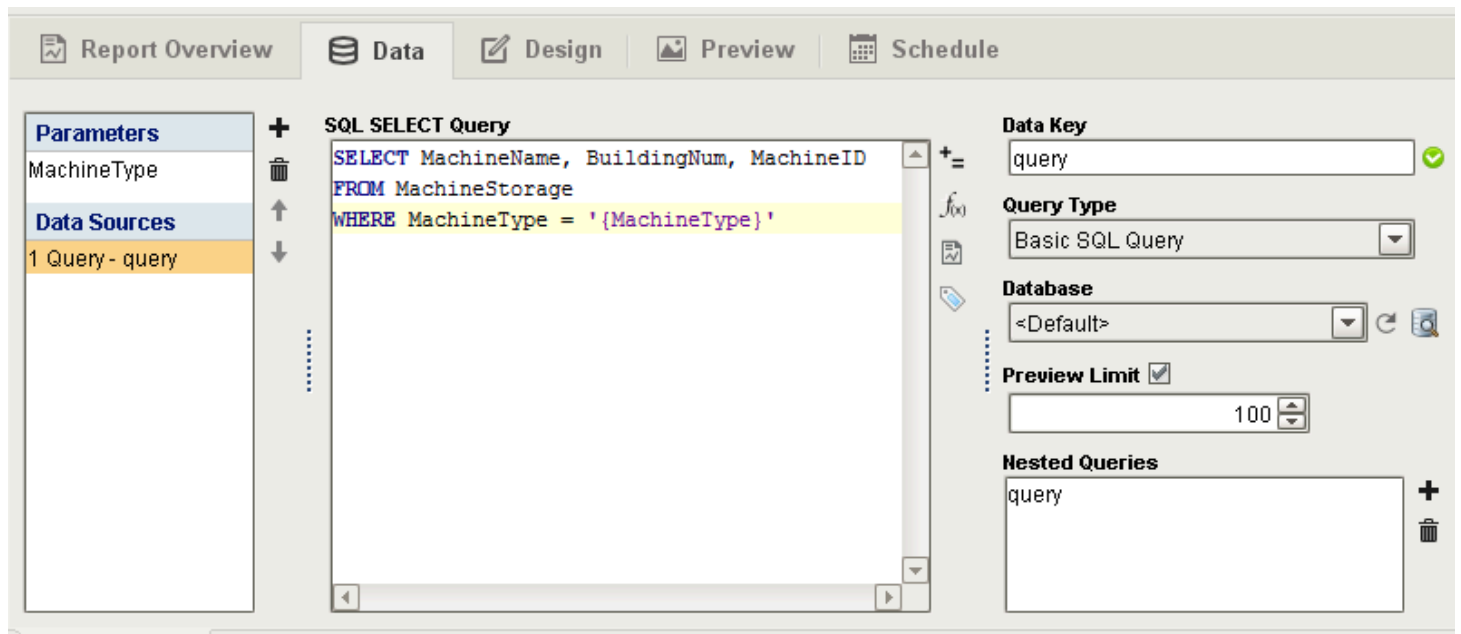
This query type is the common type typically seen through much of Ignition before version 7.8. You can write queries which include Tag path references, expressions, or report parameters which resolve at run time.

You can enter in a static SQL query like in the image below. The query will return rows that have a MachineType of 'Washer'.



Report Parameters in a Basic SQL Query

Queries can also be made dynamic using things like report [parameters](#). To insert a report parameter, simply click the Parameter Reference button () to the right of the query area and select your parameter. This allows for a more dynamic query, since new values can be passed into the parameter at runtime, giving the ability to change the type of machine this query is looking for.



Parameters are inserted directly into a report. This means that the datatype of parameter will affect how it should be referenced in the query. For example, since the parameter `MachineType` is a string, it will need a single or double quotes around it.

String Parameters

```
WHERE MachineType = '{MachineType}'
```

Since an integer does not need quotes around it, if your parameter is a Long, Double, or a Boolean, you can directly place the parameter in your query, without the quotes.

SQL - Long, Double, and Bool Parameters

```
WHERE MachineNum = {MachineNumber}
```

Working with Dates

If your parameter is a date object, then special consideration must be made.

The query will not accept a date object directly, it must first be converted to a string by putting quotes around it. However, database generally prefer datetime objects in very specific formats, such as **yyyy-MM-dd HH:mm:ss**. This means we need to reformat the date on any report parameters we want to pass to the Basic SQL Query. There are two main approaches to this:

Reformat the Date Parameters

By utilizing the expression language's `dateFormat()` function, we can simply specify the format of the date.

SQL - Reformatted Date Parameter

```
dateFormat(  
    dateArithmetic(now(), -8, "hr"), //don't forget the comma at the end of this  
    line...  
    "yyyy-MM-dd HH:mm:ss")          //...as well as the outer closing ')'
```

The screenshot shows the 'Data' tab of a report design tool. On the left, a sidebar contains 'Parameters' (with 'StartDate', 'EndDate', and 'test' listed) and 'Data Sources' (with '1 Query - query' listed). The main area displays the configuration for the 'StartDate' parameter: 'Parameter Name' is 'StartDate', 'Parameter Type' is 'Date', and 'Default Value (expression)' is the SQL code from the previous block. The SQL code is highlighted in yellow.

Create New Formatted Parameters

In some cases, you may wish to leave the original "raw" Date parameter alone, and create a display-friendly version as a string.

To do this, simply make a parameter with type string and use the `dateFormat()` expression on a date. In the image below, you can see that the **StartDate** parameter is used in a new **StartString** parameter. Additionally, an **EndString** parameter has been created that is using the **EndDate** parameter. This way, we can bind a calendar component directly to the Start and EndDate parameters and all the formatting will be done automatically in the report.

Report Overview

Data

Design

Preview

Schedule

Parameters

StartDate

EndDate

StartString

EndString

Data Sources

1 Query - query

+

Parameter Name

StartString

Parameter Type

String

Default Value (expression)

1 dateFormat({StartDate}, 'yyyy-MM-dd HH:mm:ss')

+

+

f(x)

This format was used with a MySQL database, so your database may take a different format. Refer to your database's documentation for suggested date formats.

Once the new string parameters have been created, we can then reference them in the Basic SQL Query just like a normal string.

Report Overview

Data

Design

Preview

Schedule

Parameters

StartDate

EndDate

StartString

EndString

Data Sources

1 Query - query

+

SQL SELECT Query

SELECT *

FROM group_table

WHERE t_stamp BETWEEN '{StartString}'

AND '{EndString}'

+

+

f(x)

Data Key

query

Query Type

Basic SQL Query

Database

<Default>

Preview Limit

100

Nested Queries

query

+

+

f(x)

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Tag Historian Query



INDUCTIVE
UNIVERSITY

Tag Historian Query

[Watch the video](#)

The Tag Historian Query provides a simple way to query data from Tag Historians. In the Tag Historian Query you can collect data from Historical Tags for specific date ranges, apply aggregates, and specify the sample size. It functions the same as a [Tag History Binding](#) and uses the same interface, so check out that page for more information on the individual properties.

The screenshot shows the Ignition Tag Historian Query configuration window. The interface is divided into several sections:

- Parameters:** Includes `StartDate` and `EndDate`.
- Data Sources:** A list of data sources, with `3 Query - tag_history` selected.
- Available Historical Tags:** A tree view showing the hierarchy of tags. The selected path is `MySQL > controller:default > area 1 > tanks > level_pv` and `temp_pv`.
- Selected Historical Tags:** A table showing the selected tags and their data key aliases.

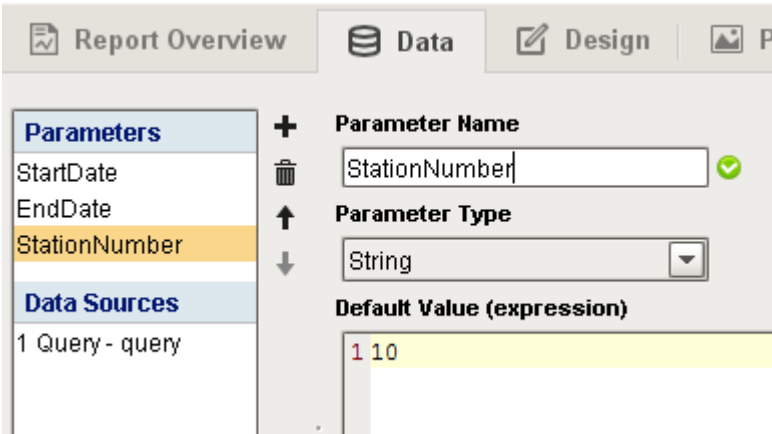
Tag Path	Data Key Alias
[mysql]level_pv	level_pv
[mysql]temp_pv	temp_pv
- Data Key:** A dropdown menu set to `tag_history`.
- Query Type:** A dropdown menu set to `Tag Historian Query`.
- Preview Limit:** A checkbox for `Preview Limit` and a numeric input set to `100`.
- Nested Queries:** A list of nested queries, with `tag_history` selected.
- Date Range:** A dropdown menu set to `Historical`.
- Start Date Binding:** A text box containing `{StartDate}`.
- End Date Binding:** A text box containing `{EndDate}`.
- Aggregation Mode:** A dropdown menu set to `Min/Max`.
- Return Format:** A dropdown menu set to `Wide`.
- Sample Size:** A dropdown menu set to `Fixed` and a numeric input set to `10`.
- Advanced:** A section with checkboxes for `Ignore Bad Quality`, `Prevent Interpolation`, and `Avoid Scanclass Validation`.

The bottom of the window shows a `Report` button and a close icon.

Using Parameters in a Tag History Query

The only difference between a Tag History Binding and the Tag History Query is when using indirection. In the Tag History Query, indirection is inserted directly into the Tag Path in the Selected Historical Tags section.

Assume we have a parameter named **StationNumber**:



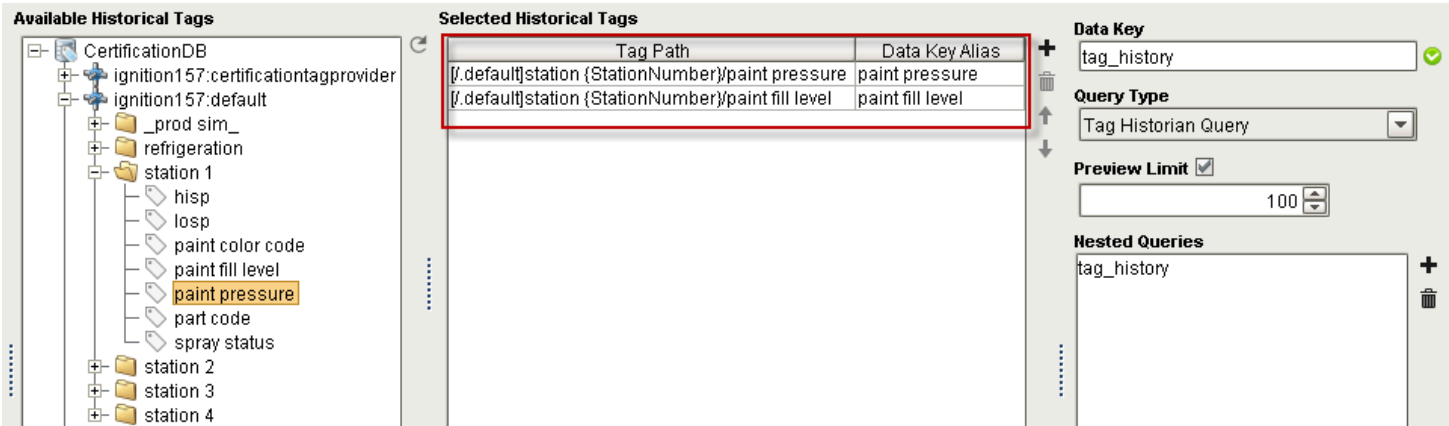
Indirection is typically done using a parameter which like the rest of reporting, is the name of the parameter enclosed in `{ }`. In the example below, I substituted the parameter '**StationNumber**' into my Tag Path for the number of the station. Note that you need to manually type out the name of the parameter, including exact spelling and capitalization.

Assuming base Tag Paths like the following:

```
[/.default]station 1/paint pressure
[/default]station 1/paint fill level
```

we can replace the "1" with a reference to **StationNumber**:

```
[/.default]station {StationNumber}/paint pressure
[/default]station {StationNumber}/paint fill level
```



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Tag Calculation Query



INDUCTIVE
UNIVERSITY

Tag Calculation Query

[Watch the video](#)

Tag Calculations are executed by the Tag Historian, providing multiple calculated values for each Tag path. While the Tag Historian Query returns a history of values and only uses the **aggregates** to perform calculations on small chunks of data in the range of time, the Tag Calculation Query will aggregate, or run calculations, on the entire range to produce a single, calculated value per Tag Path.

Assuming the following configuration:

The screenshot shows the 'Tag Calculation Query' configuration window in Ignition. The interface is divided into several sections:

- Available Historical Tags:** A tree view showing a hierarchy of tags. Under 'gen sim', there is a 'sine' folder containing 'sine0', 'sine1' (selected), 'sine2', 'sine3', and 'sine4'. A checkbox 'Use fully-qualified paths' is present.
- Selected Historical Tags:** A table with two columns: 'Tag Path' and 'Data Key Alias'. It contains two rows: '[~]gen sim/sine/...' with 'sine0' and '[~]gen sim/sine/...' with 'sine1'.
- Calculations:** A list of calculation types with checkboxes: 'Time-weighted Average' (checked), 'Min/Max', 'Closest Value', 'Basic Average', 'Sum' (checked), 'Minimum', and 'Maximum'.
- Date Range:** A dropdown menu set to 'Historical'.
- Start Date Binding:** A text field containing '{StartDate}' with a green checkmark and a magnifying glass icon.
- End Date Binding:** A text field containing '{EndDate}' with a green checkmark and a magnifying glass icon.
- Data Key:** A text field containing 'tag_calcs'.
- Query Type:** A dropdown menu set to 'Tag Calculation Query'.
- Nested Queries:** A list box containing 'tag_calcs'.

We could show the query results on a table and view the following in the Preview Panel:

Average	Sum	Tag Path	Timestamp
0.11	-57.26	sine0	Aug 28, 2017
0.05	9.86	sine1	Aug 28, 2017

Tag Historian vs Tag Calculation

When deciding between a Tag Calculation Query and a Tag Historian Query, the main difference between the two is as follows:

- The Tag Calculation Query will always return a single row for each Tag Path, with multiple columns for each aggregate/calculation, except in the case of Min/Max (see below).
- Multiple calculations may be called on each Tag Path. This is not possible with a single Tag Historian Query.

The 'Min/Max' Exception

Note that since Min/Max returns two values (the minimum and maximum value over the range), this calculation will generate two rows per Tag Path. This extra row will appear in the underlying data, even if the MinMax key isn't used in a table or chart, as it is part of the underlying data.

If we added Min/Max to our example above, the preview panel would look like the following:

Average	Sum	Tag Path	Timestamp	MinMax
-0.42	20.6	sine0	Aug 28, 2017	100
0.12	11.29	sine1	Aug 28, 2017	-9.7
-0.42	20.6	sine0	Aug 28, 2017	-99.99
0.12	11.29	sine1	Aug 28, 2017	9.7

Note that the other calculations (**Average** and **Sum**) are simply duplicated for both MinMax rows.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Alarm Journal Query



INDUCTIVE
UNIVERSITY

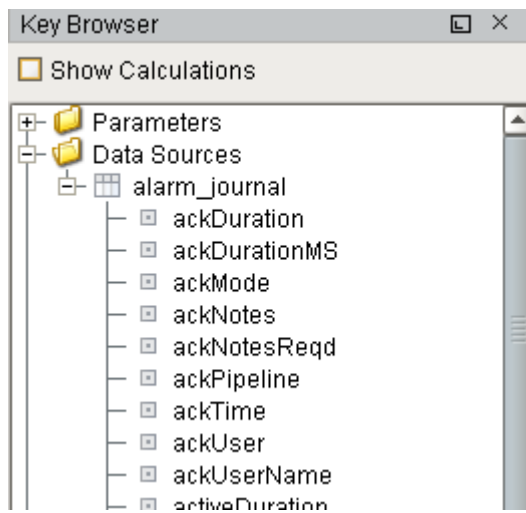
Alarm Journal Query

[Watch the video](#)

The Alarm Journal Query data source is a simple way to access Alarming data within a report. It can pull from both the alarm journal data in the database as well as the current live events that would show up in the Alarm Status Table.

The screenshot shows the 'Data' tab of the Ignition report designer. On the left, a 'Parameters' panel lists 'StartDate' and 'EndDate'. Below it, a 'Data Sources' list contains eight items, with '8 Query - alarm_journal' selected and highlighted in orange. The main configuration area is divided into several sections: 'Start Date' and 'End Date' fields with dropdown menus and search icons; 'Include Events' checkboxes for 'System', 'Active', 'Cleared', and 'Acknowledgement', along with an unchecked option 'Use current status instead of history'; 'Source Filter' and 'Provider Filter' text boxes with search icons; 'Display Path Filter' and 'Search Filter' text boxes with search icons; 'Minimum Priority' and 'Maximum Priority' dropdown menus set to 'High' and 'Critical' respectively; 'Data Key' set to 'alarm_journal'; 'Query Type' set to 'Alarm Journal Query'; 'Journal Name' set to 'journal'; and a 'Nested Queries' list containing 'alarm_journal'. At the bottom left, a 'Report' tab is visible.

Once created, the new Data Source will expose many alarming related keys in the Design Panel's Key Browser



Properties

The Alarm Journal Query has a few properties that can be configured to filter the types of events you see in the returned dataset.

Properties	Description
Start Date and End Date	The time range from which to pull Alarms from. Start Date being the furthest date in the past, while End Date is the most recent date.
Include Events	Allows you to select which type of events will be returned. The options are • System • Cleared • Active • Acknowledgment There is also a Use current status instead of history checkbox. As the text implies, this will use the current alarm status
Source Filter	Will filter the alarms based on the source path. Multiple filters can be specified by separating them with a comma, and they accept the wildcard (*) symbol. Works in the same way that the filters on the Alarm Status Table component work.
Provider Filter	Will filter the alarms based on the Tag Provider that they originate from. Can specify multiple filters by separating them with a comma.
Display Path Filter	Will filter the alarms based on the display path. Multiple filters can be specified by separating them with a comma, and they accept the wildcard (*) symbol. Works in the same way that the filters on the Alarm Status Table component work.
Search Filter	Will filter alarms by searching for the given string in both the display path and source path. Accepts comma separated paths.
Minimum Priority and Maximum Priority	The minimum and maximum priority of alarms that will be returned.
Journal Name	Located on the right side, this property allows you to specify the Journal that will be used for the query.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Scripting Data Source



INDUCTIVE
UNIVERSITY

Data Sources - Scripting

[Watch the video](#)

Script Data Source

The Script data source allows you to use **scripting** to add additional data into a report, or modify existing data. With this data source, you can pull in data from the database and then modify it using a script before pushing it out as a data key for use in the report.

The screenshot shows the Ignition Reporting tool interface. At the top is a navigation bar with tabs: Report Overview, Data (selected), Design, Preview, and Schedule. On the left, there are two panels: 'Parameters' and 'Data Sources'. The 'Parameters' panel lists 'StartDate', 'EndDate', and 'test'. The 'Data Sources' panel lists '1 Static CSV - static_data' and '2 Script' (which is selected and highlighted in orange). In the center, there is a script editor with a line number margin on the left (1-12). The script defines a function 'updateData(data, sample):' with a docstring explaining its purpose and arguments. The script then uses 'system.date' to check the year and update the 'myResults' and 'StartDate' keys in the data map.

```
def updateData(data, sample):  
    """  
    This function has the opportunity to add additional data into the report's  
    data map.  
  
    Arguments:  
        data: This is a map, whose keys are report data keys and values should  
              be sequences, maps, scalar values, or datasets to provide inform  
              the report.  
        sample: A flag that will be True if the report data being gathered is  
                for a preview of the report. Use to avoid slow queries and cal  
                to keep previews quick.  
    """  
    1  
    2    #data['myKey']='Example'  
    3  
    4    #Extract the year from the StartDate parameter and compare to 2017  
    5    if system.date.getYear(data['StartDate']) == 2017:  
    6        data['myResults'] = "The year is currently 2017!"  
    7    else:  
    8        data['myResults'] = "What year is it? Not 2017, that's for certain!"  
    9  
    10  
    11    data['StartDate'] = system.date.addDays(data['StartDate'], 2)  
    12
```

Accessing Parameters and Data Sources

One of the main uses of the Scripting Data Source is to allow for data access and manipulation as the report is being generated, allowing you to replace the original results, or add new additional content.

The syntax for creating or referencing a data key in the Script datasource is described below:

Pseudocode - Data Key Syntax

```
data['keyName']
```

ORDER MATTERS

As mentioned on the [Report Data](#) page, the order of Data Sources determines which parameters and data sources they may reference.

Because of this, it is **highly recommended** that Scripting Data Sources are placed at the bottom of the Data Sources list.

Creating a New Key

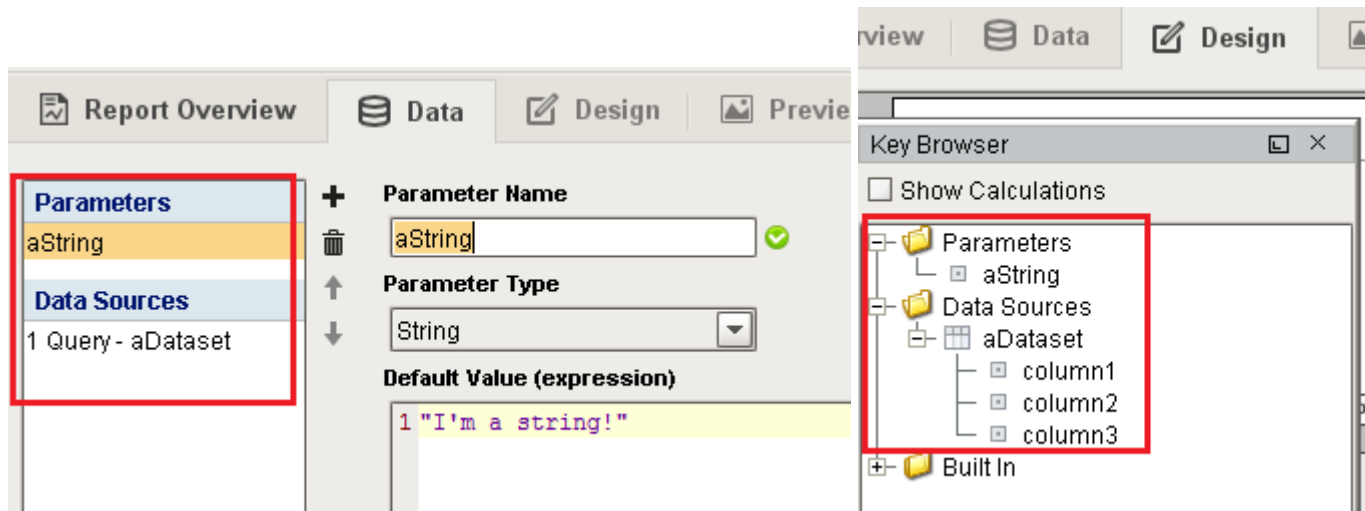
To create a new key that the report can use:

Python - Creating a New Key

```
data['newKey'] = "This key was generated on the fly!"
```

The type of data assigned to the key determines where it appears in the Design Panel.

Simple data types, like strings and integers, will appear as **Parameters**. Datasets will appear as **Data Sources**.



Parameters

Where the 'keyName' is the name of your data key. Thus, reading the value of a parameter, such as the initial StartDate parameter, can be accomplished by using `system.date.getYear()` and an **if-statement**.

Python - Accessing a Parameter's Value

```
# Extract the year from the StartDate parameter and compare to 2017.
if system.date.getYear(data['StartDate']) == 2017:
    # Do work here if the year lines up.
```

Of course, we can write back to the key and override it's value:

Python - Overriding the Default StartDate Parameter

```
# This line would override the default StartDate parameter by add adding a day.
data['StartDate'] = system.date.addDays(data['StartDate'], 1)
```

Additionally, this allows you to expand the Accessing a Parameter's Value example above by creating a new key:

Python - Accessing a Parameter's Value

```
# Extract the year from the StartDate parameter and compare to 2017.
if system.date.getYear(data['StartDate']) == 2017:
    # Create a new key and assign it one value if our condition is true...
    data['myResults'] = "The year is currently 2017!"
else:
    # ...or assign a different value if the condition is false. This way we can
    always assume the key 'myResults' exists.
    data['myResults'] = "What year is it? Not 2017, that's for certain!"
```

Data Sources

Static CSVs

While uncommon, Static CSVs may be accessed in a Scripting Data Source. The syntax is similar to working with Parameters.

Python - Static CSV example

```
# Take a Static CSV data source, and replicate its contents in a new key.
data['static_data'] = data['Area Data']
```

Query-Based Data Sources

Reading the contents of a query-based Data Source, such as a [SQL Query Data Source](#) or [Tag Historian Query](#), requires the `getCoreResults()` function, which returns the results in a **standard dataset**:

Python - Accessing a Query-Based Data Source's Value

```
# Query-based Data Sources are slightly different than parameters, so we must use
getCoreResults() to extract the data.
rawResults = data['keyName'].getCoreResults()

# getCoreResults() returns a dataset, so we can utilize getValueAt() and rowCount
within our scripts.
resultsCount = data['keyName'].getCoreResults().rowCount
```

When working with data sources, it is unusual to attempt to write back, since datasets are immutable. Instead, the preferred approach is to create a new key with the modified results.

Say we have a query data source named "**Area Data**" which contains four columns: **month**, **north_area**, **south_area**, and **t_stamp**. If we need to build a new data source without the **t_stamp** column, we can use the following code:

Python - Building a New Data Source

```
# build a header and initialize a pydataset
header = ['month', 'north_area', 'south_area']
filteredDataset = []

# get the results from the Area Data data source
rawDataset = data['Area Data'].getCoreResults()

# build the new pydataset out of only some of the Area Data's data keys
for row in range(rawDataset.rowCount):
    valCategory = rawDataset.getValueAt(row, 'month')
    valNorthArea = rawDataset.getValueAt(row, 'north_area')
    valSouthArea = rawDataset.getValueAt(row, 'south_area')
    filteredDataset.append([valCategory, valNorthArea, valSouthArea])

# convert the pydataset to a standard dataset
filteredDataset = system.dataset.toDataSet(header, filteredDataset)

# create a new data source with the filtered results
data['updated Area Data'] = filteredDataset
```

Report Overview

Data

Design

Preview

Schedule

Parameters

StartDate

EndDate

Data Sources

1 Query - Area Data

2 Script

+

🗑️

⬆️

⬇️

⋮

```
1 def updateData(data, sample):
2     """
3     This function has the opportunity to add additional data into the report's
4     data map.
5
6     Arguments:
7         data: This is a map, whose keys are report data keys and values should
8             be sequences, maps, scalar values, or datasets to provide information to
9             the report.
10        sample: A flag that will be True if the report data being gathered is
11            for a preview of the report. Use to avoid slow queries and calculations
12            to keep previews quick.
13    """
14
15    #build a header and initialize a pydataset
16    header = ['month', 'north_area', 'south_area']
17    filteredDataset = []
18
19    #get the results from the Area Data data source
20    rawDataset = data['Area Data'].getCoreResults()
21
22    #build the new pydataset out of only some of the Area Data's data keys
23    for row in range(rawDataset.rowCount):
24        valCategory = rawDataset.getValueAt(row, 'month')
25        valNorthArea = rawDataset.getValueAt(row, 'north_area')
26        valSouthArea = rawDataset.getValueAt(row, 'south_area')
27        filteredDataset.append([valCategory, valNorthArea, valSouthArea])
28
29    #convert the pydataset to a standard dataset
30    filteredDataset = system.dataset.toDataSet(header, filteredDataset)
31
32    #create a new data source with the filtered results
33    data['updated Area Data'] = filteredDataset
```

Nested Queries

What if our **'Area Data'** query has a nested query called **'Area Details'** that we would like to manipulate in a script? This is useful when using **Table Groups**.

Python - Script data source for nested query

```
nested = data['Area Data'].getNestedQueryResults()    # Gets results from our
parent query
subQuery = nested['Area Details']    # Gets the subquery we want -- there can be
more than one
header = ['productName', 'cost', 'triple']
alteredDataset = []
for child in subQuery:
    children = child.getCoreResults()    # children is a dataset
    for row in range(children.rowCount):
        valProductName = children.getValueAt(row, 'productName')
        valCost = children.getValueAt(row, 'cost')
        valTimesThree = None
        if valCost != None:
            valTimesThree = 3 * valCost
        alteredDataset.append([valProductName, valCost, valTimesThree])

# convert the pydataset to a standard dataset
alteredDataset = system.dataset.toDataSet(header, alteredDataset)

# create a new data source with the altered results
data['Updated Area Details'] = alteredDataset
```

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Static CSV



INDUCTIVE
UNIVERSITY

Static CSV

[Watch the video](#)

The Static CSV data source allows you to quickly craft a data source to use in your report. This data source is ideal when you need some test data to begin creating a new report. All values are brought in as strings, which means that you don't need quotation marks around any of the string values

As the name implies, this Data Source is strictly static, and does not have any built-in means to update or import new values in. In these scenarios, you may want to look into the [Scripting Data Source](#) to import CSV directly into the report, or first store the data into a SQL database and retrieve the results with a [SQL Query Data Source](#).

Format

The first line of the Static CSV will be a list of columns separated by commas. Each subsequent line after that will then be a row in the dataset, with each value separated by a comma as well. See the text below for an example dataset that has three columns and thirteen rows. You can double click the block to copy it's text.

CSV - Creating a Custom Static CSV

Equipment, Time, Site
Motor, 15, Site A
Motor, 23, Site A
Conveyor Line, 148, Site B
Pallet Wrapper, 58, Site A
Motor, 96, Site C
Conveyor Line, 23, Site B
Palletizer, 40, Site B
Conveyor Line, 56, Site A
Pallet Wrapper, 45, Site C
Motor, 43, Site C
Conveyor Line, 87, Site D
Motor, 23, Site D

Report Overview

Data

Design

Preview

Schedule

Parameters

StartDate

EndDate

Data Sources

1 Static CSV - Downtime_By_Site

+ Data Key

🗑

Downtime_By_Site

✓

↑ Data

↓

Equipment,Time,Site

Motor,15,Site A

Motor,23,Site A

Conveyor Line,148,Site B

Pallet Wrapper,58,Site A

Motor,96,Site C

Conveyor Line, 23, Site B

Palletizer,40,Site B

Palletizer,30,Site D

⋮

Conveyor Line,56,Site A

Pallet Wrapper,45,Site C

Motor,43,Site C

Conveyor Line,87,Site D

Motor,23,Site D

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Nested Queries



INDUCTIVE
UNIVERSITY

Nested Queries

[Watch the video](#)

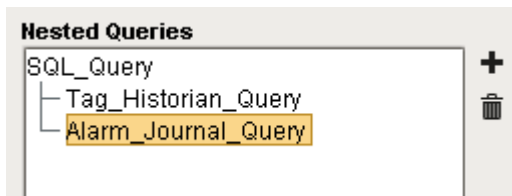
What are Nested Queries?

The simple definition is that a *Nested Query* uses the results of a previously executed query to collect data. The general structure of a Nested Data Source is one in which you have a Parent query and child queries. Each child query can then use the columns from the parent query as parameters using `{columnName}`. For example, if a parent query had a column called `machineID`, the child query can then use `{machineID}` as a parameter, and the child query will run for each row of the parent query, using the different values of `machineID` for each run. Those well-versed in SQL are probably thinking that this sounds like a `JOIN` and in fact, there are some similarities. There are also some major differences which allow Nested Data to be both easier and more powerful:

- Nesting relationships are not restricted to data in a single schema, database or even source! Nesting is easy to configure across tables, between different databases, or even with sources like the Tag Historian!
- Writing queries for nested query sources can be far simpler and easier to maintain than writing complex JOIN operations.
- Nested structures allow more control in how data is collected, allowing data structures and relationships that are more expressive.

Mix and Match Data Source Types

It is important to remember that subqueries may have a Query Type that differs from the parent. For example, it is possible to have a SQL Query return a list of Tag paths, and then use a Tag Historian Query as the subquery, referencing the Tag paths returned by the parent query.



How Nesting Works Let's use a simple data relationship to help illustrate how nesting occurs. Imagine we have data collected from two unrelated sources that look the ones seen in the tables below.

CodePK	Code
1	ZG
2	GB
3	DC

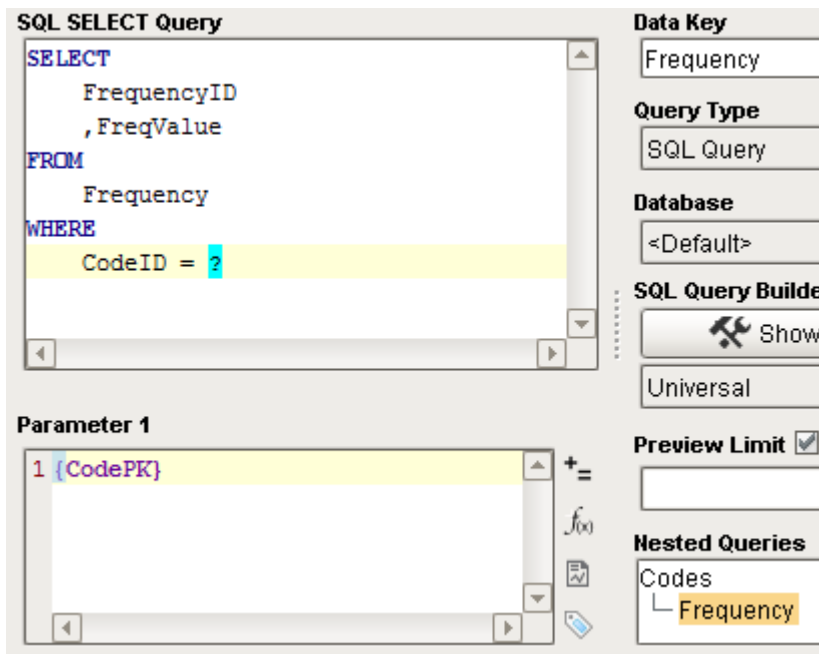
FrequencyID	CodeID	FreqValue
1	3	11
2	1	41
3	2	13
4	3	26
5	1	13
6	3	32
7	1	11

We want to create a data source connecting all these things for reporting using nesting. The trick is to identify where the two datasources connect. You may notice in the data above The CodePK column in the Codes datasource matches up with the CodeID column of the Frequency datasource. This is where we will connect the two datasources. We will make our Codes datasource the parent and the Frequency the child. The queries would look something like below.

Pseudocode - Select Statement Examples

```
-- Parent Query
SELECT CodePK, Code FROM Codes

-- In the Child Query, accessed by clicking on the Frequency leaf of the Nested
Queries Tree
-- This assumes the value of Parameter 1 would be equal to {CodePK}
SELECT FrequencyID, FreqValue FROM Frequency WHERE CodeID = ?
```



What this means for our resulting data is that the parent query is called first, and a set of results is returned to the parent query, named `Codes`. After this data has been retrieved, the Child query will execute, once for each row of the parent, substituting the value of `CodePK` into the child query where we have the `{CodePK}` reference.

The resulting data will have a structure like this:

Pseudocode - Code Frequency Structure

Codes

```
| -Row 1
|   | _ CodePK - 1
|   | _ Code   - ZG
|   |   Frequency
|   |   | -Row 1
|   |   |   | _ FrequencyID - 2
|   |   |   | _ FreqValue  - 41
|   |   | -Row 2
|   |   |   | _ FrequencyID - 5
|   |   |   | _ FreqValue  - 13
|   |   | -Row 3
|   |   |   | _ FrequencyID - 7
|   |   |   | _ FreqValue  - 11
| -Row 2
|   | _ CodePK - 2
|   | _ Code   - GB
|   |   Frequency
|   |   | -Row 1
|   |   |   | _ FrequencyID - 3
|   |   |   | _ FreqValue  - 13
| -Row 3
|   | _ CodePK - 3
|   | _ Code   - DC
|   |   Frequency
|   |   | -Row 1
|   |   |   | _ FrequencyID - 1
|   |   |   | _ FreqValue  - 11
|   |   | -Row 2
|   |   |   | _ FrequencyID - 4
|   |   |   | _ FreqValue  - 26
|   |   | -Row 3
|   |   |   | _ FrequencyID - 6
|   |   |   | _ FreqValue  - 32
```

Each row returns the CodePK and Code from the parent query, but also the results of the child query, that apply to the CodeID we get from the parent PK.

Now, in our Report Design, we will have access to our datasource Frequency that linked these two sets of data together through a shared value.

Special Considerations

Nested Queries are powerful and easy to use, but users should be aware of runtime implications. Imagine the scenario above, where we have two sets of data, each with 5000+ rows. When our child

query executes, each row of its query is going to require a lookup from the parent. For most common sets of data and database sizes, this won't be an issue, but it's possible to imagine that instead of just one child query, we have a dozen.

In addition, some of those children also have many children. It's very easy to see in this scenario how exponential growth occurs and our system performance may suffer. Most report designers will limit query sizing as oversized data structures are simply not as easy to work with. However, if you feel an urge to generate massive complex trees of million line queries, you may be waiting a while.

Nested Query Example - Equipment Downtime

Here in this example, we have two database tables: an equipment table, which contains all of the equipment we have, as well as a downtime table, which contains a list of every downtime event. The tables are put together like the tables below.

Equipment_Table

Id	Name	Description
25489	Conveyor Line	Transfers product
55684	Labeler	Makes labels
88456	Palletizer	Makes pallets
626145	Filler	Fills tanks

Downtime_Table

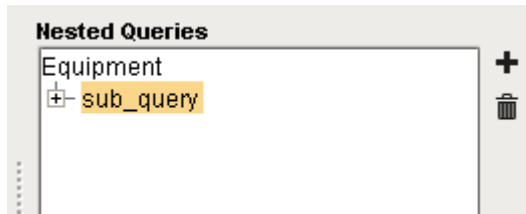
Id	Equipment_id	Cause	Minutes_Down
1	25489	Backup	22
2	55684	Out of labels	25
3	55684	Stuck labels	15
4	88456	Misalignment	38
5	626145	Overflow	50
6	25489	Scheduled Maintenance	12
7	626145	Scheduled Maintenance	40
8	88456	Misalignment	55

1. Create two tables in the database that are similar to the tables listed above.
2. Open a report, and navigate to the **Data Panel**.
3. Click the **Plus +** button and add a **SQL Query** to the data sources.
4. Change the **Data Key** property to **Equipment**.
5. Type the query below into the query area. Here we are pulling out three columns of our equipment table, and giving each column a more descriptive name. Pay close attention to the spelling of the aliased names (names following the **AS** keywords), as we will have to reference **EquipmentIDNumber** later in our subquery.

SQL - Equipment Table Query

```
SELECT id AS EquipmentIDNumber,  
       Name AS EquipmentName,  
       Description AS EquipmentDescription  
FROM Equipment_Table
```

6. On the right of the Nested Query section, click the **Plus +** button to create a sub_query.



7. Select the sub_query, and rename the Data Key property to rename it. In this example, we will use the name **EquipDowntime**.

8. Type the following query into the query area.

SQL - Downtime Table Query

```
SELECT cause AS DowntimeCause,  
       minutes_down AS DowntimeMinutes  
FROM Downtime_Table  
WHERE Equipment_id = ?
```

Here we pull in our downtime table, but we only need the cause and minutes_down, since we are already grabbing the Equipment_id from the first query. The WHERE clause is where we link this query to the parent with the equipment ids.

9. In the **Parameter 1** field, type the following:

Expression Language - Referencing a Parameter

```
{EquipmentIDNumber}
```

Note that we are directly referencing one of the aliased column names from the **Equipment** query. That is all it takes to make a Nested Query! We now have two separate tables being called and linked together by the equipment id. 10. To check that your queries worked, go to the Preview tab and look at the XML data that comes up. You should have a data set inside (indented) each row of Equipment data. 11. If you'd like, you can continue with this example and either use this data in a **table group**, or in a **nested chart**.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Report Design



INDUCTIVE
UNIVERSITY

Report Designer

[Watch the video](#)

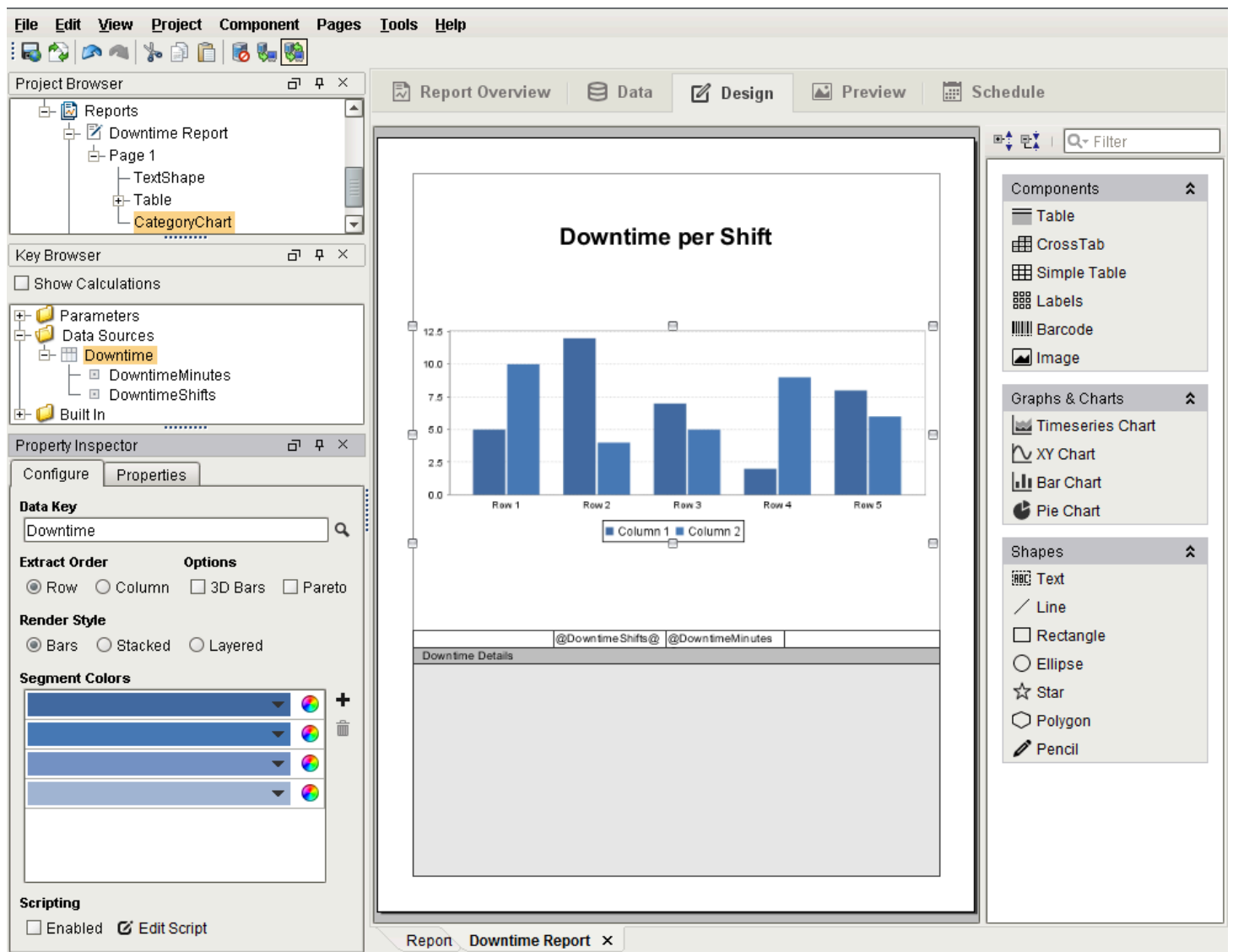
Report Designer Interface

The Report Workspace's Design tab lets you design reports with the same intuitive feel, familiar layout, and drawing tools that you get when designing windows in Ignition. Just like designing Vision windows, you can design reports using the same familiar drag-and-drop method and choose from a library of graphs, charts, tables, and images.

Looking at the screenshot of the Report Designer, you'll notice that an open Report Resource in the **Project Browser** can be expanded to provide information at a glance. This tree lets you visualize the relationship and hierarchy of Report Design elements on the page. Find an element in the tree by selecting items on the Design Panel, or find an element on the page by selecting it from the tree. As you do so, you'll notice that the bottom left of the default Report Designer will change to provide configure panels and/or property tables depending on the selected item.

Just below the Project Browser is the **Key Browser** which provides the **Data Keys** we use to reference data in our report. The Key Browser is home to all your Data Sources, **Parameters**, as well as a number of built-in calculation keys to speed you through the report design process. When you configure a data source and switch to the Design tab, a sample of all your queries is run on the Gateway to get information about the structure of your data. Columns in your data are represented as children in the Key Browser tree.

The right side of the panel is where the component palette lives, providing a number of powerful components, charts, and shapes which are used to build the layout and visualization of your report.



Report Design Components

Designing reports is where the fun begins. The Design Panel has a Report Design Palette with a host of components, charts, and shapes that when used together help you design and create meaningful, informative, and professional reports. **Tables** are used to collect and store varying amounts of data. Sophisticated charts can be created from the data collected in a table. **Images** and Shapes can add that extra touch of class and polish making a report stand out.

Just like designing in Vision windows, Report Design components have properties which are editable in the Property Inspector which is located on the lower left side of the Design Panel. Below is a brief overview of the report design tools you can choose from for building your reports.

i SOME PROPERTY CHANGES DON'T TAKE EFFECT IN THE DESIGN PANEL

Note that modifying some component properties, such as the **Font** or **Fill Color**, will immediately cause a visual change in the **Design Panel**.

However, other properties, such as the **Date Format**, **Number Format**, and **Overflow Behavior** properties on a Text Shape component will not cause an immediate change. The properties described here wait until the report executes. Because of this, you will only see the results of these changes in the **Preview Panel**.

It is highly recommended that you review your report in the Preview Panel often. This provides an opportunity to visualize what the resulting report will look like.

Report Objects

Report Objects are the actual report and page components, and each have their own properties. They have a host of settings and properties that allow you to customize the appearance of your reports based on your requirements.

- **Report Object** - allows you to set the report properties for paper size, dimensions, orientation, margins, and much more.
- **Page Object** - allows you to add and remove pages in your report, as well as configuring additional properties that can add that creative touch to your reports.

Reporting Components

The Reporting Components are the foundation for creating useful reports. There are variety of different tables that you can choose from based on the type of data you're collecting. There are even Label and Barcode components you can use to create shipping and product labels. Here is a brief description of all the report design components.

- **Table** - allows you to display tabular data in a variety of ways using the Table properties. Tables are configurable and highly flexible giving Reporting users the ability to flexibly layout and organize tabular data. They also support advanced grouping and pagination.
- **CrossTab** - similar to a Table component, the CrossTab component is commonly used to summarize the relationship between two categories of data by showing summaries of cross sections of the data source. The CrossTab component creates a dynamic pivot table from your data.
- **Simple Table** - is a basic grid table that dynamically creates new rows and columns for rows returned by the Data Keys on the component.
- **Labels** - used to print out a host of different types of labels such as mailing labels, name tags, and more. You can create a set of labels using preset or custom sizes.
- **Barcodes** - allows reports to contain dynamically generated barcodes. The following formats are supported: Code 128, Code 39, QR Codes, PDF 417, Aztec, Data Matrix, to name a few. Refer to Barcode in the Appendix section to see all the supported formats.

- **Images** - can be embedded into reports. They can be dropped onto reports, pulled as binary data from a data source, or even a URL.

Reporting Charts

Report charts allow you to display your data in a variety of graphical ways. There are four types of charts so you can show the data any way you like. Below is a brief description of the different types of charts you can choose from to build your reports.

- **Timeseries Chart** - is a type of line chart whose domain, or X-Axis, represents a timestamp or datetime. The Timeseries Chart is a great way to display Tag History.
- **XY Chart** - is very similar to the Timeseries chart with the main difference being the domain, or X-Axis which is a numeric value and not a date.
- **Bar Chart** - is a very easy chart to use that provides a bar representation of any numeric values, and because it has many properties, it can greatly impact how the data is presented. It is useful for displaying series with relative values.
- **Pie Chart** - is just what the name implies, it displays items as pie wedges. It represents relative quantities as wedges of a circle.

Shapes

You may have seen some of these shapes when designing your Vision windows. The shapes you see in the Report Design Palette are shape tools. They are not dragged from the design palette on to your report, instead you select a shape tool and begin to draw your shape on to your report. You create its size, height, and shape. Some of the shape tools allow you to create various shapes as well as edit them. Just like components, once a shape is created, you can change its fill color, stroke color, stroke style, and more. Each shape has its own properties. Here is a list and brief description of the available shapes in the Report Design Palette.

- **Text Shape** - Creates a text area for static or data key bound content.
- **Line** - Creates a straight line. It can run north-south, east-west, or diagonally.
- **Rectangle** - Creates a square or a rectangle. Once a rectangle or square is created, you can use the handles to change its height and width.
- **Ellipse** - Creates circles and ellipses. Once an ellipse or circle is created, you can use the handles to change its size.
- **Star** - Creates a star or a polygon. Once the star is created you can use the handles to resize it.
- **Polygon** - Creates custom polygon shapes by drawing and connecting the lines. Once all the lines are connected to form the polygon shape, you can use the handles to change its size. Double click on any of the individual lines to change the size and shape of the selected line.
- **Pencil** - Draws freehand shapes with smooth paths.

To learn more about report design components, refer to the [Report Design Tools](#) section, and to the Appendix for details on each component.

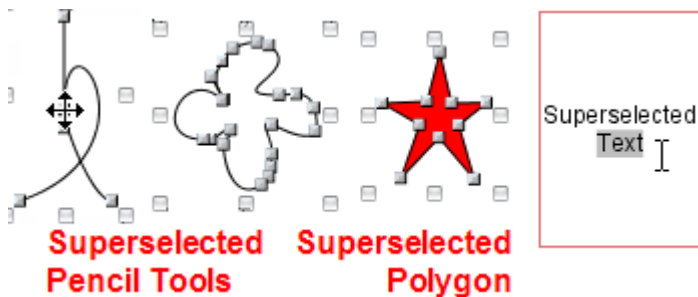
Selection and Alignment

Selecting components is done with the selection tool (), which is the default tool in the Report Designer whenever another tool is not activated.

Reporting has a "deeper" selection model than the Ignition Designer. Simple object selection is done by single clicking an object, and is typically used to move an object around or resize it. "Selecting deep" is done by double-clicking on the object to get into the report hierarchy. For instance, if you group two rectangles together, you can select the individual rectangles by double clicking "into" the group. Visualizing selection is simplified by viewing the Project Browser to see which node in the Report Resource tree is selected.

Super Selection

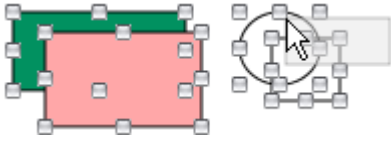
Super Selection refers to an editing state that some shapes go into when double clicked. Text is the most common of these. When a text box is selected you can move and resize it. When it's super-selected, you can place the text cursor or select a range of characters and insert or delete text. The polygon and pencil are two other basic tools that support Super Selection.



Multiple Selection

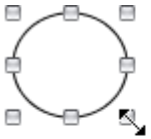
Multiple Selection of components can be done two ways:

- Clicking and dragging the mouse over a range of the report. Every object the selection rectangle touches becomes selected.
- Hold the shift key while making a selection or dragging a selection rectangle. Shapes selected by that action will be added or removed from the currently selected shapes.



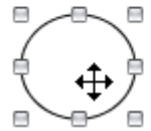
Resizing and Moving Objects

To resize an object, first select it with a single click, then left or right click, and drag one of the 8 resizing handles. To move an object, left or right click on the shape, and drag it anywhere on the report. Both resizing and moving operations support shift-dragging.



Select object then use any of the 8 handles to resize

 **Resize Both Directions**
 **Resize Vertical**
 **Resize Horizontal**



Select object then drag from anywhere but handles to move

Alignment

Alignment is accomplished by selecting multiple objects, then choosing from any of the **"Make"** commands in the **Component Menu** to align objects. You can align objects in rows by their top, center or bottom border, and align columns by their sides or center. You can make objects the same size (width and height), and equally space rows and columns horizontally and vertically.

Z order defines relative order of objects when they overlap. Simply select the object and click "Bring to front" or "Send to back" in the Component Menu.

Shift Drag

Holding the shift key while you drag shapes around your report will constrain movement to horizontal, vertical, or 45 degrees.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

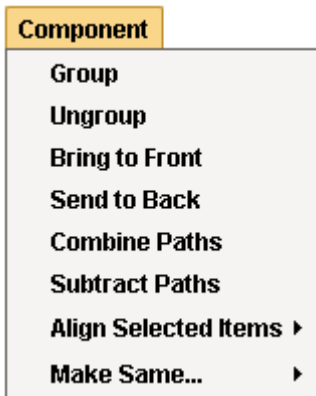
Report Design Tools

Object layout is an important aspect in creating professional reports. The Reporting Module provides a host of intuitive tools and functions to help you design and edit reports. Ignition Reporting uses a WYSIWYG (what you see is what you get) approach.

Report Menus

The Ignition Designer menubar provides quick access to many common Reporting Design functions. The Component and Pages menu items are added to the menubar when the Report Design Panel is open. Many of the menus, such as **File, Edit, and Tools** are similar to other areas of the Designer. Reporting-specific Toolbar and Menu functions are described below.

Component Menu



The Component menu allows you to modify the layout of objects in a report. To perform any of the Component menu functions, you must first select an object or multiple objects before you're able to perform the function.

Menu Item	Function
Group/Ungroup	Grouping allows you to merged selected shapes and makes them behave as one with respect to: selection, moving, and resizing. To "drill down" to individual objects, superselect the grouped object. Ungroup separates grouped shapes.
Bring to Front/Send to Back	All shapes have an order on the page that determines what is drawn on top when two shapes overlap. These options allow you to move a selected shape either forward or backward. This is also known as the Z Order.
Combine/Subtract Paths	Takes multiple overlapping shapes (such as a rectangle and an oval) and combines them into a single shape using the combined paths. A powerful tool to construct complex shapes.
Alight Selected Items - Make Row Top/Center/Bottom	Quickly align several shapes in a row, either by their top, center, or bottom border. This is particularly useful when shapes are of different heights.
Align Selected Items - Make Column Left/Center/Right	Same as above, but for columns, aligning their sides or center.
Equally Space Row/Column	Equalizes the distance between shapes horizontally or vertically.
Make Same Size, Height, Width	Make several shapes the same width, height or both.

Pages Menu

Pages

Add Page

Add Page Previous

Remove Page

The Pages menu allows you to add or remove pages in the report. Adding and removing pages in the Design Panel is easy. Under the **Page menu**, click on either **Add Page** to add a page after the

currently selected page, or **Add Page Previous** to add a page before the currently selected page. To remove a page, open the page you want to remove and select **Remove Page**.

Menu Item	Function
Add Page	Adds a page to the current open document, after the currently selected page.
Add Page Previous	Adds a page to the current open document, before the currently selected page.
Remove Page	Removes the currently selected/visible page in the current open document.

Toolbar



The Toolbar provides a variety of functions to assist in editing while designing a report. Here is a complete list of editing functions and their descriptions in the table below.

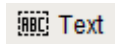
Toolbar Item	Function
	Toggles bold of the selected text
	Toggles <i>Italic</i> the selected text
	Toggles underline of the selected text
	Aligns the selected text left
	Centers selected text
	Aligns the selected text right
	Adds a \$ to the data key
	Adds a % to the data key
	Forces a decimal place on an integer value in a text shape
	Add the number of allowed decimal digits
	Subtract the number of allowed decimal digits
	Increase the size of the report in view
	Decrease the size of the report in view
	Have the report fill the available space

Shape Tools

You may have seen some of these same shapes tools when designing your Vision windows. The shapes you see in the Report Design Palette are shape tools: Line, Rectangle, Ellipse, Star, Polygon, and Pencil. Shape tools allow you to create various shapes as well as edit them to create other

shapes. They behave a little differently from the typical drag and drop function that you perform on a Vision window. They are not dragged from the design palette on to your report, instead you select a shape tool on the design palette to make the tool active and begin to draw your shape on to your report. You create its size, height, and shape. You'll also notice the Property Inspector will change to reflect that specific tool's properties once the tool is active. Just like components, once a shape is created, you can change its fill color, stroke color, stroke style, and more. Each shape has its own properties. To learn more about shapes, refer to the section on [Report Design](#).

Text Shape



The **Text Shape** is worth mentioning here because it is a fundamental component used to create text within a report. The Text Shape is used for report titles, customized page headers and footers, and of course, any additional text you want to add to your report. Simply drag a Text component on to report and begin typing. You can move the Text component around, expand it, or shrink it by using the handles when the component is selected. You can even change the font type, size, and color. The Text Shape also has a lot of properties associated with it, including Data Key Format Properties for Date and Number formats by choosing a format from the list of available templates.

Headers and Footers

Creating headers and footers is just like creating any other set of objects in your report. There is no explicit header or footer functions, or specific area on the report where they belong. Typically, headers and footers are placed at the top and bottom of the first page respectively, using the Text component or the **Built-In Keys**.

When designing your report, the key is sizing and positioning of components within your page around your header and footers. If you have a lot of data in your data table, your report will automatically create additional pages. Each new page that the table creates will have the same header and footers.

You can use the Text component to customize your header and footers or you can use the Built-In Keys from the Key Browser. The Built-in Keys allow you to drag and drop keys to your header and footer areas such as Path, Timestamp, Date, Page number, and more. So place your header and footers where you want them on the first page, and the Report Designer will automatically place them on all additional pages.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Report Parameters



INDUCTIVE
UNIVERSITY

Parameters

[Watch the video](#)

As we covered in the overview, the Data Panel is pretty simple at first glance. When creating a new report, there is an empty list of Data Sources as well as pre-made Parameters. **Parameters** are Ignition **Expressions**. These parameters resolve at report execution time and provide a convenient way to dynamically specify content for your report. These expressions are expected to resolve to specific types.

The Parameter Types available in the ComboBox will be familiar to anyone who has used Expressions, and include:

- **Date**
- **String**
- **Long**
- **Double**
- **Boolean**
- **Dataset**
- **Binary Data**

Adding Parameters function similar to custom properties on Vision windows and components. Their value can be shown directly on your report, or be referenced by data sources and other parameters. Parameters are given a default value when first created, but this value can be overridden once the report runs.

How Parameters Are Used

Parameters share some similarities to Custom Properties in Vision components in that they allow an Ignition user to specify dynamic data to be used in reports at runtime. For example, a manufacturer may want to generate reports for the active production lines every Monday morning, but you don't know ahead of time which production lines will be running. We can create a parameter called *activeProductionLines* and use an expression or Tag value to determine which lines were active, and use the parameter in a SQL Query datasource to only include data from active lines.

Parameters can be declared with an empty (null) value. In this event, it can be fed values from the Vision Report Viewer component, or in the Report Scheduling panel. The Report Viewer can use parameters to bind to data through Ignition Bindings. If a parameter is supplied a default value in the Report Data panel (we call these default parameters), that value will be automatically be supplied to the Report Viewer component. If a parameter has no default, the Report Viewer and Preview Panel will notify the user that a parameter is undefined. It is important to note that both Scheduled Report and Vision component parameters take precedence over default parameters, so any default parameter can easily be overridden while also providing a baseline value.

NOTE

Default values for parameters are evaluated in the gateway, not by clients. As such, the expressions have a Gateway scope.

Default Starting Parameters

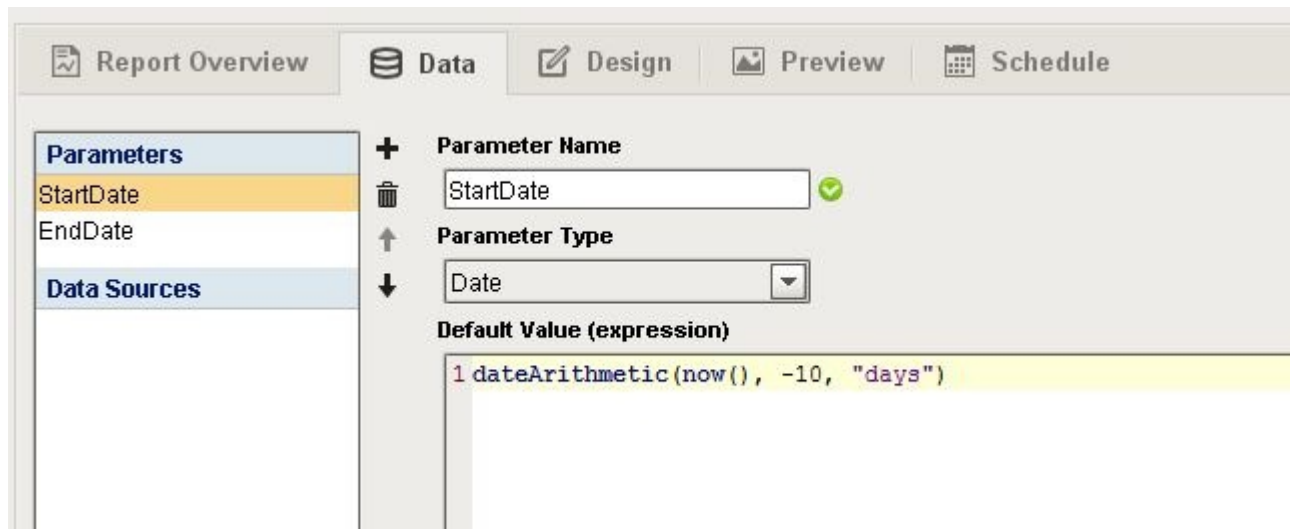
When you first create a report, you start with two parameters: **StartDate** and **EndDate**. EndDate will have a default value of 'now()', while StartDate will have a default value of 'dateArithmetic(now(), -8, "hr")'. The default value of the StartDate can be modified from 8 hours to 10 days by editing the Default Value expression.

StartDate Default

```
dateArithmetic(now(), -8, "hr")
```

StartDate Modified

```
dateArithmetic(now(), -10, "days")
```

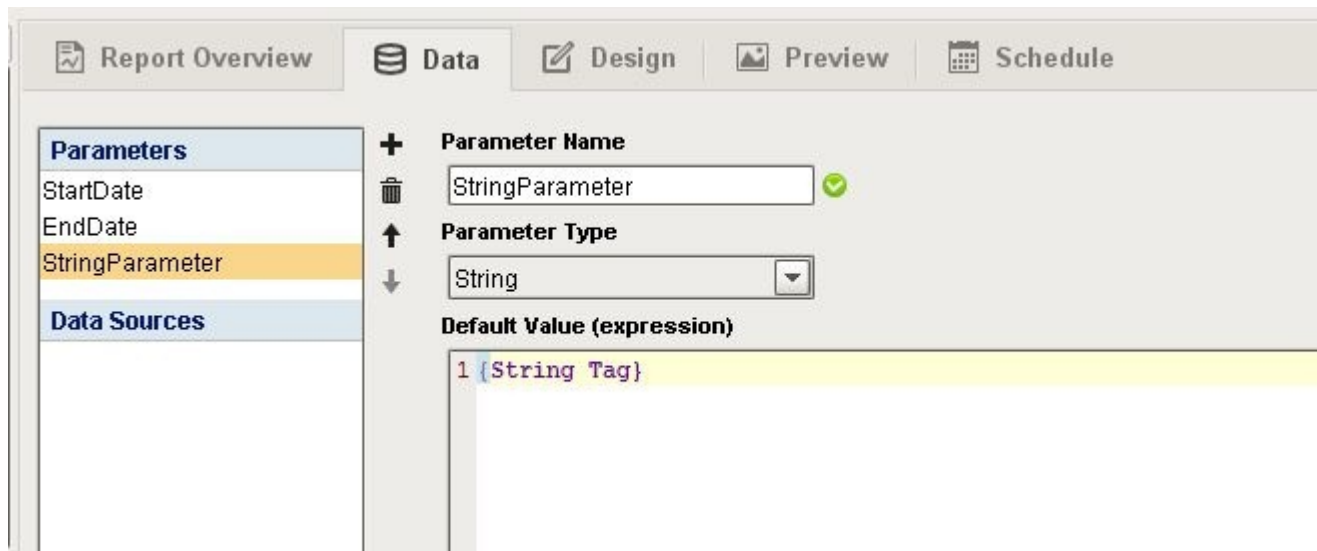


You can also delete both StartDate and EndDate parameters if you do not need date parameters in your report.

Creating New Parameters

New parameters may be added by clicking the **Add +** icon.

Once created, a default value should be given to the parameter. In addition to expressions, you can also have your default value reference a Tag using the **Tag**  icon on the right of the expression area.



You can also use a literal value. Normal Expression languages syntax applies, so **dates** and **strings** must be wrapped in quotation marks, but **numerical** values can be written without quotes.

The screenshot shows the 'Data' tab of the Report Designer. On the left, the 'Parameters' list includes StartDate, EndDate, StringParameter, IntegerParameter (highlighted), and Data Sources. The main configuration area for IntegerParameter shows:

- Parameter Name:** IntegerParameter (with a green checkmark)
- Parameter Type:** Long (selected from a dropdown)
- Default Value (expression):** 1 987654321

Finally, you can reference other parameters. In the example below, **BoolParameter** is referencing the value of **IntegerParameter**.

```
if ({IntegerParameter} = 1000, 1, 0)
```

NOTE

When referencing other parameters, you must type the name of the parameter exactly within a set of curly braces , including capitalization. Also, you can only reference parameters that are above the current parameter. Thus, while **BoolParameter** can reference the **IntegerParameter**, the **IntegerParameter** may not reference **BoolParameter**.

The screenshot shows the 'Data' tab of the Report Designer. On the left, the 'Parameters' list includes StartDate, EndDate, StringParameter, IntegerParameter, BoolParameter (highlighted), and Data Sources. The main configuration area for BoolParameter shows:

- Parameter Name:** BoolParameter (with a green checkmark)
- Parameter Type:** Boolean (selected from a dropdown)
- Default Value (expression):** 1 if({IntegerParameter} = 1000, 1, 0)

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

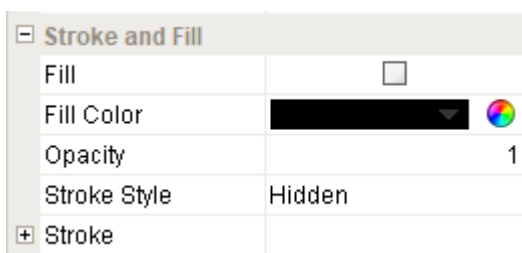
Stroke and Fill Properties for Reports

As you probably learned working with Vision components, each component has its own unique properties. The **Report Design** components are no different. The Report Designer has some of the same components like tables, bar charts, graphs, and various shape tools, but their properties are different from the Vision components because they are designed to go in reports. For example, Stroke and Fill properties control the appearance of the reporting components so you can change stroke style or border, create a dashed line around the shape, change the width of the lines and the color, and more. Stroke Style and Stroke properties can be particularly useful when using a Table in a report. You can make objects in a table and add an outline or set any edges of they outline to show to make your report stand out.

Stroke and Fill Properties

Stroke and Fill properties are commonly used by many reporting components. All of the components, charts, graphs, and shapes in the Report Design Palette use the same Stroke and Fill properties. These properties affect how the components look in a report. There are five basic Stroke and Fill properties:

- **Fill** - if this property is marked true, the shape will fill its space with color.
- **Fill Color** - if the Fill property is selected, the color you select will fill the shape.
- **Opacity** - defines how opaque the color is in the shape, between 0 and 1.
- **Stroke Style** - defines what type of stroke or border to use.
- **Stroke** - An expandable list of properties for the chosen stroke style.



The Fill, Fill Color, and Opacity properties are pretty easy to understand, so we are going to focus on Stroke Styles and Stroke properties in the following sections so you can take full advantage of how to use these properties to make professional looking reports.

Stroke Styles

Stroke Style defines what style of stroke or border to use. Primarily, the Stroke Style controls the type of line drawn around the component (table, chart, shape, etc) that you are using. There are four pre-configured Stroke Style templates you can choose from.

- **Hidden** - no stroke or border
- **Shape Outline** - a solid line outline
- **Border** - border or rectangle shape
- **Double** - two parallel lines

Each of these Stroke Styles come with their own set of properties. In the Design panel, select a component, and choose any one of the style templates from the Stroke Style property in the Properties tab. You can use the Stroke Style as is, or double click on any of the rows or cells to change its individual properties. For example, you might want to change the thickness of the lines or create a dashed line around the border of a component. This is easy to do: select the component, row, or cell, and change the Width under the Stroke property (thickness of the lines is specified in pixels). Then view your report with the Preview tab. You can keep repeating this step until you get the desired result.

Stroke Properties

Each Stroke shares some properties with other strokes, but have their own unique properties. The only exception is that the **Hidden** Stroke Style does not have any properties. **Note:** If you switch between stroke styles your values will be overwritten by defaults. This is not true when switching to the Hidden type.

The properties for each Stroke Style are shown below.

Shape Outline Stroke Properties

Shape Outline is a solid line outline around a component or shape. It has the following properties that you can use as is, or change.

Property	Description
Color	Color to use for the stroke.
Dash Pattern	A "Dash Pattern" string to specify the number of pixels on and off. For example "5,10"
Width	Width of the shape in pixels.

Stroke Style - Shape Outline Example

This example shows a report with a Shape Outline around the Table component at the bottom of the image. Select the whole table to set stroke and fill for it. The property values used to create the outline around the data are shown in the properties list in the table below.

Vegetable Report



● Squash ● Zucchini ● Carrots
● Potatoes ● Beans ● Peppers

Vegetable	Qty
Squash	10
Zucchini	18
Carrots	25
Potatoes	8
Beans	33
Peppers	9

Stroke and Fill	
Fill	☐
Fill Color	
Opacity	1
Stroke Style	Shape Outline
Stroke	
Color	
Dash Pattern	
Width	2

	Vegetable	Qty	
Vegetables Header			
	@Vegetable@	@QTY@	
Vegetables Details			

- Stroke Style - Shape Outline
- Stroke Color - Red
- Width - 2 pixels

Border Style Stroke Properties

The **Border** Stroke Style is similar to the Shape Outline, but it gives you the option to disable the any of the lines around the border; bottom; left; right; or top.

Property	Description
Color	Color to use for the stroke.
Dash Pattern	A "Dash Pattern" string to specify the number of pixels on and off. For example "5,10"
Border Bottom	If true, show the border on the bottom side of the shape.
Border Left	If true, show the border on the left side of the shape.
Border Right	If true, show the border on the right side of the shape.
Border Top	If true, show the border on the top side of the shape.
Width	Width of the stroke in pixels.

Stroke Style - Border Example

This example used the Border Stroke Style for the Equipment Summary cell at the bottom of a Table. (Total: @total.Downtime@Minutes). Three images appear inside the outline because they are in the Equipment Header (**unstructured**) row. The property values are shown for the Table, Row, and Cell in the property lists below.

Equipment



Labeler

Out of labels	Duration (Minutes)	Date
	50	Jan 20, 2017
	21	Feb 12, 2017
	71 Minutes	

Misalignment	Duration (Minutes)	Date
	33	Jan 20, 2017
	33 Minutes	

Filler

Overflow	Duration (Minutes)	Date
	22	Feb 20, 2017
	30	Feb 28, 2017
	52 Minutes	

Maintenance	Duration (Minutes)	Date
	19	Feb 25, 2017
	19 Minutes	

Conveyor Line

Backup	Duration (Minutes)	Date
	45	Jan 21, 2017
	45 Minutes	

Maintenance	Duration (Minutes)	Date
	47	Feb 13, 2017
	47 Minutes	

Total: 267 Minutes

Equipment Table in the Designer

Equipment 			
Equipment Header Standard			
@Equipment@			
Equipment Details Standard			
	@Cause@	Duration (Minutes)	Date
Cause Details Standard			
		@Downtime@	@T_stamp@
Data Details Standard			
		@total.Downtime@ Minutes	
Data Summary Standard			
		Total: @total.Downtime@ Minutes	
Equipment Summary Standard			

Table Property Values

Stroke and Fill	
Fill	☐
Fill Color	
Opacity	1
Stroke Style	Border
Stroke	
Color	
Dash Pattern	
Border Bottom	☒
Border Left	☒
Border Right	☒
Border Top	☒
Width	1

- Stroke Style - Border
- Stroke
 - Border Bottom - Enabled
 - Border Left - Enabled
 - Border Right - Enabled
 - Border Top - Enabled

Data Summary Cell

Shows the Total Downtime Minutes for each downtime cause for each piece of equipment.

Text Properties	
Text	@total.Downtime@ Mi...
Text Color	 
Character Spacing	0
Coalesce Newlines	☐
Font	Arial Bold 12.0 (Arial)
Horizontal Alignment	Left
Line Spacing	1
Margin	1; 2; 0; 2
Overflow Behavior	Grow row
Underlined	☐
Vertical Alignment	Top

- Text Color - Red,Bold

Equipment Summary Cell

Shows the Total Downtime Minutes for all causes and for all equipment.

Stroke and Fill	
Fill	☒
Fill Color	 
Opacity	1
Stroke Style	Border
Stroke	
Color	 
Dash Pattern	
Border Bottom	☒
Border Left	☒
Border Right	☒
Border Top	☒
Width	2

- Stroke Style - Border
- Fill - Enabled
- Fill Color - Yellow
- Width - 2 pixels

Double Style Stroke Properties

By default, the **Double** Stroke Style consists of two parallel lines around the component. You can choose to disable any of the lines at the bottom, left, right, or top. You can also change the width of both lines and the separation between these lines, if desired.

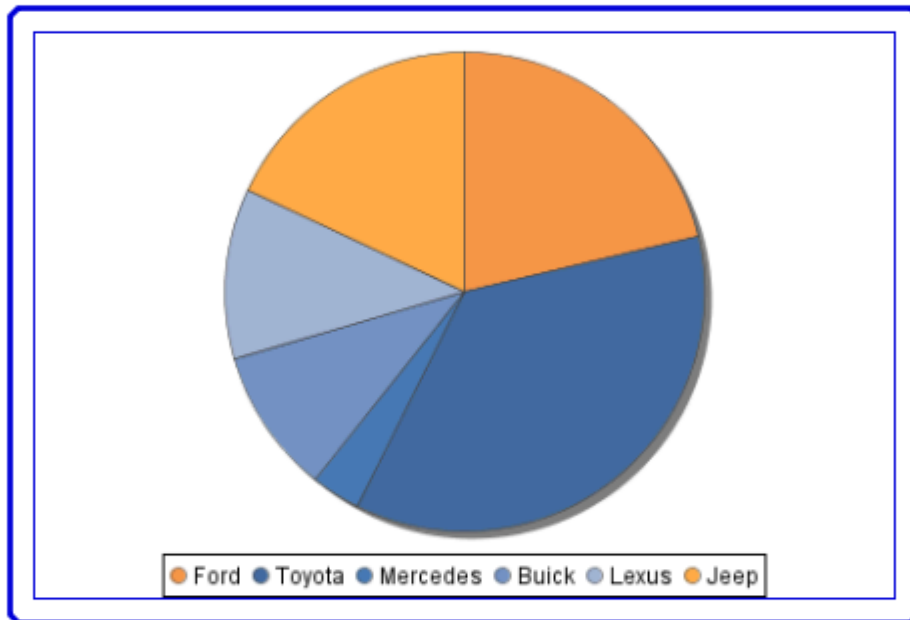
Property	Description
Color	Color to use for the stroke.
Inner Width	Width of the inner border in pixels.
Outer Width	Width of the outer border in pixels.
Position	Position of borders relative to the bounds of the shape. • Outer on path - Places the outer border on the edge of the component and the inner border inside the component. • Inner on path - Places the inner border on the edge of the component and the outer border outside the component. • Centered about path - Centers the whole border (based on inner+outer+gap widths) on the shape edge. • Gap on path - Centers on the gap on the shape edge.
Separation	Separation between inner and outer borders in pixels.

Stroke Style - Double Example

Here is an example of a Pie Chart with a Double border used in a report. The property values for the Border Stroke Style are listed in the properties list table below.

Used Car Inventory Report

August 8, 2017



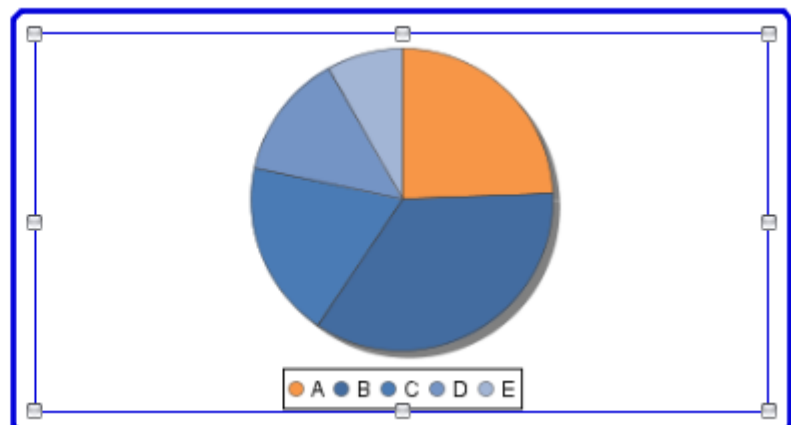
Ford	13
Toyota	22
Mercedes	2
Buick	6
Lexus	7
Jeep	11

Controller
Aug 8, 2017

Car Inventory

1 of 1

Stroke and Fill		
Opacity		1
Stroke Style	Double	
Stroke		
Color		
Inner Width		1
Outer Width		3
Position	Inner on path	
Separation		10



- Stroke Style - Double

- Stroke Color - Blue
- Inner Width - 1
- Outer Width - 3
- Position - Inner on Path
- Separation - 10

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

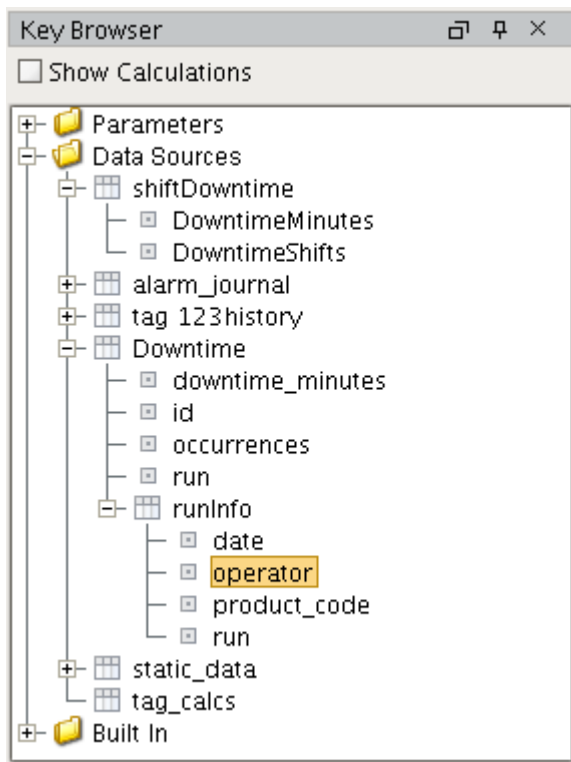
For documentation on the latest stable release, see the [latest version](#) (8.3).

Data Keys

Data Keys

In the Reporting Module, we use **Data Keys** to pull values from data sources and show them on the report. In simple terms, Data Keys are placeholders for your data. The simplest reference to data is a simple Data Key. At report generation time, these keys resolve to the values (or sets of values) provided by the data source. Additionally, Data Keys may be used as expressions, which are referred to Keychain Expressions.

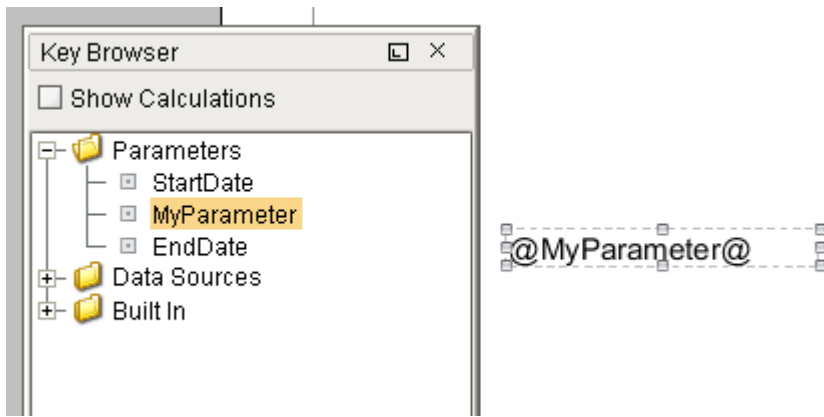
As you add Parameters and Data Sources to the **Data section** of your report, they will appear in the Key Browser's **Parameters** and **Datasources** folders.



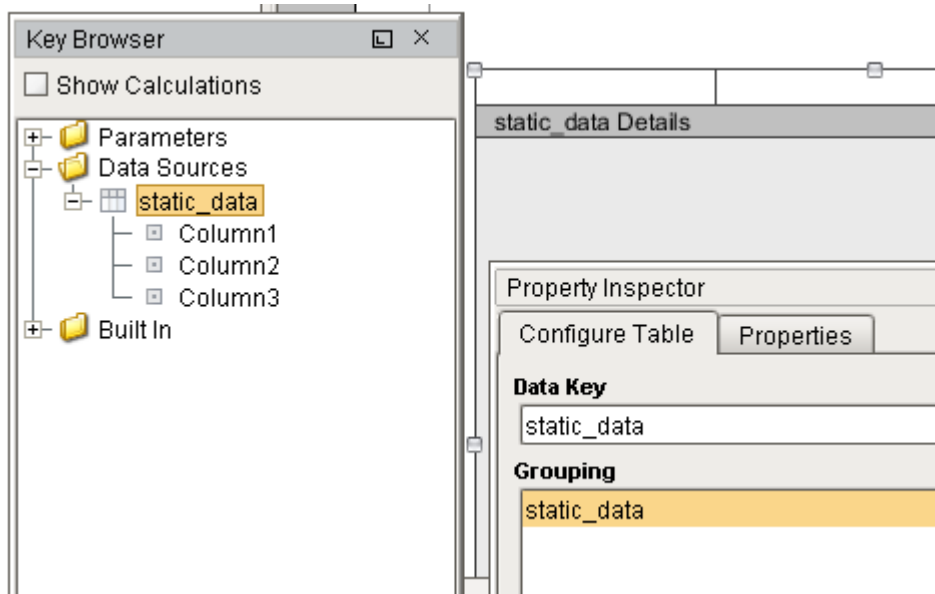
Data Keys on the Report

Data Keys are enclosed in the "@" character when utilized by components in the report. They may be typed manually, or dragged directly from the Key Browser.

Keys that contain a single value will create a **Text Shape** when dragged onto the report.



Keys that represent datasets will create a **Table** component, and configure the Data Key property to use the key.



Built-In Keys



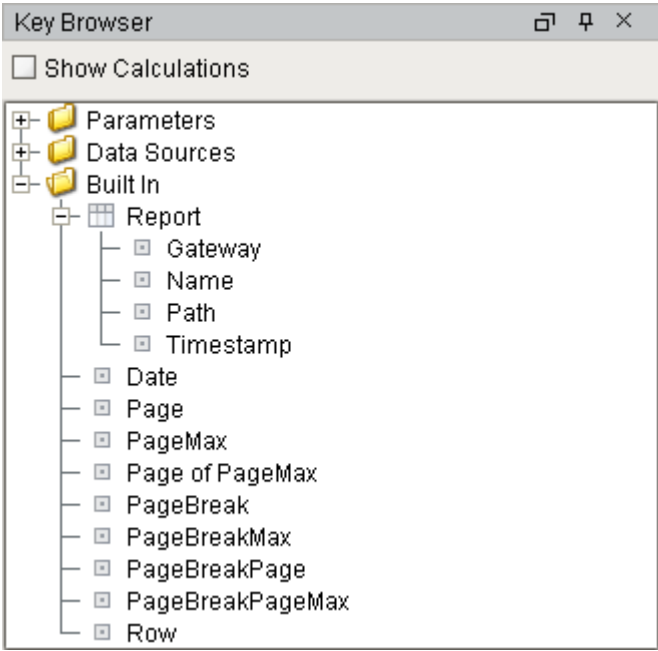
INDUCTIVE
UNIVERSITY

Built-in Keys

[Watch the video](#)

Built-In Keys provide a lot of useful information on your report at a glance. The Built-In keys are found in the Key Browser. Expand the Built-In folder and you'll see all the default keys, including a Report folder. The keys in the Report folder are specifically related to the report: Gateway name that the report is located in, report name, folder path from the Project Browser to the report, and the Timestamp of the Gateway. The other Data Keys are related to information you may want to add to a report like the date you are viewing or printing the report, page number, number of total pages, and more.

Here's a screenshot of the Key Browser showing all the default Built-In Keys that can be used on a report.



The tables below show the Built-In Report Data Keys and Built-in Data Keys along with a brief description of each key.

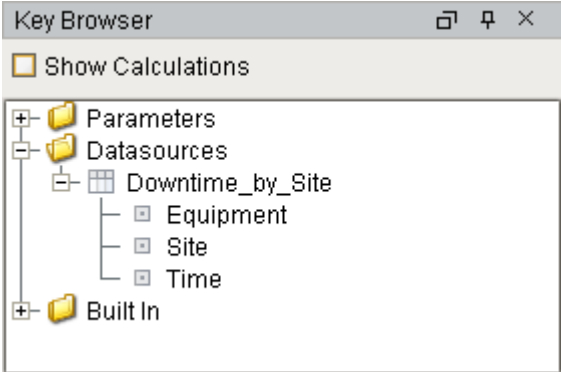
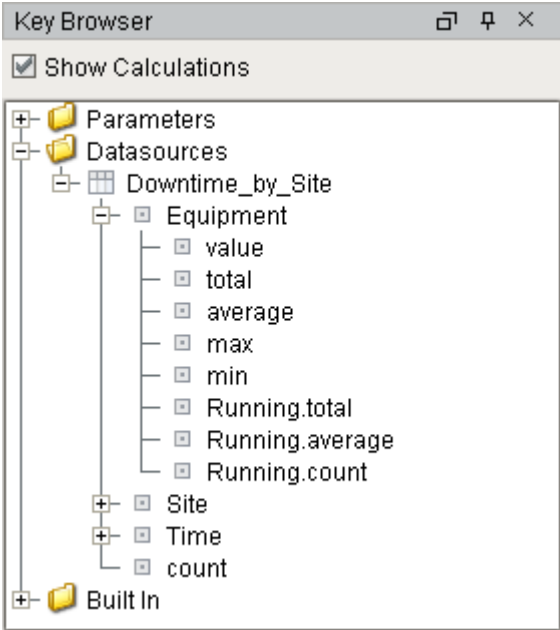
Built-In Data Key Description

Key	Description
Report	The key has multiple sub-keys that provide meta-data about the report. • Gateway: Name of the Report's Ignition Gateway • Name: Name of the Report • Path: Path to the Report in the Project Browser • Timestamp: The Gateway's current timestamp
Date	The current date/time
Page	The current page
PageMax	The total number of pages in the generated report
Page of PageMax	Shows current page number and the total number of pages in the report
PageBreak	The number of explicit page breaks encountered
PageBreakMax	The total number of explicit page breaks in generated report
PageBreakPage	The number of pages since last explicit page break
PageBreakPageMax	The total number of pages in current explicit page break
Row	Shows the current row number. Must be used in a table

Show Calculations Property

In the Key Browser, the **Show Calculations** property will add several aggregates to each key. These allow your reports to easily display things like the total of a key. These calculations are typically used in the **summary row** on the Table component.

Once Show Calculations is enabled, the Key Browser will refresh, and each key will be expandable. Expanding a key will show the available calculations.

Show Calculations Disabled	Show Calculations Enabled
	

Calculation Keys work like any other key: they may be dragged onto the report, and utilized in [Keychain Expressions](#).

Dynamic Data Keys



INDUCTIVE
UNIVERSITY

Use Dynamic Data Key

[Watch the video](#)

Normally, Data Keys may only be used to display the value of a key, such as the Text property on a [Text Shape](#) component. However, they can not be used in the same manner to modify other properties on a report component. Instead, you can utilize Dynamic Data Keys.

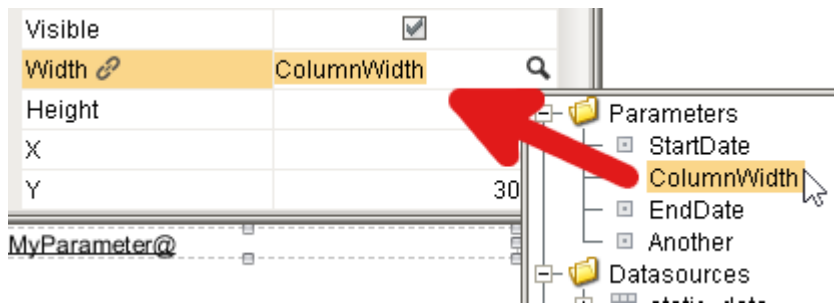
Dynamic Data Keys allow you to use the value of a Data Key on a non-string property. With Dynamic Data Keys, you can modify properties on report components, such as the background color or width, based on the value of a key. This is very similar to the binding system used by components in the [Vision Module](#).

Configuring a Dynamic Data Key

There are two ways to configure a dynamic data key. Note that the syntax of keys differs in Dynamic Data Keys: the "@" are omitted, as demonstrated below.

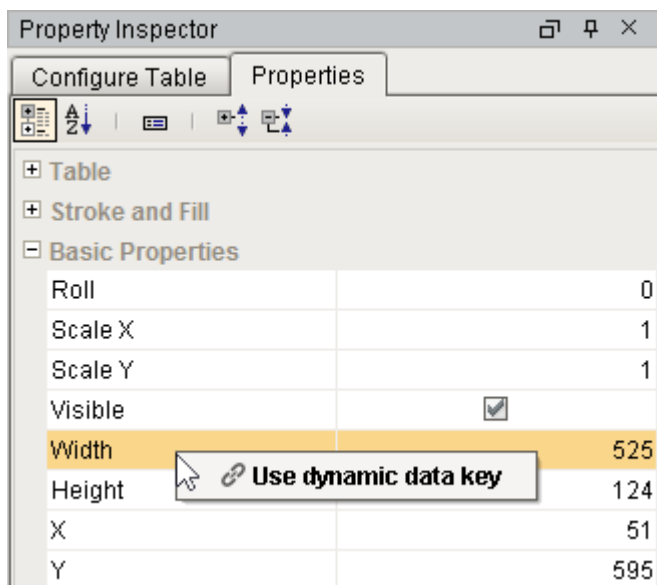
Drag-and-Drop

The easiest approach is by simply dragging a data key from the **Key Browser** directly to a property on a report component.

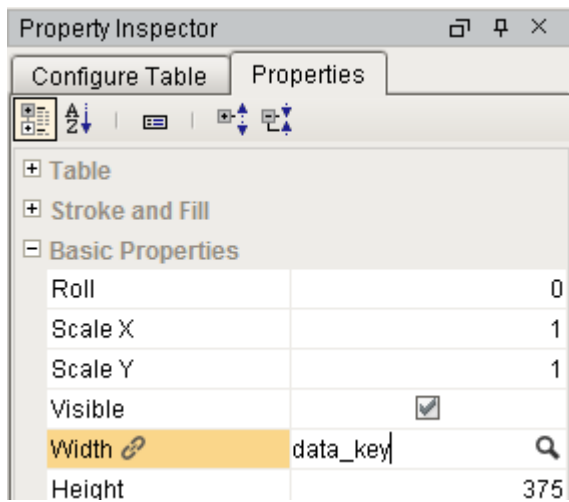


Right-Click

1. With a report component selected, look in the **Property Inspector**, and left click on the name of a property you wish to place the dynamic data key on.
2. Once selected, you can right-click on the property name and a menu will appear.
3. Click on the **Use dynamic data key** menu item.



4. This will place a dynamic data key on the property. An icon of a link (**link icon**) will appear next to the property name, and a default key will be applied to the property.
5. Next we will want to override the default value with one of the keys from the Key Browser. Simply left click on the value field and a magnifying glass (**magnifying glass icon**) icon will appear.



- Click on the ([magnifying glass icon](#)) icon. From here a popup of available keys will appear. Select the key you wish to use, and then click OK.

Data Key Usage

Data Keys as Paths

Data Keys are *relative*, and use 'dot notation' to reference children. Meaning, if we have a nested data structure, we can use Data Key paths (also known as *Keychains*) to reference the nested data. In the key browser image below, we have a nested data source called *Downtime*. *Downtime* contains a number of columns, and then contains a reference to additional data called *runInfo*. If we wanted to access the highlighted operator data, we could use the keychain dot notation in the Designer - **@Downtime.runInfo.operator@**. Nested data sources are outside the scope of this page, but you can learn about Data source nesting in the [Nested Queries](#) section.

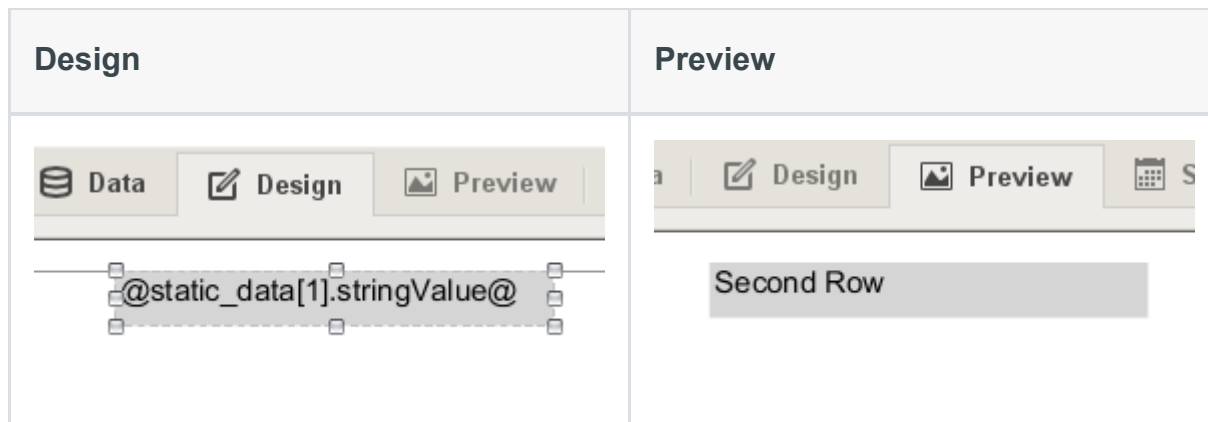
Array Index of Data

You can reference an individual object in a list using standard array indexing syntax (brackets) like this: **@dataSource[0].columnName@**, where "dataSource" is a data source that contains a child data key named **columnName**. Assuming a data source with the values listed below, we can retrieve the value of "Second Row" by specifying index **1** and the column stringValue:

@static_data[1].stringValue@

static-data Example

```
indexColumn, stringValue
0, "First Row"
1, "Second Row"
2, "Third Row"
4, "Fourth Row"
```



Colors in Expressions

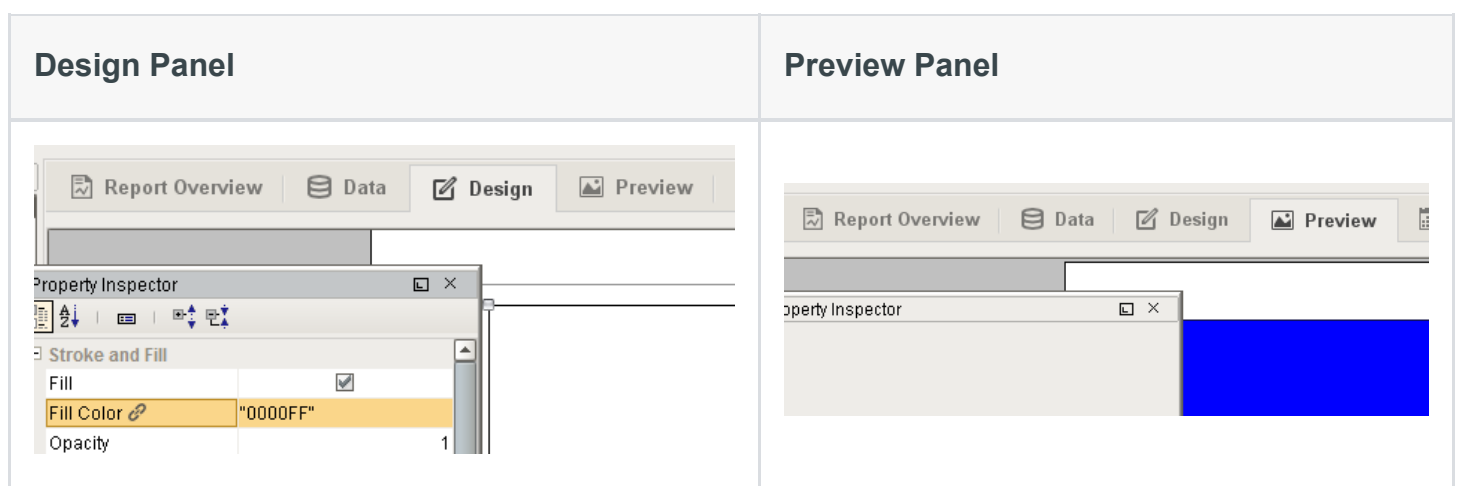
Colors may be references in Keychain Expressions in several ways.

Colors in Hexadecimal

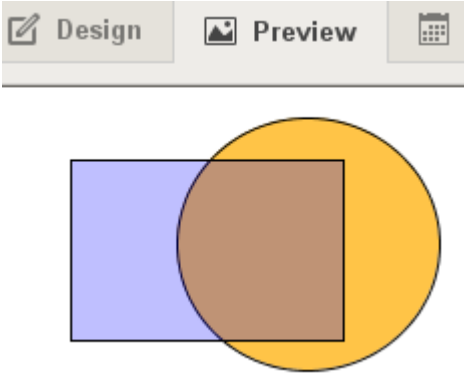
First, hexadecimal case-insensitive color codes may be used. The code must be wrapped in quotation marks to be evaluated correctly. Note that the color change will only appear when the report is executed. The easiest way to test the expression is to switch to the Preview Panel.

Below we see a Blue hexadecimal code of **"0000FF"** is used on the Fill Color of a Rectangle. The Fill Color on the Rectangle was originally set to White. Because the expression will not evaluate in the Design Panel, the Rectangle will appear as a White color.

However, switching over to the Design Panel will generate the report, and evaluate the expression. This in turn returns a Blue Fill Color.

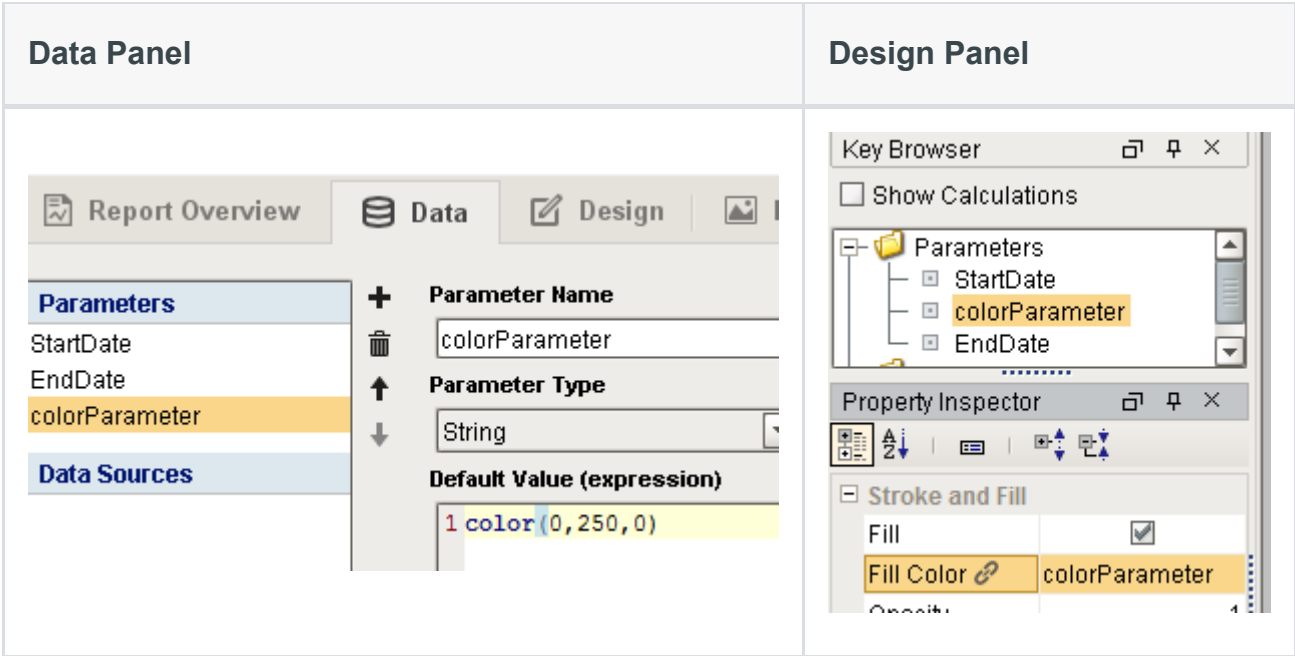


The 7th and 8th digits may be added to specify an alpha channel, or the opacity of the color: **00** is fully transparent, while **FF** is fully opaque. Below, we see a similar rectangle overlapping an ellipse, but with a code of **"0000FF40"**. This represents ~25% opacity, so objects behind the rectangle will be visible, and the fill color will only be slightly opaque.



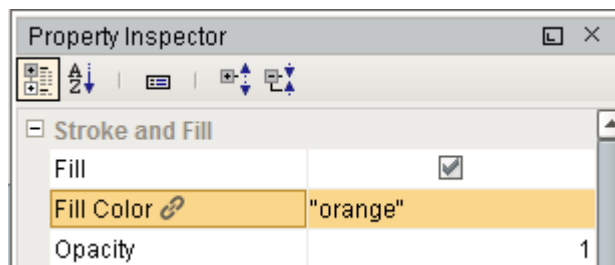
Parameters as Colors

You may also leverage Report Parameters to specify colors. This typically involves creating a parameter with a string datatype, and using the **color** expression function. Once created, you can simply create a dynamic data key reference on the property by dragging the parameter from the **Key Browser** onto the property in the **Property Inspector**. This way, you can have several components use the same color, and modify the color in a single location.



Strings as Colors

Additionally, case-insensitive string color names may be used to return a color. Again, the value must be wrapped in quotation marks.



String Value Color Reference

The following string values may be used:

Value	Example	Value	Example	Value	Example
"beige"	#F5F5DC	"gold"	#FFD700	"lavender"	#E6E6FA
"black"	black	"goldenRod"	#DAA520	"lightGray"	lightGray
"blue"	blue	"gray"	gray	"lime"	#00FF00
"brown"	#A52A2A	"green"	green	"magenta"	magenta
"crimson"	#DC143C	"hotPink"	#FF69B4	"maroon"	#800000
"cyan"	cyan	"indigo"	#4B0082	"navy"	#000080
"darkGray"	darkGray	"ivory"	#FFFFFF0	"olive"	#808000
"fuchsia"	#FF00FF	"khaki"	#F0E68C	"orange"	orange

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Keychain Expressions



INDUCTIVE
UNIVERSITY

Key Calculations

[Watch the video](#)

It's possible to perform calculations on data keys. This section documents the various operators and functions that are available.

Keychain Expressions

Keychains have their own expression language that is largely similar in syntax to Java. To separate this language from others in Ignition, we refer to it as a Keychain Expression.

Keychain Expressions are configured by simply utilizing any operators or functions within the "@" characters. Assuming a key named **"myKey"** with a value of 100, we can multiple its value by 10 with the following expression:

```
//Expression
```

```
@myKey * 10@
```

```
//Output
```

```
100
```

Again, note that the "*" operator and multiplier are enclosed in the "@" characters. Of course, we can also use a different key as the multiplier. Assuming we have another key named **"myMultiplier"** that has a value of 5:

```
//Expression
@myKey * myMultiplier@
```

```
//Output
50
```

Any characters outside of the "@" characters are not part of the expression, so you can easily add prefixes and suffixes with static text.

```
//Expression
Total: @myKey * myMultiplier@ gal
```

```
//Output
Total: 50 gal
```

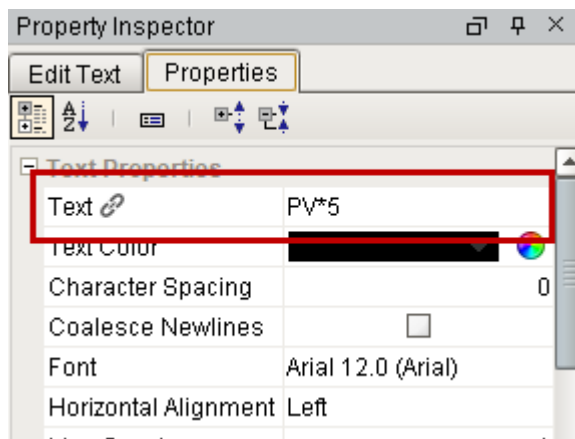
Additionally, you can utilize separate Keychain Expressions in the same TextShape. Note that "**myUnits**" (which has a string value of "**gal**") is enclosed in a separate set of "@" characters, since it is a separate expression:

```
//Expression
Total: @myKey * myMultiplier@ @myUnits@
```

```
//Output
Total: 50 gal
```

Dynamic Data Key Expressions

Keychain Expressions may also be used with Dynamic Data Keys. The syntax and operators work exactly the same, except there is no need to type the "@" characters. Because of this, the entire field is treated as a single keychain expression.



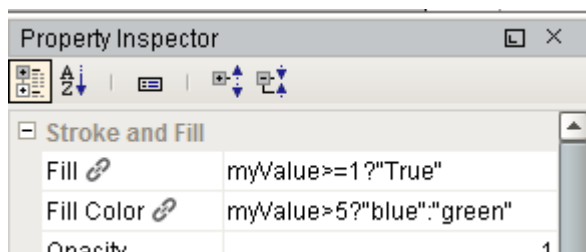
Conditional Keychain Example

The following example demonstrates an if-statement using a Dynamic Data Key. This allows us to highlight different values or ranges contextually, making important values stand out.

Assuming a key named "myValue" has been created, and contains a numerical value, we can use the following syntax:

```
#If the value of the "myValue" key is greater than 5, a blue color will be
returned. Otherwise, a green color will be used.
@myValue>5?"blue":"green@"
```

In the image below, the Fill property is also using a dynamic data key, so the Fill Color will be disabled if myValue is less than 1, Blue if myValue is between 1 and 4, and Green if greater or equal to 5. Note that in the Property inspector, the @ symbols are not needed



To add more color-value pairs, we simply add more if statements to the end of the expression with a colon:

```
#If the value of "myValue" will determine one of multiple colors:
#Greater than 10 will return Red
#greater than 5 (but not greater than 10) will return Blue
#anything else will return Green.
@myValue>10?"red":myValue>5?"blue":"green@"
```


Operators and Functions

Operators

The following operators may be used in a Keychain expression.

Operator	Function	Example
Parenthesis	(expr) Nested Expressions	Any portion of a Keychain can be enclosed with parenthesis to guarantee precedence.
Multiplicative	*, /, % Multiply, divide, modulo	These are the most common and intuitive operators. You might want to display <i>@quantityprice@ in an invoice line-item or calculate a percent like this @profit/revenue100@.</i>
Additive	+, - Add, subtract	See multiplicative above
Relational	>, <, >=, <= Greater-than, less-than, greater/less-than-equal	These are most useful for conditionals: <i>@amount>=0? "Credit" : "Debit"@</i> or <i>@name=="this"? "that" : name@</i>
Equality	==, != Equal, not-equal	See Relational above
Logical	AND &&	These operators make it possible to test multiple conditions: <i>@revenue>100 && budget<50? "Winner!"@</i> or <i>@name=="Jack" &#124&#124 name=="Sam"? "Good Name!"@.</i>
Logical	OR &#124&#124	See AND above
Conditional	? : If/then - with form "expr? true_expr : false_expr"	Provides IF/THEN/ELSE expressions. Note: a false expression is optional. 'null' will be evaluated to false and non-null as true. You can provide null substitutions like this: <i>@name? name : "(None provided)"@.</i> You can also nest conditionals for more conditions. For example, <i>@age>=21?"Adult":(age>12?"Teen":"Child")@.</i>
Assignments	=, +=	For the brave, you can create temporary variables for use in a report. Most of the functionality you might use this for is covered in more intuitive ways (such as the

Operator	Function	Example
		Running key), but it is possible to define a variable in a header row: @revTotal=0@ and update it in details rows @revTotal+=revenue@.

Math Functions

The following functions return floats.

Menu Item	Function
floor(float)	Round input down to the nearest whole number.
ceil(float)	Round input up to the nearest whole number.
round(float)	Round input to the nearest whole number.
abs(float)	Returns the absolute value of the input (if number < 0 return number * -1).
min(float, float)	Returns the input number with the least value.
max(float, float)	Returns the input number with the greatest value.
pow(float, float)	Returns first number to the second number power.

String Functions

The following functions return strings.

Menu Item	Function
startsWith(String, String)	Returns true if the first string starts with the second.
endsWith(String, String)	Returns true if the first string ends with the second.
substring(String, int start)	Returns a substring of String beginning at position start.
join(List aList, String aKeyChain, String aDelimiter)	Used to display an individual attribute of individual objects as a single String. Suppose you have a list of movies and want to show their titles in a comma separated list: @join(getMovies, "getTitle", ", ")@
substring(Object aString, int start, int end)	Obtain a subset of a given string. This could be useful if you wanted to restrict a text field to a certain number of chars:@substring(title, 0, 10)@

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

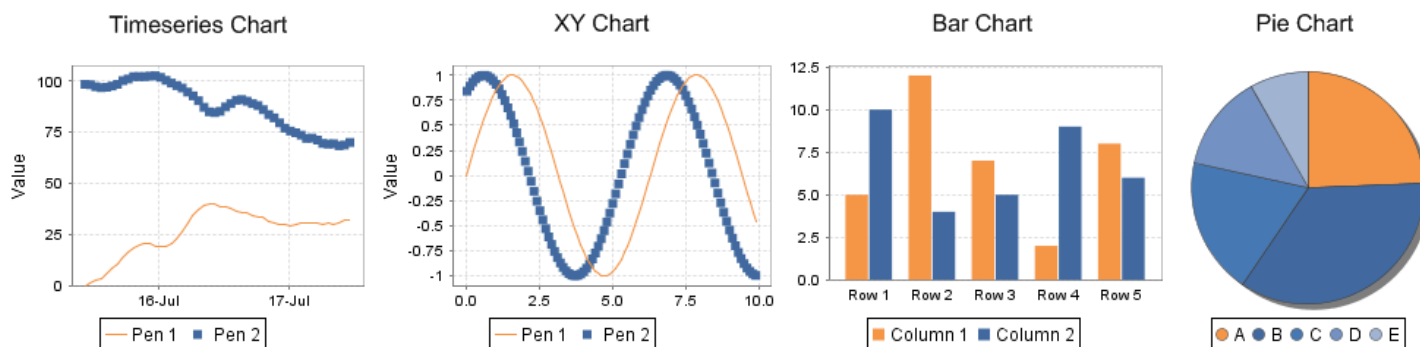
For documentation on the latest stable release, see the [latest version](#) (8.3).

Report Charts

Report Charts

Report charts allow you to display your data in a graphical way, just like the charts in the rest of Ignition. Charts can be driven by any **data source** in a report, or even **embedded into table rows** using nested data sets. There are four types of charts so you can show the data any way you like.

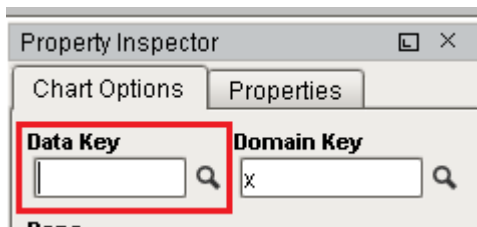
- **Timeseries Chart**: a simple chart that plots values against a timestamp X axis. Great for showing historical trending.
- **XY Chart**: similar to the Timeseries chart, this chart plots lines on X-Y axes, but is configured to show numeric categories on the X axis.
- **Bar Chart**: a bar chart that uses text categories on the X axis. Can be configured as a Pareto chart.
- **Pie Chart**: a basic pie chart that uses text categories. Percentages are automatically calculated.



The Data Key Property

Similar to Tables in reporting, all charts must **first** be assigned a Data Key. This assignment configures the chart to look at specific keys in a data source, and prevents any name collisions with other data keys: if multiple data sources in your report return a column named "id", the chart wouldn't know which column you were referring to without this initial assignment.

Assignment can be achieved by drag-and-dropping a key into the Data Key property, or by clicking the Key Search 🔍 button.



Once assigned, you can the start adding pens or bars to your chart.

Timeseries Chart

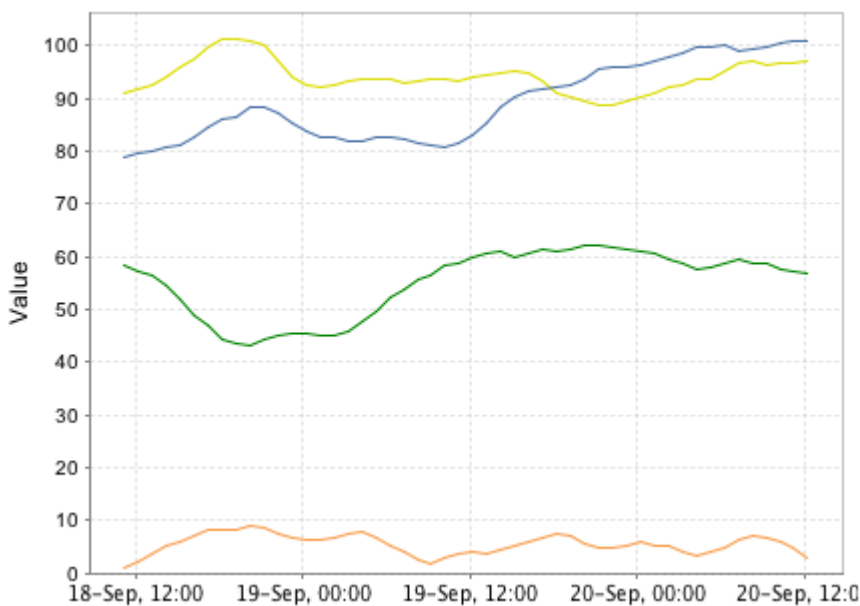


INDUCTIVE
UNIVERSITY

Timeseries Chart

[Watch the video](#)

The **Timeseries Chart**  is a type of XY chart whose domain or X-Axis represents time series data and range can be one or more pens. The Timeseries Chart is a great way to display data visually from Tag History, or similar time related data sources.



Usage

To use the chart, drag the component from the Report Palette to your report. Type or drag a data key from the Key Browser into the **Data Key** field of the Chart Options tab in the Property Inspector. Select a time series domain (for example a `t_series` column of your query). The Y axis (value) can be modified in the Chart Options tab, but the X axis (time range) is based on the data in the 'Data Key' that powers the chart dataset.

To add pens to your query, simply click the  button on the Chart Options tab next to the Pens table in the Property Inspector. You can either double click a pen, or select a pen and click the  button on the right side, to navigate to the Pen Configuration area. Click the  to return to the Chart Options tab.

Chart Options area

Property Inspector

Chart Options

Properties

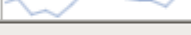
Data Key

history

Domain Key

t_stamp

Pens

Range Key	Pen Name	Preview	
ambienttemp	Temperature		+  
ambienthum	Humidity		 
accumulatorle...	Accm. Level		 
receiverLevel	Rec. Level		 
key	New Pen		+  

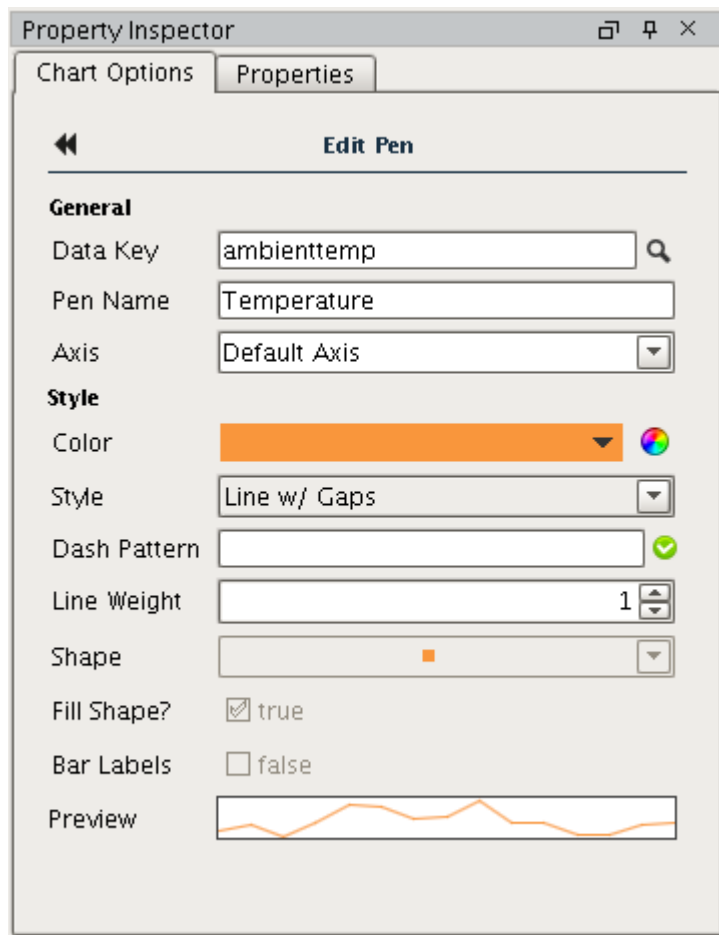
Axes

Name	Label	Auto?	Low	High	
Default Axis	Value	☒	0	100	+  

Scripting

☐ Enabled  Edit Script

Pen Configuration area



XY Chart

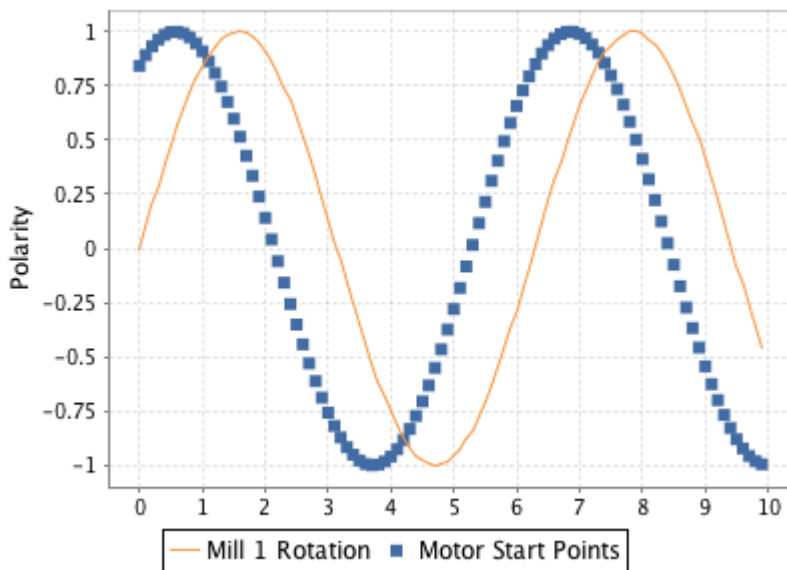


INDUCTIVE
UNIVERSITY

XY Chart

[Watch the video](#)

The **XY Chart**  generates an X vs Y plot of your data. XY Charts can have multiple pens and axes per data source, and each is easily configurable in the Chart Options tab for the component.



Usage

To use the chart, drag the component from the Report Palette to your report. Type or drag a data key from the Key Browser into the **Data Key** field of the Chart Options tab. Select Domain Key to use for the X axis.

To add pens to your query, simply click the  button on the Chart Options tab next to the Pens table in the Property Inspector. You can either double click, or select a pen and click the  button on the right, to navigate to the Pen Configuration area. Click the  to return to the Chart Options tab. The setup and Configuration of this chart and its pens works similarly to the Timeseries Chart, the difference being that instead of using a date datatype for the domain, a different datatype is needed.

Bar Chart

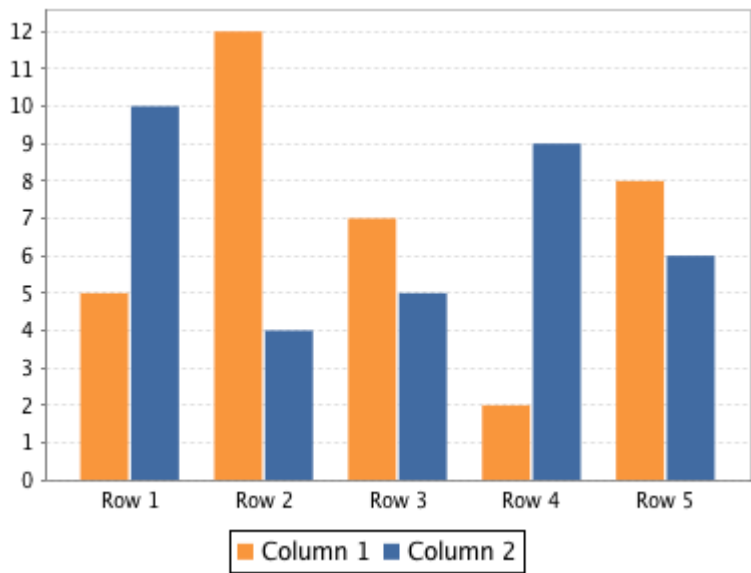


INDUCTIVE
UNIVERSITY

Bar Chart

[Watch the video](#)

The **Bar Chart**  Bar Chart component can be used to add bar charts to a report. It uses text categories on the X axis, and can also be configured as a Pareto chart.



Usage

To create a Bar Chart, simply drag the component from the Report Palette and drop it onto your report. Bar Charts are quite easy to use and have a large number of customization options. Configuring a Bar Chart requires a data source whose first column generally contains the name or identifier the bar represents, and the following one or more columns represent some numerical value to be plotted. A Bar Chart example can be found in the [Report Workflow Tutorial](#).

Property Inspector

Configure

Properties

Data Key

Extract Order

☒ Row

☐ Column

Options

☐ 3D Bars

☐ Pareto

Render Style

☒ Bars

☐ Stacked

☐ Layered

Segment Colors

Scripting

☐ Enabled

☒ Edit Script

Pie Chart

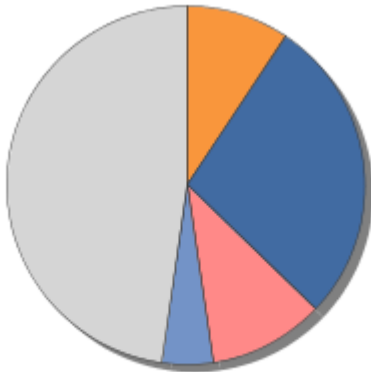


INDUCTIVE
UNIVERSITY

Pie Chart

[Watch the video](#)

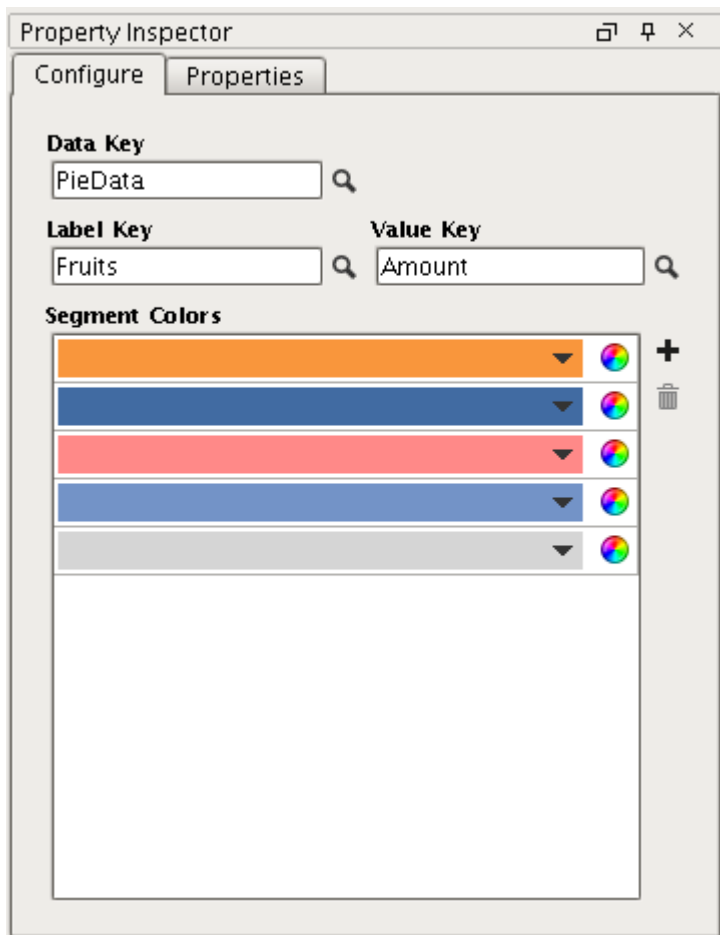
The **Pie Chart**  **Pie Chart** adds a Pie Chart based on a simple two column data structure.



 Bananas  Grapes  Apples  Pineapples  Other

Usage

To create a new Pie Chart, drag the pie chart palette item from the Report Design Palette onto your Report Page or Parent Shape. A pie chart has a simple configuration consisting of label values (generally Strings) and numeric quantity values. Color segments can be added or removed by clicking the  button on the Configure tab.

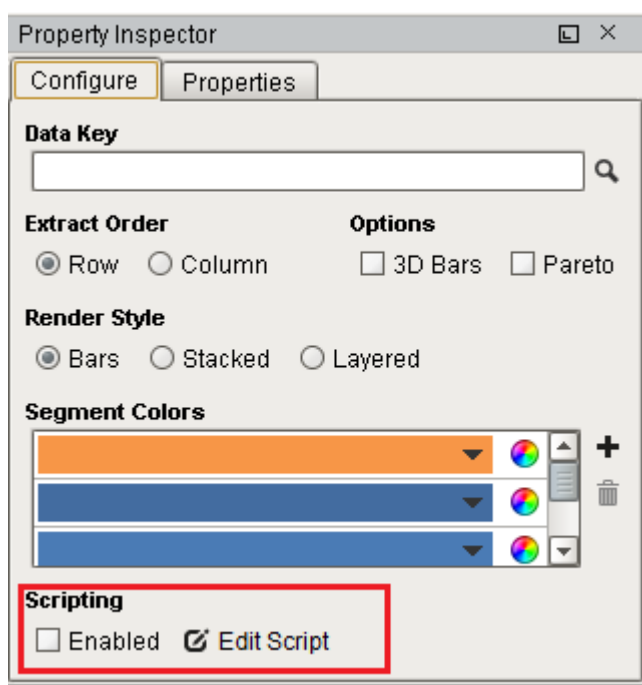


The Pie Chart in the images above was created using a simple data source which had just two columns, the first which represented our Label, and the second being a column of values.

```
Fruits, Amount
"Bananas", 52
"Grapes", 154
"Apples", 58
"Pineapples", 25
"Other", 265
```

Chart Scripting

The look and feel of the Timeseries, XY, and Bar charts may be modified through scripting. Scripting may be accessed on these charts by selecting them, and then clicking the **Edit Script** icon in the Property Inspector. Note that the **Enabled** property must be checked for the script to become active.



Clicking on **Edit Script** will provide access to **configureChart**, which allows you to make modifications to the chart right before the report is rendered. The charts are simply JFreeCharts, so [reading through the JFreeChart API](#) would be useful here.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Getting Started with Report Charts

Report charts have a ton of functions and features that it's hard to know where to begin. You'll be amazed how simple a standard chart is to create. To start with, there are several chart types to select from, but the chart type you choose depends on the type of data you have and how that data might best be graphically displayed. Next, charts are driven by specific data keys in a data source. This page provides a quick start to getting you well on your way to creating report charts that are easy to configure and look stunning in your reports. Let's get started!

Creating a Report Chart

One of the most common charts used for historical trending is a **Timeseries Chart**. The Timeseries Chart plots values against a timestamp X axis.

Adding a chart to a report involves a couple of steps. In order, they are:

1. The chart needs data, so we must create a data source.
2. Once a data source exists, we can create the chart and assign the data source.
3. Apply any additional chart configuration, such as adding an additional axis.

Now, let's create a Timeseries Chart using Tag History data.

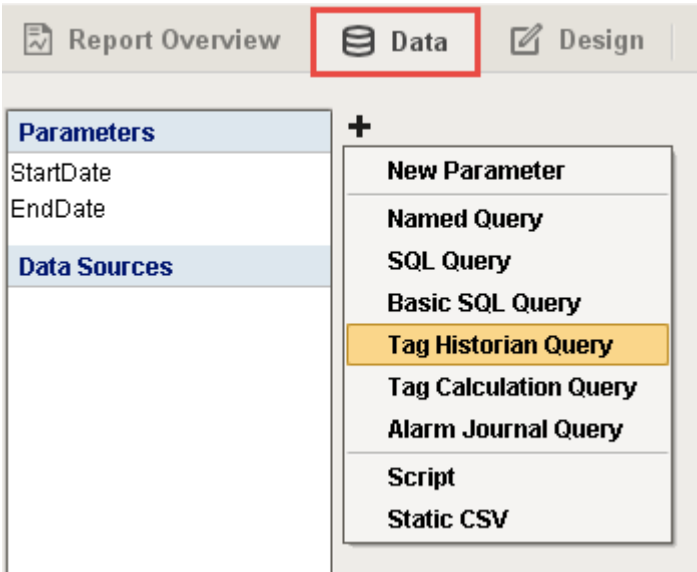
⚠ BEFORE YOU START THIS EXAMPLE...

Ignition comes with one internal Tag Provider so you can get started right away, but first you need to connect to a device. The [Simulator device driver](#) is a great starting point if your Gateway is not currently connected to a PLC. This example assumes you are using the Generic Simulator device tags, but any tags could be used instead. This example assumes that the simulator device connection was named '**gensim**', and [Sine tags were created](#).

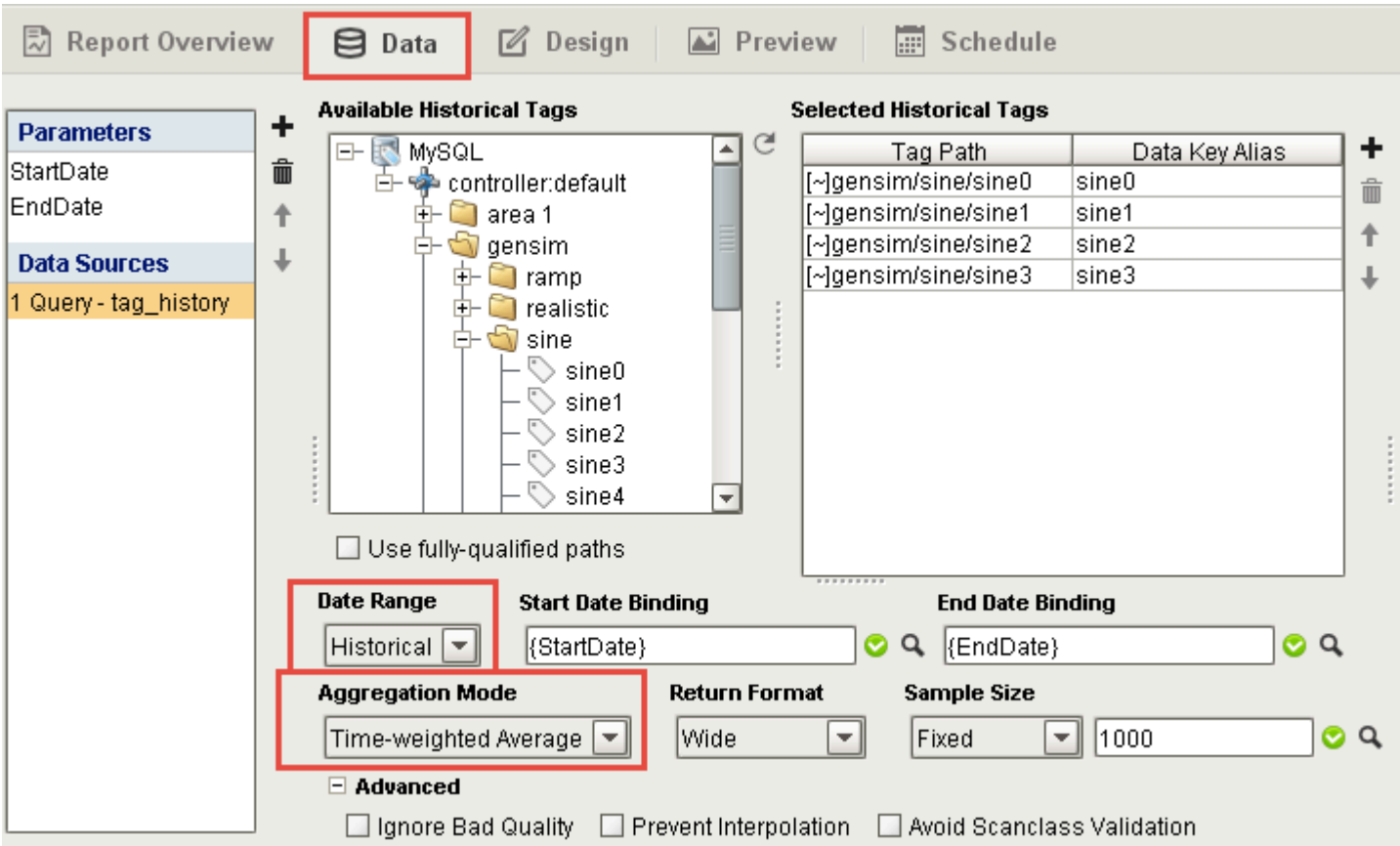
You also need a [database connection](#) that the historian can use to store Historical Tag data. This example assumes [Tag History has been enabled](#) on the **Sine** tags.

Creating a Data Source

1. In the **Data** tab, under Parameters and Data Sources, click the plus **+** button to add a data source. Select **Tag Historian Query**.

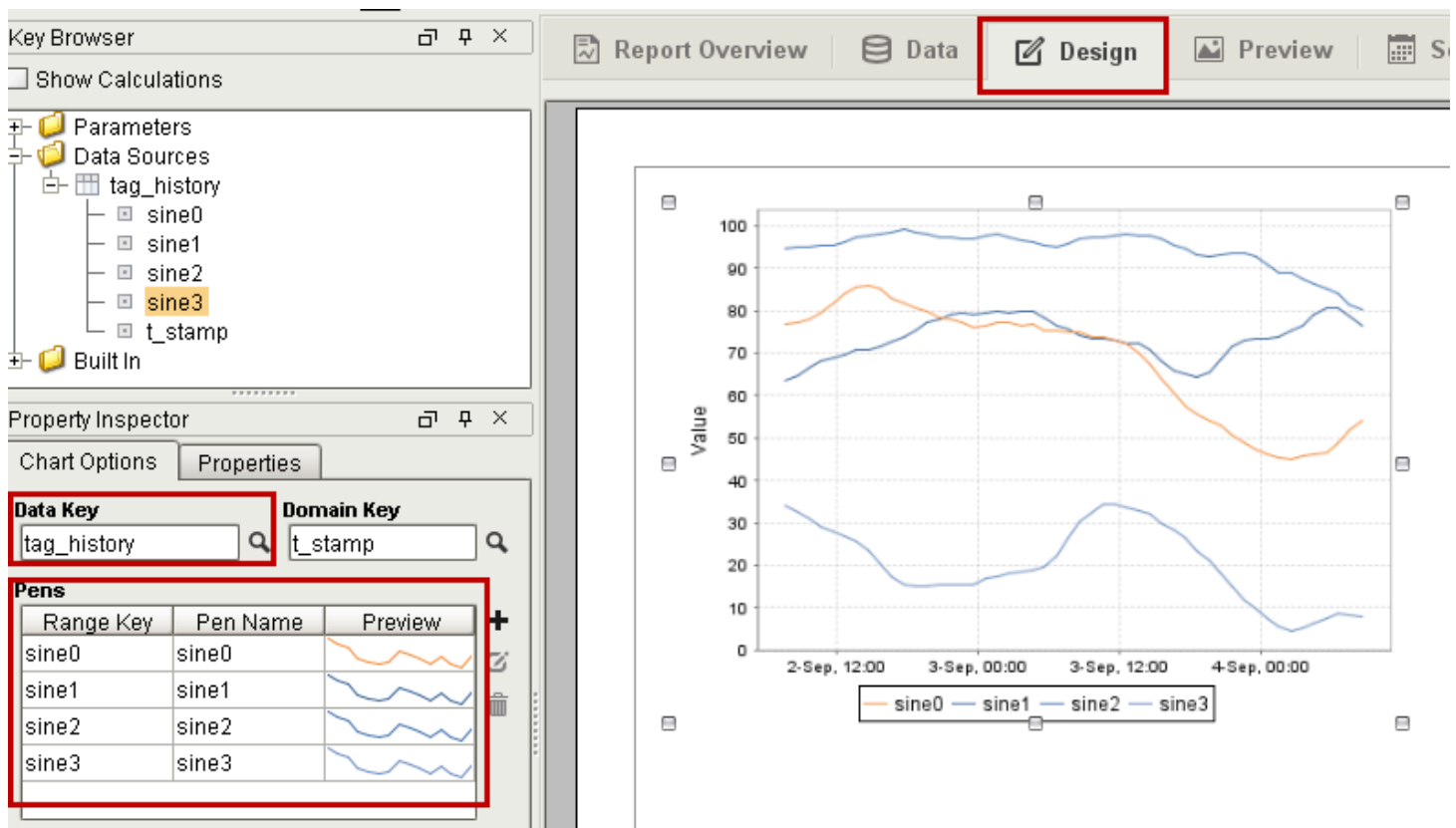


2. Under the **Available Historical Tags** column, open the folder structure until you locate your Tags. As mentioned above, we are using the device connection named '**gensim**', so we will select the '**gensim**' folder in this example. Here you'll find a bunch of Tags you can use immediately. Expand the **Sine** folder.
3. Select a few of the Tags (i.e. sine0, sine1, sine2, sine3) and drag them to the **Selected Historical Tags** column under **Tag Path**. Take a look at the properties down at the bottom. Set the **Data Range** property to **Historical**, and the **Aggregation Mode** to **Time-weighted Average**.

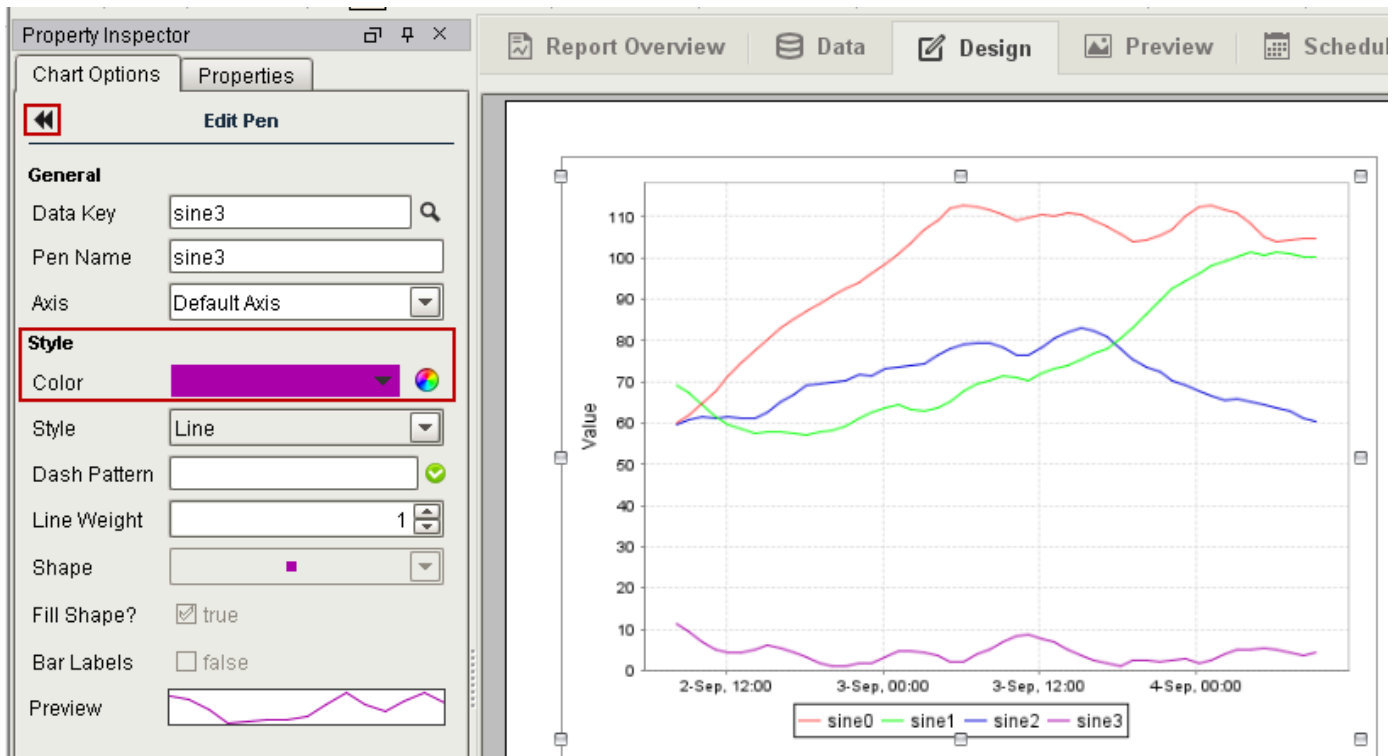


Adding a Chart

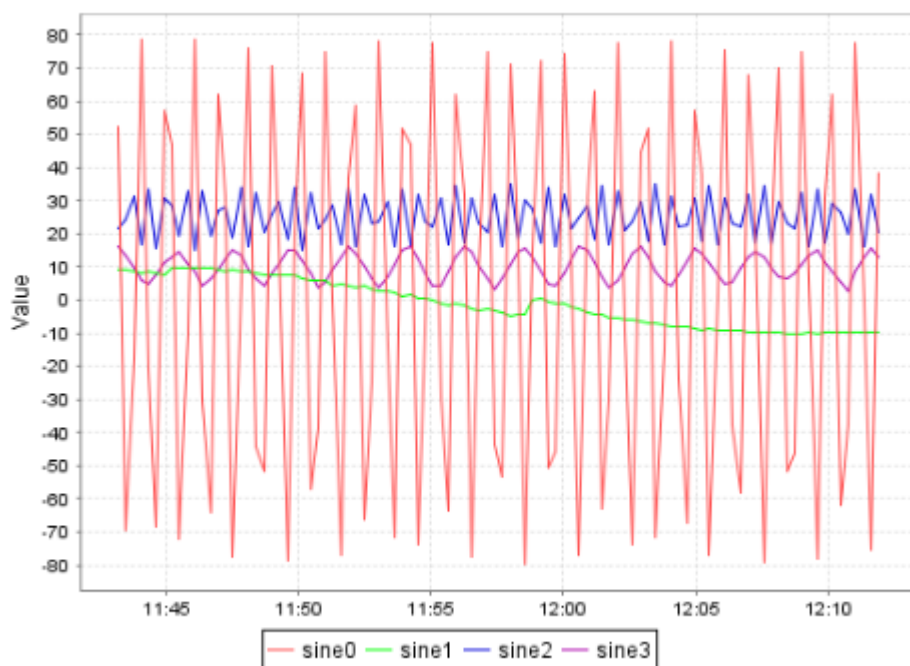
1. Now switch over to the **Design** panel. Drag a **Timeseries chart** from the Report Design Palette into the report and expand it.
2. From the **Key Browser**, expand the **Datasources** folder, and drag your datasource (i.e., tag_history) down to the **Data Key** under the **Chart Options** tab of the Property Inspector.
3. Select **Test1** and **Test2** pens and delete them by selecting each pen individually, and then clicking the Trash Can icon .
4. Once the Data Key is correct, you can assign pens to your chart. Drag in your Keys (i.e., sine0, sine1, sine3, sine4) to the Pens property table. You'll immediately notice your chart is populated with fake data, this is just a preview and does not represent the data returned by your data source.



5. You can modify a selected pen by clicking the Edit  button or double clicking the pen to open the **Edit Pen** tab. Here you can change the name of the pen, change the pen style, color, and more. Let's change the color of the pens to make them really stand out.
 - Lets start with the sine0 pen.
 - Set the **Color** to be red.
 - To return to the Chart Options tab, click the double arrow  button.
 - Repeat the above steps for the remaining three pens, making sine1 green, sine2 dark blue, and sine3 purple.



6. Go to the **Preview** tab to view the Timeseries Chart in the report.

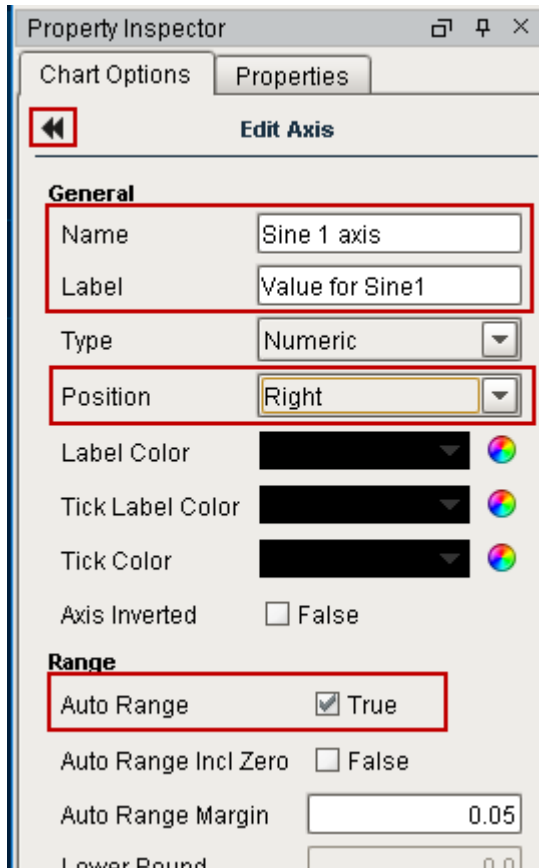


Adding an Axis

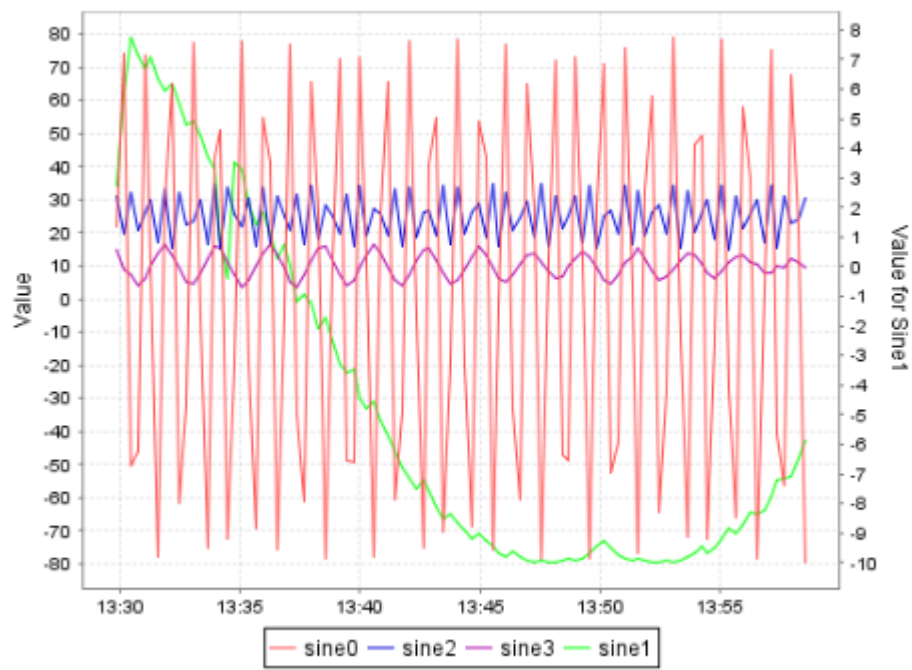
Unfortunately, it is a little difficult to see the small changes in our sine1 pen because its range is much smaller than the other pens. Let's fix this by adding a new axis.

1. Go back to the Design tab and select the chart. Click the Plus **+** button next to the Axes table to add a new axis, then click the Edit **E** button below it to edit our new Axis.

- Give it a name of **Sine 1 axis**
- Give it a Label of **Value for Sine1**
- Change the position to right so that it shows up on the right side of the chart.
- We can leave Auto Range turned on, so that it will automatically pick the appropriate min and max for the sine1 pen.
- Return to the Chart Options tab by clicking the double arrow  button.



2. We now need to tell our sine1 pen to use that new axis. Select the sine1 pen from the pens list and click the Edit  button. Change the Axis of the pen to be the pen we just created.
3. Navigate to the Preview tab to view our finished chart.



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Report Table

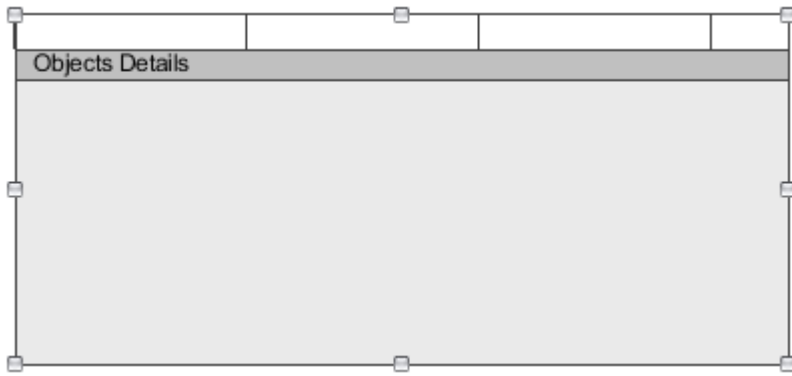
Tables are a major part of Ignition Reporting. Tables are objects that display data in a structured, repetitious format giving users the ability to flexibly layout and organize tabular data in a variety of ways. Their complexity can range from trivially simple to complicated, but fundamentally they do the same thing - take the data given to them and repeat it in a structured way for each row of the data provided. The Reporting engine will automatically create new pages to fit all data within the Table's boundaries while applying formatting preferences each step along the way such as font, size, layout and alignment. Combine that feature with powerful data manipulation and expressive layout tools, and you get an object that often forms the basis of your reports.

Dec 20, 2005 17:55	labeler	50 minutes	Out of labels
Dec 22, 2005 11:55	filler	15 minutes	Scheduled maintenance
Jan 02, 2006 22:55	palletizer	10 minutes	Misalignment
Jan 03, 2006 02:55	conveyor line	25 minutes	backup
Feb 12, 2006 06:13	labeler	10 minutes	<NA>
Feb 12, 2006 12:01	labeler	3 minutes	Out of labels
Feb 12, 2006 14:01	palletizer	17 minutes	Misalignment
Feb 12, 2006 16:23	conveyor line	23 minutes	Scheduled maintenance
Feb 12, 2006 20:04	filler	33 minutes	Overflow
Feb 12, 2006 20:13	labeler	21 minutes	Stuck labels
Feb 12, 2006 20:25	filler	20 minutes	Overflow
Feb 12, 2006 20:36	conveyor line	30 minutes	Scheduled maintenance

Let's start by looking at the various parts of the Table component and move into more features.

The Table Component

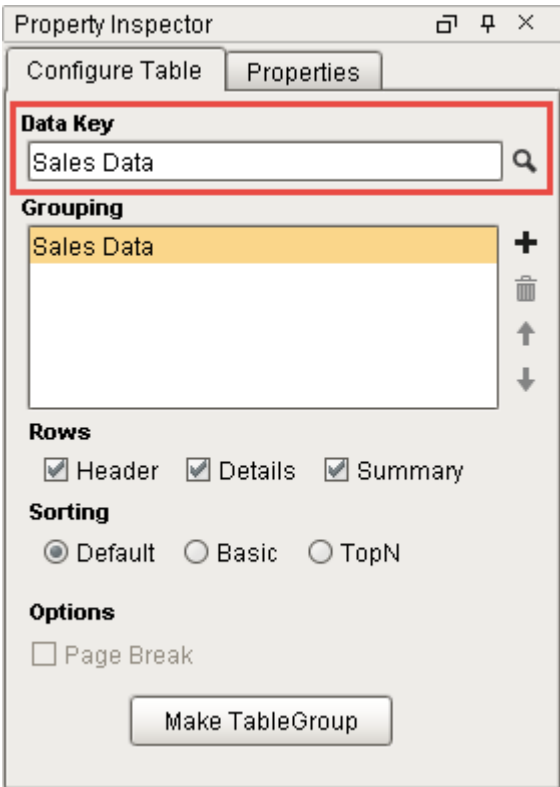
The Table  Table component can be created by dragging it from the Report Design Palette, or by simply dragging a Data Source from the Key Browser to a page in your report. A new Table with no bound Data Key is fairly simple looking, but hides a wealth of functionality. When you create a table, by default, you will see a basic table like the image below on the report page. It says "Object Details" because it has not yet been set up to use a Data Source.



Once you have your dataset, you're probably thinking about how to display that data in a report. To best understand how to use a table to illustrate your data in a report, you need to have a basic understanding of how the report table works.

Table Configuration

When you first drag a table into the Design panel of your report, the **Configure Table** tab will appear in the Property Inspector. You will see that it has a default **Data Key** of "Objects", which matches the **Details** row in the table. The first step when configuring your table is to drag your **Datasource** from the Key Browser to the Data Key field in the Configure Table tab. Changing the Data Key will also change the label on the Details row making it easier to determine what you are looking at when editing multiple tables or **Groupings**. Depending on how you want to present your data, you may also want to add Header and Summary Rows.



Once you add a Data Key to your table, you can start designing. There are many tools built into the Table interface that you can use to help you layout and organize the data in your table. Now all you

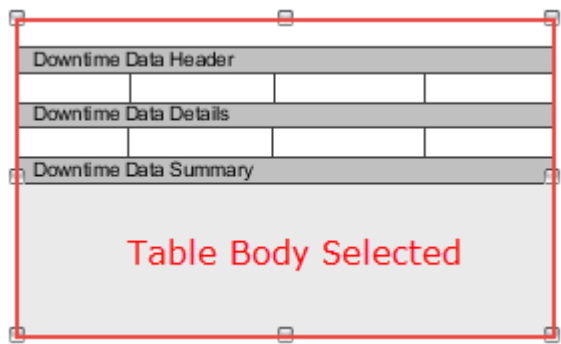
need to do is decide how you want to organize and display the data in your report.

If you have a lot of tabular data, you probably want to use a Standard row. You might start by simply dragging your Data Keys from the Key Browser to columns in your Details row. Then, you might give each Header column a label. If you have a bunch of numbers that you want to add together, find out the 'min', 'max,' you can create a Summary row, enable the **Show Calculations** property, and choose from the list of calculation keys. Lastly, you might want to make some cosmetic changes to the font size, font color, or changing the Header row from **structured to unstructured** so you can add an image, chart, text shapes, etc.

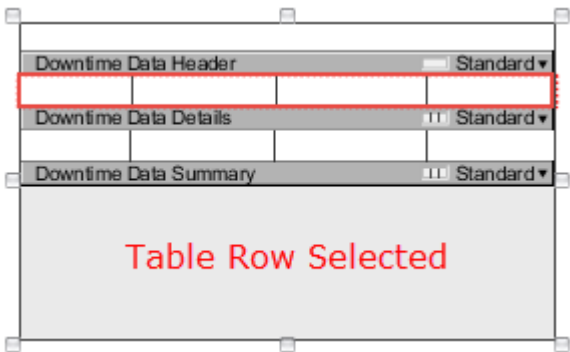
Anatomy of a Table

A table has multiple sections, each with its own properties. By default, a Table consists of three sections:

- **Table Body** - Grey area at the bottom of the table template. You can stretch and shrink the table boundaries to position and size a table.



- **Table Rows** - There are three types of table rows: Header, Details, and Summary. A table row can also be structured or unstructured. More information on these row types can be found on the Table Rows page.



- **Text Shapes / Cells** - are available only in a structured table row. In Reporting, Text Shapes are commonly referred to as cells. Can select multiple cells with alt or shift.

Downtime Data Header		Standard ▼
Downtime Data Details		Standard ▼
Downtime Data Summary		Standard ▼
Text Shape / Cell Selected		

❗ TEXT SHAPES OR CELLS

If you're looking in the Project Browser tree, you'll notice a node called 'Text Shape.' In Reporting, the Text Shape is also called a 'cell.' For the sake of clarity, Text Shapes will be referred to as 'cells,' unless we are referring to the Text Shape component.

The following example shows the basic anatomy of a table with data populated in the Header, Details, and Summary rows. Once you're finished designing your report, go to the Preview panel and check your results. You can continue to navigate between the Design and the Preview panels making changes until you get your report just right!

Order Number	Date Shipped	Sales Person	Amount
Sales Data Header			
@Order Number@	@Date Shipped@	@Sales Person@	@Subtotal@
Sales Data Details			
			Amount: \$@total.Subtotal@
Sales Data Summary			

Header Row
Details Row
Summary Row
Row
Cell
Row Structure icon
Row Version
Table Boundaries

Each section (i.e., Table, Rows, and Cells), when selected, has its own unique properties (shown below), along with some common basic properties that appear in the **Properties** tab of the Property Inspector. Refer to the [Table](#) page in the Appendix for a complete list of all Table properties.

Table	
Data Key	Downtime
Column Count	1
Column Spacing	10
Filter Key	
Starting Page Break	☐

Table Row	
Structured Columns	☒
Column Count	4
Move to Bottom	☐
Print Empty Group	☒
Sync Alternate Ver...	☐
Sync Row Above	☐
Version Key	

Text Properties	
Text	Site
Text Color	
Character Spacing	0
Coalesce Newlines	☐
Font	Arial Bold 12.0 (Arial)
Horizontal Alignm...	Left
Line Spacing	1
Margin	1; 2; 0; 2
Overflow Behavior	Grow row
Underlined	☐
Vertical Alignment	Top
Data Key Format Properties	
Date Format	MMM d, yyyy 
Null Format	<N/A>
Number Format	#0.## 
Negative In Red	☐

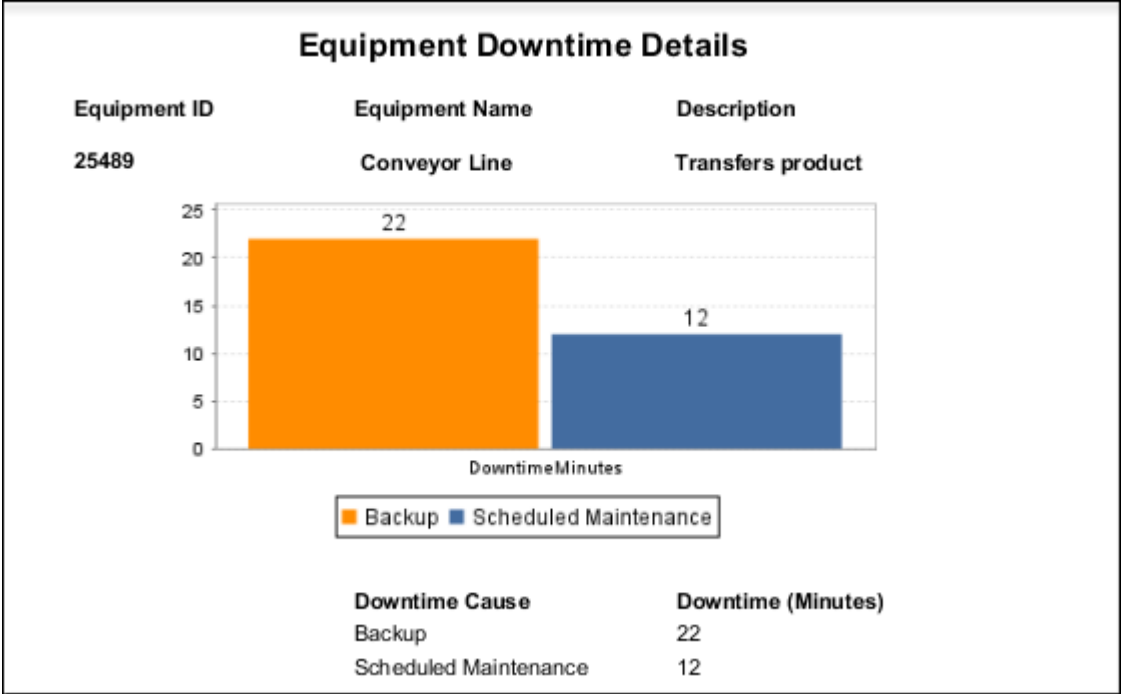
⚠ TABLE COLUMN COUNT VS ROW COLUMN COUNT

When modifying properties on a section of the Table, always pay attention to which section you selected. Some property names on one section may look similar to properties on a different section. For example, Table Rows have a **Column Count** property that adds a new column to the row, but the Table Body section also has a **Column Count** property that will cause the same table to spill out horizontally on the same page. This is great for landscape pages, but on portrait oriented pages it can cause some of the data to be rendered on an unprintable area of the report!

Report Table Features

Tables have a lot of built-in features that give reporting users the ability to create simple tables and increasingly more complicated tables. The table features are quite powerful, not only do they allow you to organize your tabular data, they also provide sophisticated data manipulation and layout tools. There are three types of table rows: **Header, Details, and Summary** rows that make up the body of a table. You can break tables down by data keys that share a common value using Dataset Grouping. The different types of rows can be independently enabled for each level of grouping, each group having its own Header, Details, and Summary rows. Additionally, the keys from '**Show Calculations**' and other **keychain functions** are supported for any level of grouping.

By default, when you initially create a table all the row types are **structured**. A structured row has a variable number of columns and allows you to organize your text based table data, whereby, an unstructured row is highly customizable allowing you to place images, text shapes, and charts within the table row. Unstructured rows are perfect for placing charts anywhere within the row of the table. An unstructured Details row typically works best when used in conjunction with **Grouping Data Inside of Tables** or **Nested Queries** so that each row has a chart with data from that group or query.



Another feature is **Table Row Versioning** which gives you the option of displaying rows with a different format to make them stand out, such as creating an alternate background row color, making the first row different, or even creating your own custom row version. You have have a choice of using any of the Built-in row versioning options or creating a custom version.

Candy Bar Name	Inventory	Cost per Candy	Value of Candy Supply
Pay Day	22	\$ 2.10	\$ 46.20
Baby Ruth	44	\$ 2.40	\$ 105.60
Milky Way	16	\$ 1.90	\$ 30.40
KitKat	28	\$ 1.85	\$ 51.80
Butterfinger	7	\$ 2.30	\$ 16.10
Snickers	19	\$ 2.00	\$ 38.00
Twix	13	\$ 2.05	\$ 26.65

Report Tables also support **Table Grouping** which is an easy way to add multiple Data Sources to an existing table in a report using Peer Tables and Child Tables. Peer Table groups allow the second table to begin exactly where the first table ends. Child Table groups allow you to nest one table inside another. What's really nice about Table Groups and Nested Data Sources is that you can create Summary Tables for categories of items or drill-down charts all in one report.

Once you get familiar with all the report table features, you can easily design professional tables for your reports.

Other Table Components

There are a few **other table components** in the Report Module, Simple Table and CrossTab Table, and they both behave a little differently. Selecting the right table depends on the type of data you have and how you want to display it.

The **Simple Table** is similar to a basic table that dynamically crates new rows and columns for rows returned by the Data Keys on the component. With a Simple Table, you can very quickly add a table inside a report.

The **CrossTab Table** is commonly used to summarize the relationship between two groupings of data by showing summaries of cross sections of the datasource. The CrossTab Table has lots of repetitious data, a datasource that provides at least two columns of data which are repetitious compared to the number of rows, and one or more columns that represent a value that requires a calculation. This table will stretch both its height and width to accommodate the underlying dataset.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Getting Started with the Report Table

The **Report Table** is another Reporting component, like report charts, that has a lot of features and functions that help you create meaningful reports. Tables are a major part of Reporting and simple to create. This page will give you a great head start for creating your own standard report tables using the built-in design tools.

Creating a Report Table

Creating a table is a simple process, but the order of how you create a table is important. The first thing you need before creating your report table is to create a **data source**. Your data source can be a query, script, or CSV file. Next, add a table and configure the data in the table that you want to display. Lastly, you can use the built-in design tools to enhance your reports. You can add images to your reports, change row types, show mathematical calculations, change the font style, size, or color of the text, and much more.

Let's get started!

This example creates a standard chart that tracks the downtime minutes for each production line. We'll configure the data in the table, do a calculation, add an image, and make a few property changes so you get a sampling of the power of report tables.

Adding a Datasource

ⓘ DATASOURCE

This example uses a CSV file as it's datasource. You can use the one shown in the code block below, or create your own.

1. In the **Data** panel, click the plus **+** button to add a datasource, and select **StaticCSV** from the resulting list of options. From the code block below, copy the text and paste it into the **Data** field. This is the data that we will use in this example.

CSV Datasource

Production Line, Downtime

Line A, 75

Line B, 92

Line C, 43

Line D, 54

Line E, 66

Line F, 80

Line G, 40

Line H, 88

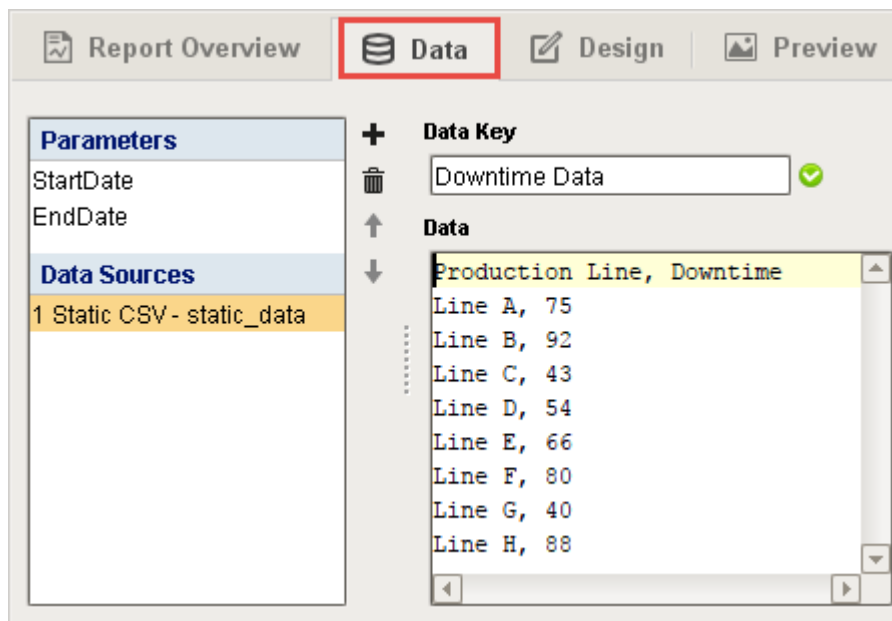
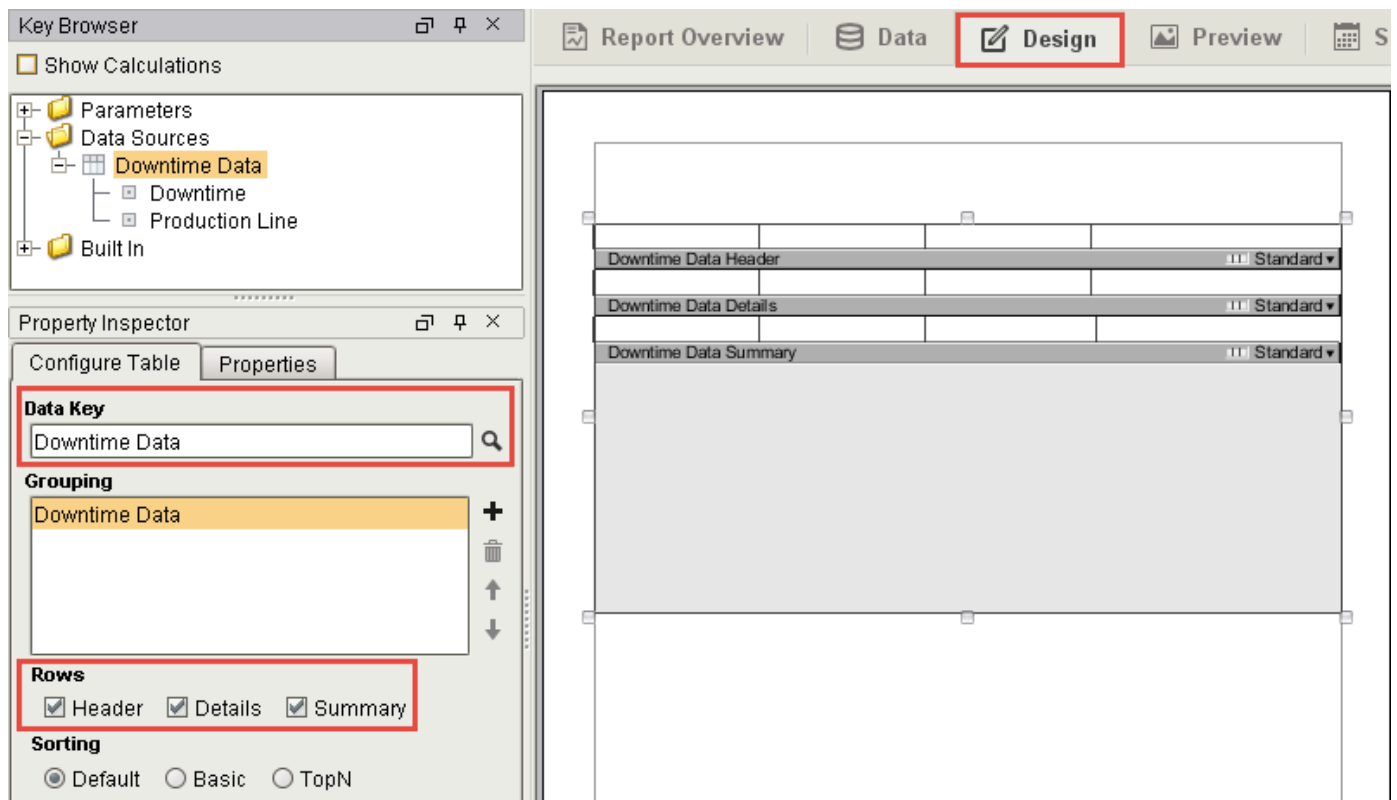
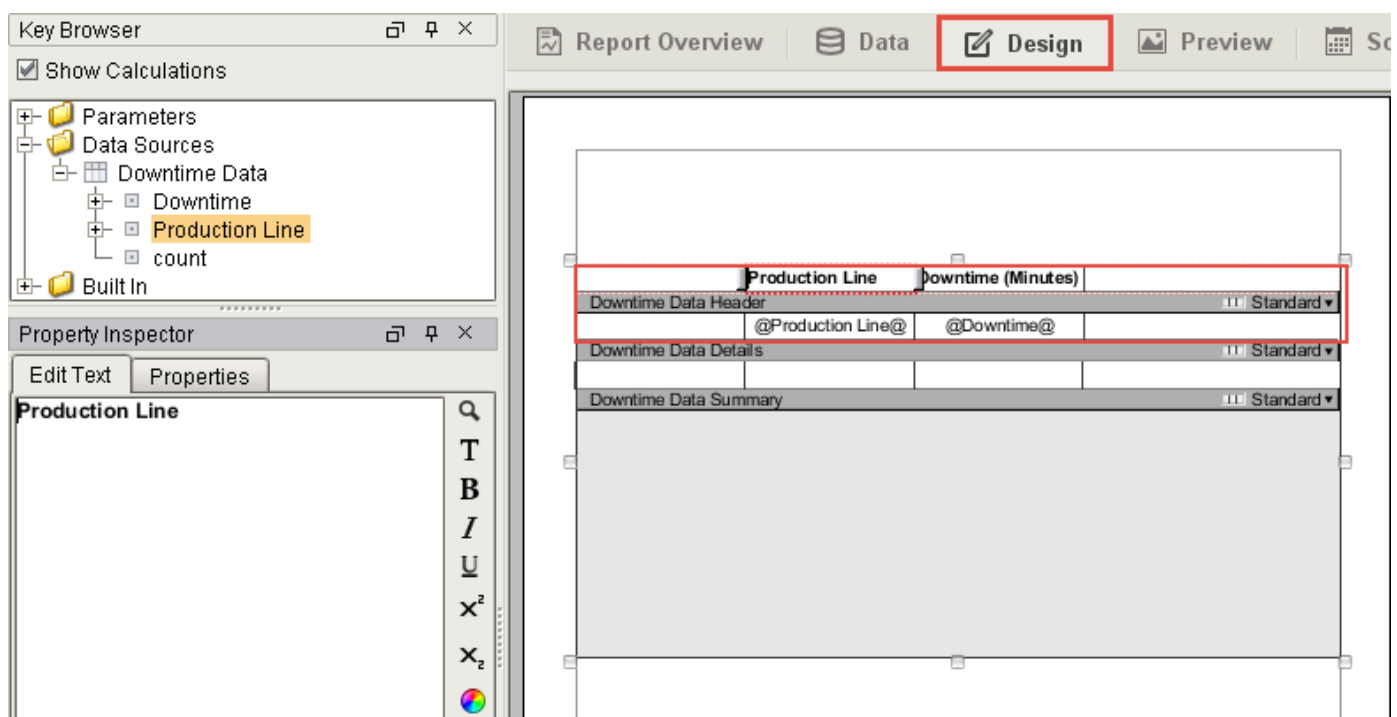


Table Configuration

1. Go to the **Design** panel, and drag a **Table** component into the page.
2. From the **Key Browser**, expand the **Datasources** folder, and drag your data source (i.e., Downtime Data) into the **Data Key** field of the **Configure Table** tab of the Property Inspector.
3. Add **Header** and **Summary** rows by marking the checkboxes.



4. Now, let's configure the data in the Table. With the Table still selected, drag each of the data keys to a cell on the table above the Data Details row: (i.e., Production Line, and Downtime).
5. Next, let's add column headers by selecting a cell above one of our details cells, and enter the appropriate column header name (i.e., Production Line, and Downtime [Minutes]). You can change the font style, size, and color using the functions on the right side of the **Edit Text** Tab. Bold the headers using the **Bold B** button.



6. Go to the **Preview** panel to view your report with the data. It's very common to make formatting changes once you see how your data looks in the report.

Production Line	Downtime (Minutes)
Line A	75
Line B	92
Line C	43
Line D	54
Line E	66
Line F	80
Line G	40
Line H	88

Finishing Touches

1. Back in the **Design** panel, let's make a few changes. Select the "**@Production Line@**" in the Details row and click on the **Properties Tab** to see all the cell properties. Let's change the **Horizontal Alignment** property from left to **Center** so the data falls nicely under the header. Repeat this for "**@Downtime@**."

❗ CHECKING YOUR TABLE IN THE REPORT

At any point during the table configuration process you can go to the Preview panel and see how your table looks with all the data. Configuring the data and designing the table layout is an iterative process. You can go back and forth between the Design and Preview panels as many times as you want making and viewing changes until you get the results you want.

2. Let's add the total downtime by using the **'total'** calculation. In the **Key Browser**, set **Show Calculations** to **'true'**.
3. The Data Summary row is a good place to show the total number of downtime minutes. Expand the **Downtime** data key and you'll see a list of calculations you can use. Drag the **'total'** key to a cell in the Data Summary row.
4. You can even give the total number of downtime minutes a title, make it blue, and bold it (i.e., Total Downtime). If the cell is too small to show the title, you can make the cell larger by dragging the cell border either to the left or right. You can also select the Data Summary row, go to the **Properties tab** and change the **Column Count** (i.e., 2), as shown in the screenshot below. Click the **Preview** tab to check your work.

Preview a Report

The Preview panel lets you validate that all your data is organize and formatted, and you're completely satisfied with how your data is presented in the report. Once your satisfied, you can **schedule the report** to be run and delivered automatically.



Weekly Production Downtime

August 2017

Production Line	Downtime (Minutes)
Line A	75
Line B	92
Line C	43
Line D	54
Line E	66
Line F	80
Line G	40
Line H	88

Total Downtime: 538

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Table Rows

Row Types

Rows are an important fundamental aspect of tables. There are three types of table rows: **Header**, **Details**, and **Summary**. The different types of rows can be independently enabled for each level of **Grouping**, and for each table in a **Table Group**. Adding or removing the Header, Details, or Summary rows is as simple as selecting the Row boxes in the Configure Table tab of the Property Inspector. A row can also be structured or unstructured. A structured row having a variable number of columns, and unstructured row having no columns so you can add images, charts, data, and text shapes anywhere in the row. Additionally, **Table Row Versioning** gives you the option of conditionally displaying rows with a different format to make them stand out.

To learn more about Tables, go to [Report Tables](#) and [Table](#) in the Appendix.



INDUCTIVE
UNIVERSITY

**Table Header, Details and
Summary Rows**

[Watch the video](#)

Header Row

The Header Row allows for a single row to be placed before the Details rows. This is commonly used to create a header for the Table. In many cases where one header is used, the other header could be used equivalently in its place. An interesting feature of the header row is the **Reprint When Wrapped** property which allows the header row to be reprinted on a new page when its data crosses a page boundary. To disable this feature, uncheck the **Print When Wrapped** property in the Properties tab.

Automatic Text Resizing

New in 7.9.6 As of 7.9.6, if text in a Header Row is so long that it will result in truncation or overflow, the size of the text will automatically resize to accommodate.

Detail Rows

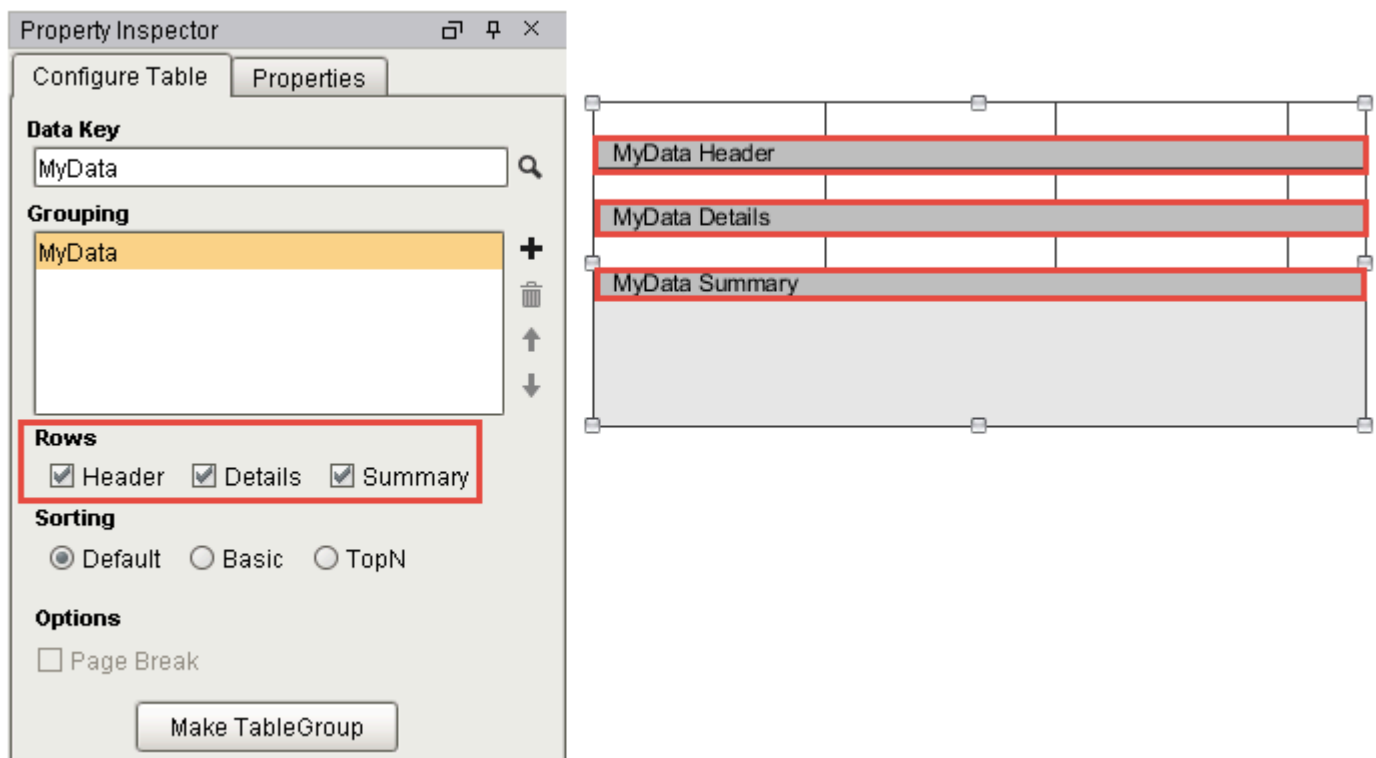
The Detail Rows typically represent the majority of the data on a table or commonly referred to as the "middle" rows. Once you configure the Detail rows with your datasource, the table displays your data in a structured, repetitious format. You can customize the layout and organize the data to determine how you want your data to look on the report. You can even disable Detail Rows in unusual situations such as only displaying aggregate summaries.

Summary Row

The Summary Row is like the Header row only it prints at the bottom of the table, and is typically used to display aggregates for keys. They are typically used in conjunction with some of the [Show Calculation keys](#), such as count, total, running total, etc.

Configuring Header, Details, and Summary Rows

Adding Header, Details, and Summary Rows is super easy. Once a Table component is in your Design Panel and your Datasource is populated in the Data Key field, check the **Header** and **Summary** boxes. The Header, Details, and Summary rows will be added to your report. The **Details** box will be checked by default. To remove the Header, Details, or Summary rows, uncheck the applicable rows.



Structured vs Unstructured Rows



INDUCTIVE
UNIVERSITY

Table Row - Structured vs Unstructured

[Watch the video](#)

Structured Rows behave like a row in a spreadsheet: a series of columns are horizontally adjacent to each other. In the case of a row, the columns are Text Shapes. A structured row can have a variable number of Text Shapes. Structured rows provide more control over the layout, wrapping, and organization of text based table data. Since structured rows only allow for **Text Shapes**, they can not contain components like images, barcodes, charts, and text shapes.

To view the properties of the Details row (or any row), super-select the row by clicking on the Details row (dark bar in the table), or clicking the **Details** node in the Project Browser tree. You can confirm that a row is structured by looking at the **Structured Columns** property in the Properties Tab, or by looking at the **Row Structure** icon  next to the **Standard** row version label. You can also change the number of columns in a row, using the **Column Count** property.

You can tell when a row (any row) is selected by looking at the Project Browser tree, or by looking at the red outline of the rows in a table. A row's content is above its respective dark row bar.

Unstructured Rows are functionally very similar to structured rows, but they do not offer the same column based text constraints. Instead, Unstructured Rows allow you to place any other report component inside of the row. Unstructured rows are highly customizable rows allowing you to place data, text shapes, images, or charts anywhere you want them within the row.

There are two ways to make a row unstructured once your row is selected:

- In the Property Tab of the Property Inspector, set the **Structured Columns** property to '**false**'.
- In your Table, click on the **Row Structure icon**  on the right side of your row. Click the icon to make the row unstructured . You can toggle the Row Structure icon to switch between a structured row and unstructured row.

Production Line		Downtime (Minutes)
Downtime Data Header		Standard ▾
@Production Line@	@Downtime@	
Downtime Data Details		Standard ▾
Total Downtime: @total.Downtime@		
Downtime Data Summary		Standard ▾

Now that the Header row is unstructured, drag the Header row down the page making room to add any component(s) from the Report Design Palette, and place it anywhere in the row. In this example, you can see a Text Shape was added for the report title, an image was dragged from the Image Management Tool, and a Line Shape was drawn to separate the image from the report data.

Product Line Downtime Report 		
Production Line		Downtime (Minutes)
Downtime Data Header		Standard ▾
@Production Line@	@Downtime@	
Downtime Data Details		Standard ▾
Total Downtime: @total.Downtime@		
Downtime Data Summary		Standard ▾

Here is what the report looks like in the **Preview** panel.

Production Line Downtime



Production Line	Downtime (Minutes)
Line A	75
Line B	92
Line C	43
Line D	54
Line E	66
Line F	80
Line G	40
Line H	88

Total Downtime: 538

Report Data Configuration

In addition to configuring rows, you can also configure how you want your data to appear in a report using the **Sort** and **Filter** functions in the Table component.

Sorting

Sorting orders your data by a single data key or list of data keys. There are three types of Sorting in Tables.

- **Default** - data is sorted based on the order in which it is retrieved.
- **Basic** - takes a list of data keys and sorts by the first one. If the sort results in a tie, the tie will be resolved by the next data key in the list, and so on.
- **TopN** - uses a single key path, with a **Count** value that allows a limit to the number of rows that are processed.

Basic and **TopN** sorts can be configured for either ascending (↗) or descending (↘) sorts. They can also utilize aggregate (calculation) keys. The **TopN** sort option, **Include Others** ☐ Include "Others" if selected, will include all values outside of the specified **Count** range by compressing them into a single row.

Sort and Filter Examples

The CSV dataset containing all the data and the table used for the following Sort and Filter examples are shown below.

Dataset

Code Block | language = js | title = Dataset

```
Order Number,Date Shipped,Subtotal,Sales Person
10248,"Jul 16, 2017",440,"Buchanan, Steven"
10252,"Jul 10, 2017",1863.40,"Leverling, Janet"
10260,"Jul 18, 2017",517.80,"Smith, Steven"
10249,"Jul 17, 2017",624.30,"Howard, Rodney"
10250,"Jul 16, 2017",1444.20,"Johnson, Grace"
10255,"Jul 18, 2017",2018.25,"Jack, Lisa"
10253,"Jul 16, 2017",101.10,"Wilson, George"
10251,"Jul 16, 2017",1259.40,"Lawrence, Sara"
10254,"Jul 16, 2017",843.60,"Buchanan, Steven"
10259,"Jul 16, 2017",1050.75,"Smith, Steven"
10257,"Jul 16, 2017",999.50,"Howard, Rodney"
10256,"Jul 16, 2017",2114.30,"Johnson, Grace"
10258,"Jul 16, 2017",288.50,"Wilson, George"
```

Table in the Design Panel

Acme Reporting Company			@Date@
Making Cents from Reporting Sense!			
<u>Shipped Orders Report</u>			
Order Number	Date Shipped	Subtotal	Sales Person
Sales Data Header			
@Order Number@	@Date Shipped@	\$ @Subtotal@	@Sales Person@
Sales Data Details		Order Totals: \$ @total.Subtotal@	
Sales Data Summary			

Default Sort Example

The data for the **Default** sort is retrieved directly as-is from the dataset. No sort order is applied.

Configure Table

Property Inspector

Configure TableProperties

Data Key

Sales Data

Grouping

Sales Data

Rows

☒ Header☒ Details☒ Summary

Sorting

☒ Default☐ Basic☐ TopN

Options

☐ Page Break

Make TableGroup

Default Sort Results

Acme Reporting Company				Aug 22, 2017
Making Cents from Reporting Sense!				
Shipped Orders Report				
Order Number	Date Shipped	Subtotal	Sales Person	
10248	Jul 16, 2017	\$ 440.00	Buchanan, Steven	
10252	Jul 10, 2017	\$ 1,863.40	Leverling, Janet	
10260	Jul 18, 2017	\$ 517.80	Smith, Steven	
10249	Jul 17, 2017	\$ 624.30	Howard, Rodney	
10250	Jul 16, 2017	\$ 1,444.20	Johnson, Grace	
10255	Jul 18, 2017	\$ 2,018.25	Jack, Lisa	
10253	Jul 16, 2017	\$ 101.10	Wilson, George	
10251	Jul 16, 2017	\$ 1,259.40	Lawrence, Sara	
10254	Jul 16, 2017	\$ 843.60	Buchanan, Steven	
10259	Jul 16, 2017	\$ 1,050.75	Smith, Steven	
10257	Jul 16, 2017	\$ 999.50	Howard, Rodney	
10256	Jul 16, 2017	\$ 2,114.30	Johnson, Grace	
10258	Jul 16, 2017	\$ 288.50	Wilson, George	
Order Totals: \$ 13,565.10				

Basic Sort Example

This table was sorted by the **Order Number** data key in **ascending order**. To sort in descending order, right-click on the **Order Number** and select **Sort Descending**.

Configure Table

Property Inspector

Configure Table

Properties

Data Key

Sales Data

Grouping

Sales Data

+

↑

↓

Rows

☒ Header

☒ Details

☒ Summary

Sorting

☐ Default

☒ Basic

☐ TopN

1. Order Number

+

↑

↓

Options

☐ Page Break

Make TableGroup

Basic Sort Results

Acme Reporting Company

Aug 22, 2017

Making Cents from Reporting Sense!

Shipped Orders Report

Order Number	Date Shipped	Subtotal	Sales Person
10248	Jul 16, 2017	\$ 440.00	Buchanan, Steven
10249	Jul 17, 2017	\$ 624.30	Howard, Rodney
10250	Jul 16, 2017	\$ 1,444.20	Johnson, Grace
10251	Jul 16, 2017	\$ 1,259.40	Lawrence, Sara
10252	Jul 10, 2017	\$ 1,863.40	Leverling, Janet
10253	Jul 16, 2017	\$ 101.10	Wilson, George
10254	Jul 16, 2017	\$ 843.60	Buchanan, Steven
10255	Jul 18, 2017	\$ 2,018.25	Jack, Lisa
10256	Jul 16, 2017	\$ 2,114.30	Johnson, Grace
10257	Jul 16, 2017	\$ 999.50	Howard, Rodney
10258	Jul 16, 2017	\$ 288.50	Wilson, George
10259	Jul 16, 2017	\$ 1,050.75	Smith, Steven
10260	Jul 18, 2017	\$ 517.80	Smith, Steven
Order Totals: \$ 13,565.10			

TopN Sort Example

This table was sorted by the **Subtotal** data key in **ascending order** with a **Count** value of **10** rows. To sort in descending order, click the **Ascending icon**  next to the Data Key to switch to **Sort Descending**.

Configure Table

Property Inspector

Configure Table

Properties

Data Key

Sales Data

Grouping

Sales Data

Rows

☒ Header
☒ Details
☒ Summary

Sorting

☐ Default
☐ Basic
☒ TopN

Data Key

Subtotal

Count (N)

10

☐ Include "Others"

Options

☐ Page Break

Make TableGroup

TopN Sort Results

Acme Reporting Company				Aug 22, 2017
Making Cents from Reporting Sense!				
<u>Shipped Orders Report</u>				
Order Number	Date Shipped	Subtotal	Sales Person	
10253	Jul 16, 2017	\$ 101.10	Wilson, George	
10258	Jul 16, 2017	\$ 288.50	Wilson, George	
10248	Jul 16, 2017	\$ 440.00	Buchanan, Steven	
10260	Jul 18, 2017	\$ 517.80	Smith, Steven	
10249	Jul 17, 2017	\$ 624.30	Howard, Rodney	
10254	Jul 16, 2017	\$ 843.60	Buchanan, Steven	
10257	Jul 16, 2017	\$ 999.50	Howard, Rodney	
10259	Jul 16, 2017	\$ 1,050.75	Smith, Steven	
10251	Jul 16, 2017	\$ 1,259.40	Lawrence, Sara	
10250	Jul 16, 2017	\$ 1,444.20	Johnson, Grace	
Order Totals: \$ 7,569.15				

Filtering

Filtering gives the option of processing data based on an expression. The **Filter Key** property can be found in the Property Inspector under the **Properties** tab.

NOTE

Once you enter your expression in the **Filter Key** field, hit return to commit your expression. If the expression resolves to **'false'**, the row will be skipped.

Filtering Example

This example is sorted in ascending order using the **Subtotal** data key, and filtered for any **'Subtotal>1000'**. To alternate to descending order, click the **Ascending icon** (↕) to toggle to a descending sort.

Configure Table & Property Tab

Property Inspector

Configure Table

Properties

Data Key

Sales Data

Grouping

Sales Data

Rows

☒ Header ☒ Details ☒ Summary

Sorting

☐ Default ☐ Basic ☒ TopN

Data Key

Subtotal

Count (N)

10

☐ Include "Others"

Options

☐ Page Break

Make TableGroup

Property Inspector

Configure Table

Properties

Table

Data Key	Sales Data
Column Count	1
Column Spacing	10
Filter Key	Subtotal>1000
Starting Page Break	☐

Filtering Results

Acme Reporting Company

Aug 22, 2017

Making Cents from Reporting Sense!

Shipped Orders Report

Order Number	Date Shipped	Subtotal	Sales Person
10259	Jul 16, 2017	\$ 1,050.75	Smith, Steven
10251	Jul 16, 2017	\$ 1,259.40	Lawrence, Sara
10250	Jul 16, 2017	\$ 1,444.20	Johnson, Grace
10252	Jul 10, 2017	\$ 1,863.40	Leverling, Janet
10255	Jul 18, 2017	\$ 2,018.25	Jack, Lisa
10256	Jul 16, 2017	\$ 2,114.30	Johnson, Grace
		Order Totals: \$ 9,750.30	

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Table Row Versioning



INDUCTIVE
UNIVERSITY

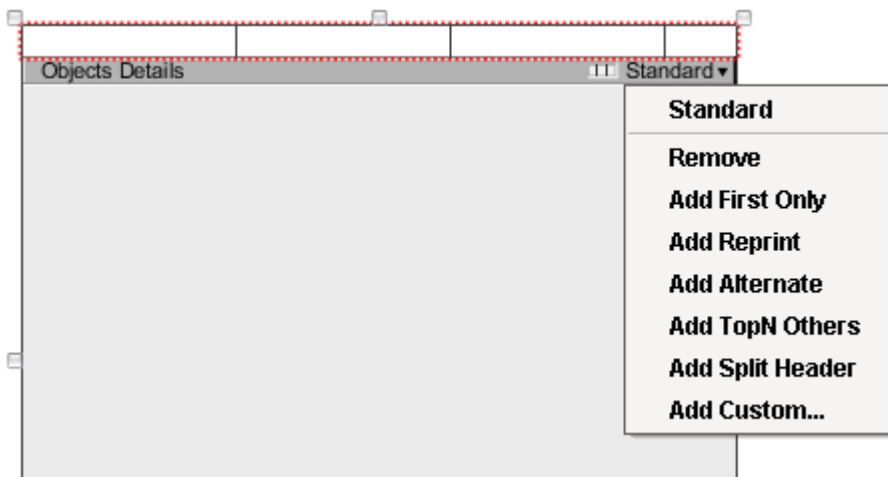
Table Row Versioning

[Watch the video](#)

Table Row Versioning

Table Row Versioning allows you to conditionally display rows of data in different formats. It is used to make certain data stand out or to make a report more legible. It allows you to do things like create alternate background row colors, or make the first row different, or show negative valued rows differently. You will see a list of your existing row versions (only one named "Standard" appears first in the list) followed by options to add new row versions. When creating new row versions, the currently selected row will be duplicated as the new row version. From here, you can modify your row in any way. You can make it similar to the Standard row or completely different. Once you added a few row versions, you can swap between them by selecting one from this list.

To use Row Versioning, click on the word "Standard" on the far right of a row. Row versions are either **Built-in** or **Custom**, and may be specified with a version key expression. They are applicable to the **Header and Summary Rows**, but are most often added to the Detail Row.



Once a Row Version has been added, it becomes selectable in the dropdown. The word "Standard" also gets replaced with whatever Row Version I have currently selected to edit. In this case, I added an Alternate Row Version, and it is currently selected.

 CAUTION

It is important to always be aware of what Row Version you are currently working on by checking the name of the Row Version on the right of the Table row. After navigating away from the design panel, the Table will automatically display the standard Row Version on return to the Design panel.

Here is a list of the Row Version types and their descriptions.

Row Versions	Description
Standard	Default row version. When adding a new Row Version, it will start off as a copy of the Standard Row Version.
Remove	Removes the currently selected Row Version. The Standard row cannot be removed.
First Only	Applies only to the first instance of the row. Good for showing header information without using an upper level Detail row. This is not the same as a Header row.
Reprint	Applies to every page after the first. Good for one time headers or (continued) indications to save space.
Alternate	Applies to every other row starting with the second row. Good for changing the background color on alternate rows.
TopN Others	Applies to count number of rows in a TopN sort. Using "Include Others" will then distinguish between TopN and non-TopN rows.
Split Header	Applies to Headers that have been split due to excessive height. Good for providing "Continued" type indicators.
Custom	Create your own Row Version. When this option is selected, you must enter a label for the new row version. Instead of being used automatically like with the other Row Versions, all custom Row Versions are driven by the Version Key Property.

Using Row Versions

Once a new Row Version has been added to the table row, it is easy to configure each row version to appear visually distinct.

When using a Table to display a large number of rows, an Alternate Row Version can be added to alternate between two fill colors, which in turn improves readability. We start off by adding a table to the report, assign a data source to the table and add some data keys to the text shapes in the Details row. In this example, we will alternate between white and orange, so we will leave the standard row's **Fill** property disabled to use the default color of our paper.

I then add an Alternate row to the details row of the table. On the Alternate row, we can enable the **Fill** property and set **Fill Color** to an orange background.

Taking a look at a preview of the report, we can see that the Alternate row with the orange color is automatically used on every other row starting with the second row.

Sync Alternate Versions

Once a new row version has been created, it is not linked to the standard row in any way, so configuration changes made to one row will not be automatically applied to other row versions.

The common case where this behavior becomes a problem is when the width of one or more Text Shapes in a row are modified. Here we see a row with an Alternate version using a different **Fill**

Color. After the Alternate row version was created, the second and third Text Shapes on the Standard version has their **Width** increased, causing the text inside to shift. However, the Text Shapes on the Alternate version will not automatically resize themselves by default, so the Preview Panel shows offset columns on the Alternate version.

Instead of manually adjusting the Text Shapes on the Alternate version, we can toggle the **Sync Alternate Versions** property on the Standard row. Assuming both columns have the same value for **Column Count**, enabling **Sync Alternate Versions** will automatically resize the widths of Text Shapes on the alternate row version to match the widths of the standard version Text Shapes.

Heading back to the Preview Panel, we see that the Text Shapes on both rows are now synchronized.

Custom Row Version

Custom row versions are ideal when the built-in Row Versions don't fit your needs. Custom versions are identified by a string-based name, and will be used when the Version Key property is a string that matches the Row Version name. If that string equals the name of a Row Version, that Row Version will be used. An invalid string will default back to normal built-in Row Version behavior. The Version Key property also accepts Data Keys used in expressions, using the same syntax that **Keychain Expressions** use. For example, if I wanted to highlight red all of the rows where the Inventory is less than 20, I can use a simple expression of:

```
Inventory<20?"Row3"
```


Where "Row3" is the name of my custom Row Version enclosed in quotes. This can help me keep track of low inventory, excessive downtime, or anything that may be important to highlight.

Finally, it is actually possible to use many different Row Versions in conjunction with each other. You can setup multiple custom rows and have a more complex expression that helps decide when a particular Row Version gets used. You can also use multiple built-in rows, and even a combination of built-in rows and custom rows. In the event of a conflict between the custom row and a built in row, the custom row will take precedence. For example, when combining my custom Row3 with the alternate rows, you can see that the row Snickers would be the next alternate row in the table, but since it has less than 20 inventory, it uses the custom Row3 instead.

You can also reference multiple row versions in the same conditional expression:

```
Inventory<20?"Row3":Inventory<40?"Row4":"Row5"
```

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Charts Inside of Tables

Adding Charts Inside of Tables

Adding charts inside of tables provides a lot of flexibility designing reports as well as organizing and displaying data in a report. The most common way of adding charts inside tables is using an unstructured row and placing the chart inside the row. Unstructured rows are highly customizable allowing you to place charts anywhere within the row of the table.

There are two common locations to place a chart inside a table:

- Using a Header followed by a chart component.
- Using a Details Row followed by a chart component.

It is very common to add a chart in an unstructured Header row because then the chart will be at the top of the first page of a report. An unstructured Details row typically works best when used in conjunction with [Grouping Data Inside of Tables](#) or [Nested Queries](#) so that each row has a chart with data from that group or query.

Adding a Chart Inside a Header Row

This example shows several pieces of hardware including how many pieces were produced, and how many pieces were shipped. A Bar Chart was added after the Header row in the table.

1. In the **Data** panel, create a Static CSV data source (i.e., Hardware Data). The dataset is shown below.

Hardware Dataset

```
Hardware, Produced, Shipped
Nails, 95, 85
Screws, 80, 60
Bolts, 50, 47
```

2. In the **Design** panel, drag a Table component on to your report. Drag your **Datasource** (i.e., Hardware Data) from the **Key Browser** to the **Data Key** field, and click the **Header** box in the **Configure Tab**.
3. Next, drag each **Data Key** (i.e., Hardware, Produced, and Shipped) to a column in your table in Data Details row.
4. Enter the Header name (i.e., Metal Fastener Numbers Report) in the left column of the Data Header row.

5. Select the Header row and click on the **Row Structure** icon  to make the row unstructured so you can add a chart.

 NOTE

You can toggle between a structured and unstructured row by simply clicking the Structure Row icon.

6. With the Header row selected, drag the row down the page to make room for the chart: click and drag on the dark gray bar titled "Hardware Data Header".
7. Drag in a Bar Chart from the Report Design Component Palette into the unstructured Header row and expand it. While dragging the chart into the header, you'll notice a blue outline around the

Header as you hold the mouse button down while the cursor is in front of the header. This signifies that the chart will be placed directly into the header row, as opposed to in front of the table.

8. With the chart selected, drag a **Datasource** (i.e., Hardware Data) to the **Data Key** of the **Configure Tab**. Set the **Extract Order** as 'Column.'

ⓘ **EXTRACT ORDER**

For a Bar Chart, Extract Order is the order in which the data is extracted from the data key, by Row or Column. By default, Row is selected. Use Row when columns in the data key define the series of data. Use Columns when rows in the data key define the series of data; each column is a new value for the same series.

9. Now, let's put your creativity to work and make your report stand out by changing properties. This example changed the following properties:

- In the **Configure Tab** - change the third bar color to green.
- Select **Metal Fastener Numbers Report** header - change the Font Size to 24 pixels, the Text Color to red, and the Horizontal Alignment to Center in the Properties tab.
- Select the **Bar Chart** - enter a name for the Axis Label (i.e., Quantity), and set the Bar Labels to 'true' in the Properties Tab.
- Select each **Data Details cell** (i.e., Hardware, Produced, and Shipped), change the Font Size to 14 pixels, and the Hardware Text to blue in the Properties tab.

The Design panel will update the displayed sample data after changing the properties above.

10. Go to the **Preview** panel to view your report.

Adding a Chart Inside a Details Row

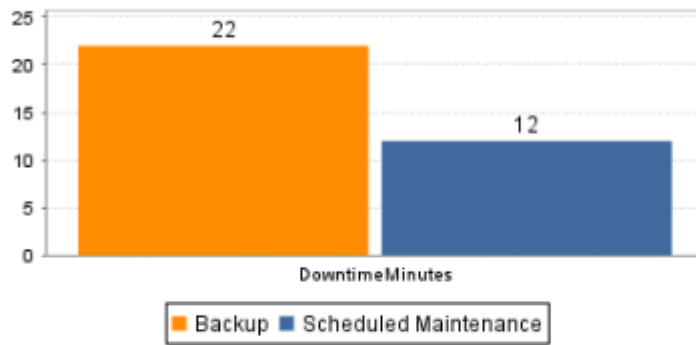
Adding a chart inside a Details row duplicates the chart for the number of Detail rows you have in your table, and displays the respective data for each row in the chart. This example shows the Equipment Downtime Details for each piece of equipment: Cause and Downtime in Minutes. It also adds a Bar Chart after the Details row using an unstructured row, and a nested query so that each row has a chart with data from the query. Go to the [Nested Queries](#) page and complete the Equipment Downtime example at the bottom if you want to use the same datasets for this example.

1. Once you have your Data set up from the above example, click on the **Design** tab and add a table component to your report.
2. With the table created, add a Header row to the table. Drag all of your Equipment details into the table, and add header titles for each column.
3. Select the **Details** row, make the row **unstructured** , and drag the row down the page to make room for a chart.
4. Drag a chart from the Report Design Component Palette into the unstructured row and expand it.
5. With the chart selected, drag a **Datasource** (i.e., EquipDowntime) from the **Key Browser** to the **Data Key** field in the **Configure Tab**.
6. If you are using a Bar Chart, set the **Extract Order** to **Column**.

7. Go to the **Preview** panel to view the report. If you want to make any changes to the chart like changing Segment Colors, adding Bar Labels, or adding a page break, etc., go back to the **Design** panel to modify any of the chart properties. When you're finished, return to the **Preview** panel. You'll notice that for each piece of equipment there is a chart and data. You can make each piece of equipment have its own page by setting the Page Break option on the Table under the Configure Table tab.

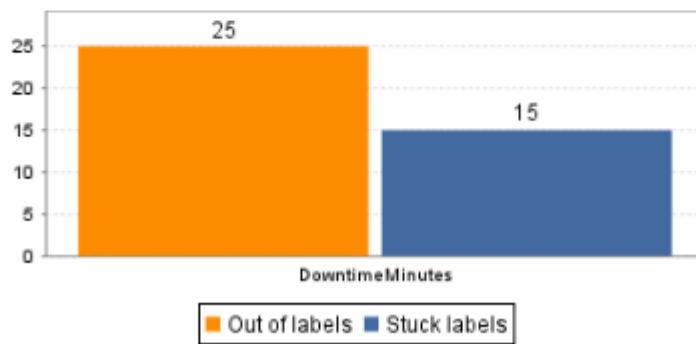
Equipment Downtime Details

Equipment ID	Equipment Name	Description
25489	Conveyor Line	Transfers product



Downtime Cause	Downtime (Minutes)
Backup	22
Scheduled Maintenance	12

55684	Labeler	Makes labels
-------	---------	--------------



Downtime Cause	Downtime (Minutes)
Out of labels	25

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Grouping Data Inside of Tables



INDUCTIVE
UNIVERSITY

Dataset Grouping

[Watch the video](#)

Another important feature of tables is the ability to separate a single dataset into different categories or groups. When using a table in a report, you can group data in the table by a specific column. With Dataset Grouping, you can break tables down by data keys that share a common value (i.e., if you have a table that shows addresses, you can group the rows by the city, state, zip code, or any combination of the columns). This is done by dragging and dropping any of your data keys from the Key Browser to the Grouping list under the Configure Table tab.

When you add a data key to the Grouping list, a corresponding Details row will be added to your table component. Using dataset grouping allows you use data keys to organize and arrange your data into different categories, organizing the results based on the values of a key. Each group can have its own **Header, Details, and Summary rows**. Additionally, the keys from **Show Calculations** and other **keychain functions** are supported for any level of grouping.

NOTE

Table Groups and Grouping Data in Tables are two completely different things despite having similar names. Table Grouping involves using multiple datasets in the same Table component while Grouping Data Inside of Tables (this page) sorts the rows inside a single dataset.

Demonstration

Assuming an initial table that looks like the following:

Type	Count
Type 1	100
Type 2	45
Type 2	450
Type 4	123
Type 3	50
Type 1	250
Type 3	871
Type 2	984

We could utilize Dataset Grouping to group the results in the table by unique "**Type**" values. By adding a grouping on the Type column, and some additional formatting, we can produce a table that looks like the following:

Type 1	Count
	100
	250
Type 2	Count
	45
	450
	984
Type 3	Count
	50
	871
Type 4	Count
	123

Notice that we're no longer listing each type individually. Instead, the type acts as a sub-header for each group of data. See the example below for a how-to.

Grouping Data Inside of a Table Example

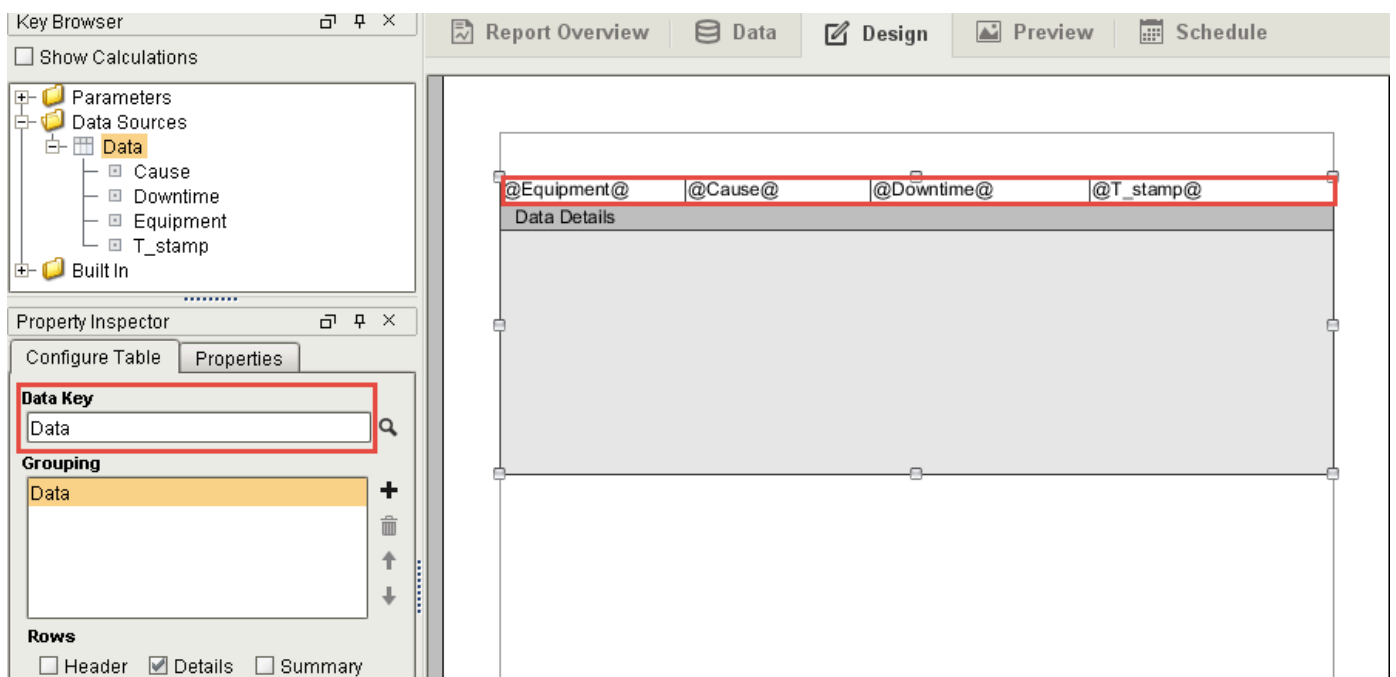
This example begins with a table similar to the one created in the [Report Workflow Tutorial](#). This example will demonstrate how to group the existing downtime report by equipment, collect downtime totals, and introduce some formatting techniques.

1. In the **Data** panel, create a Data Source that has a Timestamp, Equipment, Downtime, and Cause. Shown below is the text needed for a **Static CSV** datasource used for this example.

Data Source for Equipment Downtime Report

```
T_stamp, Equipment, Downtime, Cause
"Jan 20, 2017 17:55", "Labeler", 50, "Out of labels"
"Feb 20, 2017 18:40", "Filler", 120, "Overflow"
"Feb 28, 2017 12:45", "Palletizer", 21, "Misalignment"
"Feb 12, 2017 20:13", "Labeler", 98, "Stuck labels"
"Jan 21, 2017 18:15", "Conveyor Line", 27, "Backup"
"Feb 25, 2017 16:22", "Filler", 2, "Scheduled Maintenance"
"Feb 13, 2017 19:19", "Conveyor Line", 21, "Scheduled Maintenance"
"Jan 20, 2017 15:30", "Palletizer", 241, "Misalignment"
```

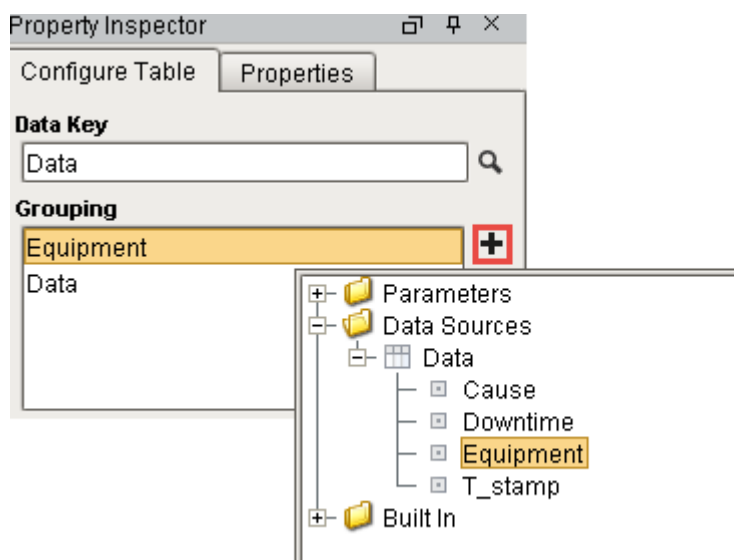
2. In the **Design** panel, drag a **Table** component to your report.
3. With the Table selected, drag the **Data** data source to the **Data Key** under the **Configure Table** tab of the Property Inspector.
4. Drag each of the **data keys** (i.e., Equipment, Cause, Downtime, and T-Stamp) to any of the columns in the table row.



5. Click on the **Preview** panel to see what the report looks like. You'll notice that all your data is there, but it's a little hard to read because it's not organized.

 Design  Preview  Schedule			
Labeler	Out of labels	50	Jan 20, 2017 17:55
Filler	Overflow	120	Feb 20, 2017 18:40
Palletizer	Misalignment	21	Feb 28, 2017 12:45
Labeler	Stuck labels	98	Feb 12, 2017 20:13
Conveyor Line	Backup	27	Jan 21, 2017 18:15
Filler	Scheduled Maintenance	2	Feb 25, 2017 16:22
Conveyor Line	Scheduled Maintenance	21	Feb 13, 2017 19:19
Palletizer	Misalignment	241	Jan 20, 2017 15:30

6. Let's sort each row in the table by equipment and group all the equipment together. Go to the Property Editor, and in the **Configure Table**, click the **Add +** icon next to **Grouping**, and a window will open with a list of data keys. Select the '**Equipment**' row, and click **OK**. You'll notice 'Equipment' was added to the Grouping list, and an Equipment Details row was immediately added to the table.



7. Go to the **Preview** panel to see that the report is now sorting by equipment name.

Data	Design	Preview	Schedule
Labeler	Out of labels	50	Jan 20, 2017 17:55
Labeler	Stuck labels	98	Feb 12, 2017 20:13
Filler	Overflow	120	Feb 20, 2017 18:40
Filler	Scheduled Maintenance	2	Feb 25, 2017 16:22
Palletizer	Misalignment	21	Feb 28, 2017 12:45
Palletizer	Misalignment	241	Jan 20, 2017 15:30
Conveyor Line	Backup	27	Jan 21, 2017 18:15
Conveyor Line	Scheduled Maintenance	21	Feb 13, 2017 19:19

- Let's remove the Equipment Name from each row in the table and add a Header. Go to the **Design** panel and cut '@Equipment@' from the Data Details row and paste it in the **Equipment Details** row. You can bold it to make it stand out.
- While the table component is selected, go to the **Configure Table** tab in the Property Inspector and select the **Equipment** item in the grouping list and check both the **Header** and **Summary** boxes. Then select the **Data** group and select the **Header** and **Summary** checkboxes for it.
- Next, make the Equipment header an **unstructured row** and add text as a title for your report using the **Text Shape** in the component palette. Unstructuring the row allows you to easily center the title of your report.
- Now add header text for each of the Data Details columns (Cause, Downtime, and Date) by typing into the **Data Header** row.

Key Browser

- Parameters
- Data Sources
 - Data
 - Cause
 - Downtime
 - Equipment
 - T_stamp
- Built In

Property Inspector

Configure Table | Properties

Data Key
Data

Grouping
Equipment
Data

Rows
☒ Header
 ☒ Details
 ☒ Summary

Report Overview | Data | Design | Preview | Schedule

Equipment Downtime Report

Grouped by Equipment

Equipment Header				Standard
@Equipment@				
Equipment Details				Standard
	Cause	Downtime	Date	
Data Header				Standard
	@Cause@	@Downtime@	@T_stamp@	
Data Details				Standard
Data Summary				Standard
Equipment Summary				Standard

- Set the **Show Calculations** checkbox at the top of the **Key Browser**. Drill down into the Downtime column and drag the '@total.downtime@' to both the **Data Summary** and **Equipment Summary** rows. You can also add any text outside of the @ symbols. In this example, we added 'minutes' after the total downtime in our cell to '@total.downtime@ minutes'.
- Lastly, do the same in the Equipment Summary row. We also added the word "Total" to the beginning of the cell: 'Total: @total.downtime@ minutes'.

i NOTE

In any **Summary** row, '@total.Downtime@' is a sum of all downtime at that level of grouping; (i.e., in the **Data Summary** row it is the total downtime grouped by equipment. In the **Equipment Summary** row, '@total.Downtime@' is the sum of all downtime for all equipment groupings).

Key Browser

- Parameters
- Data Sources
 - Data
 - Cause
 - Downtime
 - value
 - total**
 - average
 - max
 - min
 - Running.total
 - Running.average
 - Running.count
 - Equipment
 - T_stamp
 - count
- Built In

Property Inspector

Edit Text Properties

Total: @total.Downtime@ Minutes

Report Overview

Report Overview Data Design Preview Schedule

Equipment Downtime Report

Grouped by Equipment

Equipment Header			
@Equipment@			
Equipment Details			
	Cause	Downtime	Date
Data Header			
	@Cause@	@Downtime@	@T_stamp@
Data Details			
		@total.Downtime@ Minutes	
Data Summary			
		Total: @total.Downtime@ Minutes	
Equipment Summary			

Data Details for each Downtime event

Total Summary for each Equipment value

Total Summary of all Equipment values

- Click over to the **Preview** panel to check out your report. If you want to make any changes, go back to the **Design** panel to update your report. Notice that the 'total' data key respects both groupings.

Equipment Downtime Report

Grouped by Equipment

Labeler

Cause	Downtime	Date
Out of labels	50	Jan 20, 2017 17:55
Stuck labels	98	Feb 12, 2017 20:13
	148 Minutes	

Filler

Cause	Downtime	Date
Overflow	120	Feb 20, 2017 18:40
Scheduled Maintenance	2	Feb 25, 2017 16:22
	122 Minutes	

Palletizer

Cause	Downtime	Date
Misalignment	21	Feb 28, 2017 12:45
Misalignment	241	Jan 20, 2017 15:30
	262 Minutes	

Conveyor Line

Cause	Downtime	Date
Backup	27	Jan 21, 2017 18:15
Scheduled Maintenance	21	Feb 13, 2017 19:19
	48 Minutes	

Total: 580 Minutes

Separating Groupings using Page Breaks

In the **Configure Table** tab of the **Property Inspector**, there is a **Page Break** option that can be set to create breaks between each Grouping. Each new instance of that level of grouping creates a new page in the report. In the example above, we could add a page break in-between each grouping of equipment type, which would further delineate each grouping of data. This is especially useful if you are **adding charts** or other images at the beginning of each group.

Property Inspector

Configure Table Properties

Data Key

Data

Grouping

Equipment

Data

Rows

☒ Header ☒ Details ☒ Summary

Sorting

☒ Default ☐ Basic ☐ TopN

Options

☒ Page Break

Make TableGroup

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

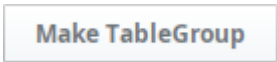
Table Groups

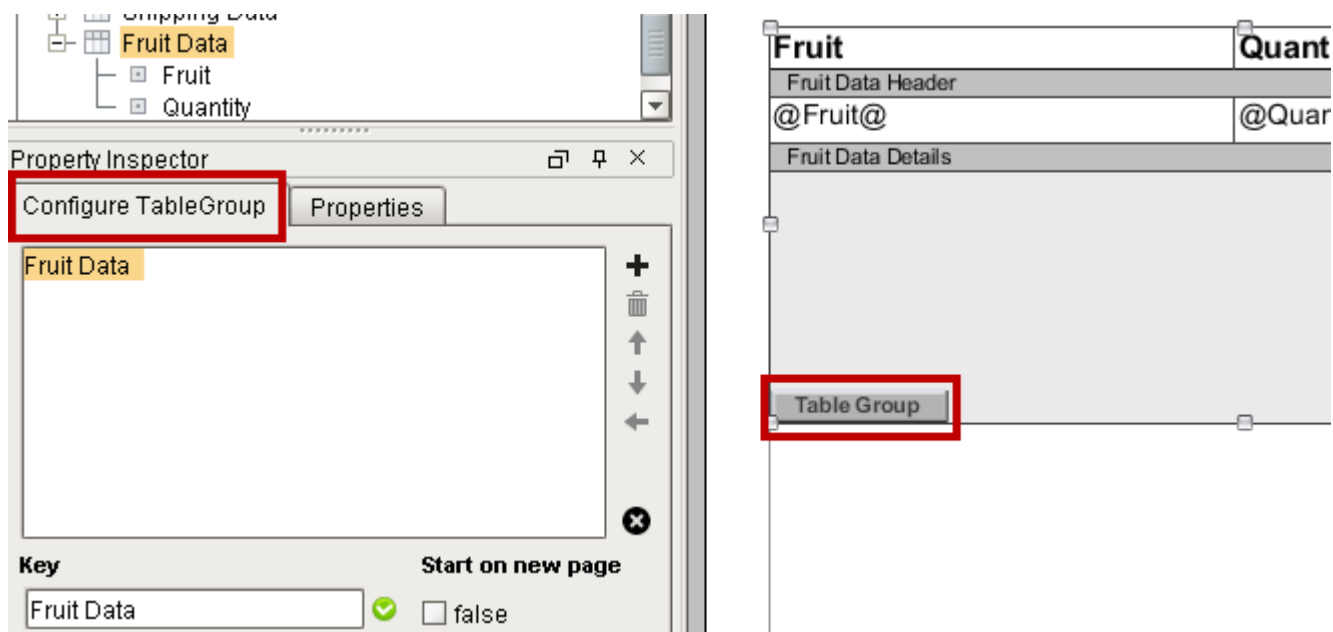
Table groups are a single component that contains many tables, each with its own unique data. There are two types of Table Groups you can create within a Table component: **Peer Tables** and **Child Tables**. Peer Table Groups allow the second table to begin exactly where the first table ends. If multiple tables are used, they can be configured as multiple page templates, providing a page break between tables. Child Table Groups allow you to nest one table inside another, and work especially well with any **Nested Queries**. What's really nice about Table Groups and Nested Data Sources is that you can create Summary Tables for categories of items or drill-down charts all in one report.

NOTE

Table Groups and Grouping Data in Tables are two completely different things despite having similar names. Table Grouping (this page) involves using multiple datasets in the same Table component while Grouping Data Inside of Tables sorts the rows inside a single dataset.

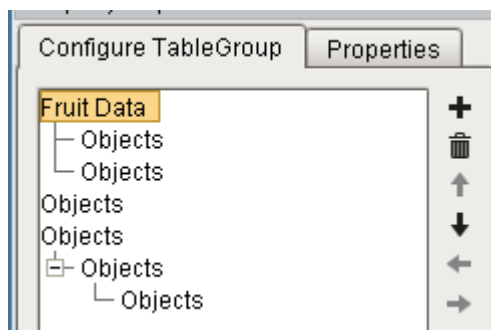
Making a Table Group

To make a Table Group, you simply need to click the **Make TableGroup**  button in the Configure Table tab of your table. You will be able to tell that it worked when you notice a **Table Group**  icon displayed in the lower left corner of the table component, and your **Configure Table** tab has now changed into a **Configure TableGroup** tab.



We can see our original table listed in the **Configure TableGroup** tab, and we have the option to add additional tables by clicking the plus **+** icon. Adding a child table will add a table that is a child to the table that is currently selected in the Table Group list. Adding a peer table will add a table that is a peer to the currently selected table. We can add as many of each of these as we need.

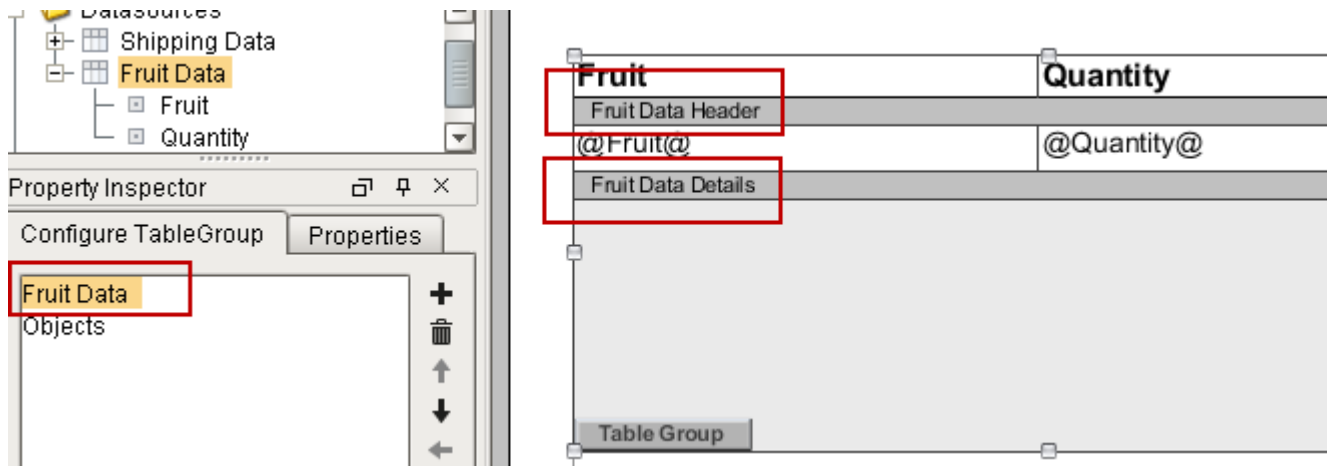
For example, in the image below we added two child tables to the original Fruit Data table. We then have two peer tables to Fruit Data, one of them is a standalone table, while one has a child table with that child table also having a child table. More information on child and peer tables is included below.



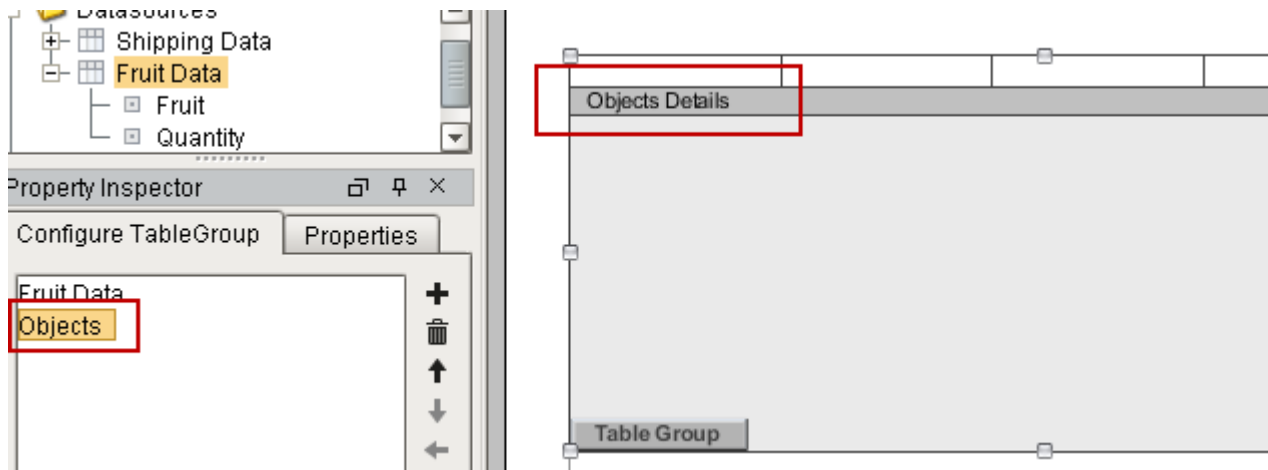
As you can see from the image above, the Table Group hierarchy can get quite confusing very quickly. It is important to understand how each part of the Table Group works, and to always be aware of which table you are actually looking at inside the Table Group. To view a different table within the Table Group, simply select it from the hierarchy. The table displayed in the design area will switch depending on which table you selected.

Navigating Within Table Groups

Fruit Data Selected



Objects Selected



This allows you to get a quick preview of how each of the tables within the Table Group are set up. To go in and configure an individual table within the Table Group, select it from the hierarchy and then click on the table object in the design area. The Configure TableGroup tab in the Property Inspector should change to the Configure Table tab, and will display properties relevant to the table you had selected. This allows you to add a Data Key, set grouping, configure relevant row styles, and format the table in any way that you want. After configuring the table, we can navigate back to the Table Group configuration by clicking on the **Table Group**  icon in the bottom left corner of the table.

At the bottom of the Configure TableGroup tab, there is an option called **Start on new page**. With this option enabled, we can force the next table in the list to start on the next page instead of immediately after the previous table. This will only apply to the table that is currently selected.

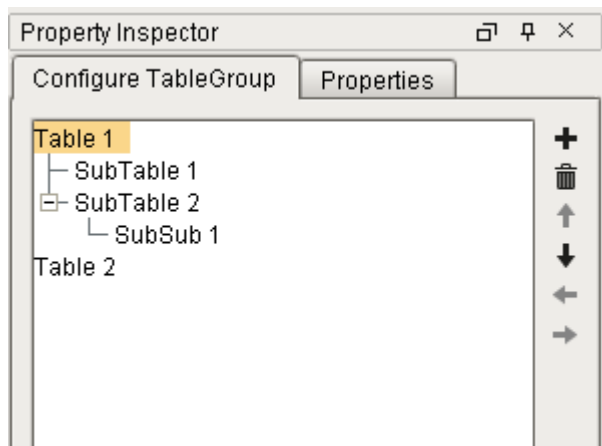
In the Configure TableGroup tab, we can modify the hierarchy of the Table Group using the **Arrow**  icons to the right of the table list. This allows us to change the order that the tables appear in, change a peer table into a child table, or change a child table into a peer table. If at some point, you realize

you have added too many tables, you can also use the **Delete**  icon to remove it. Be careful, as removing a table that has children will also remove the child tables.

Finally, if you ever want to cancel the Table Group click the **Delete Table Group**  icon to do so.

Child versus Peer

When adding tables to a Table Group, they can be added as either a "Child" or "Peer" table. These terms are always relative to adjacent tables in the group's hierarchy. Thus a table that is a child to one table may have some peers, as well as its own child tables. You can always tell the relationship between two tables by the indentation of each table.



Both Table 1 and Table 2 are peers. This is denoted by the matching indentation. In regard to the image above, the following statements are true:

- **SubTable 1** and **SubTable 2** are both children to **Table 1**, because they are indented over from their parent.
- **SubTable 1** and **SubTable 2** are also peers to each other, because they have a matching indentation.
- **SubSub 1** is a child to **SubTable 2**, and does not have any peer or children tables.

When the report is generated, the first row of **Table 1** will appear first. Each row of **Table 1** will generate an instance of **SubTable 1** and **SubTable 2**. Furthermore, each row of **SubTable 2** will generate an instance of **SubSub 1**. After all of **Table 1** has been represented on the report, **Table 2** will appear on the report.

The behavior of Peer and Child tables are further described below.

Peer Tables

Peer Tables are inserted one after another in the parent table. One table could potentially show details for each piece of equipment, with the peer table showing something completely unrelated starting at the end of the table before it. This helps keep the flow between two different datasources. Typically, each datasource would get its own table component. The issue with that is where to place them with how the tables automatically grow with more data. You could potentially have large amounts of blank space between the two tables because the first table only has a few records. This allows all the data to flow from one datasource to the next, even with unrelated datasources. You can see in the example below, the first image shows what a single individual table looks like, while the second image is a Table Group of two completely unrelated datasources stacked together. Each table inside the Table Group is also unique, with their own formatting, style, and even row versions.

Individual Datasource in Singular Table	Table Group with Two Different Datasources																										
<table><tr><th>Fruit</th><th>Quantity</th></tr><tr><td>Apples</td><td>60</td></tr><tr><td>Oranges</td><td>45</td></tr><tr><td>Melons</td><td>25</td></tr></table>	Fruit	Quantity	Apples	60	Oranges	45	Melons	25	<table><tr><th>Fruit</th><th>Quantity</th></tr><tr><td>Apples</td><td>60</td></tr><tr><td>Oranges</td><td>45</td></tr><tr><td>Melons</td><td>25</td></tr><tr><th>Customer</th><th>Order Quantity</th></tr><tr><td>John</td><td>47</td></tr><tr><td>Mary</td><td>82</td></tr><tr><td>Tom</td><td>14</td></tr><tr><td>Peter</td><td>31</td></tr></table>	Fruit	Quantity	Apples	60	Oranges	45	Melons	25	Customer	Order Quantity	John	47	Mary	82	Tom	14	Peter	31
Fruit	Quantity																										
Apples	60																										
Oranges	45																										
Melons	25																										
Fruit	Quantity																										
Apples	60																										
Oranges	45																										
Melons	25																										
Customer	Order Quantity																										
John	47																										
Mary	82																										
Tom	14																										
Peter	31																										

Child Tables

Child table groupings allow you to nest one table inside of another table. One table could potentially show details for each piece of equipment, but with a child table under each row, you can show all of the relevant downtime occurrences for that piece of equipment. These types of tables work well with [Nested Queries](#).

Using Child Tables - An Example

Creating child tables requires having a complex dataset. Go to the [Nested Queries](#) page and complete the Equipment Downtime example at the bottom if you want to use the same datasets for this example.

1. Once you have your Data set up, click on the Design tab and add a **Table** component to your report.

2. With the table created, add a **Header** row to the table. Drag all of your Equipment details into the table, and add header titles for each column. Equipment table column headers for this example:

- **Equipment ID**
- **Equipment Name**
- **Description**

The screenshot shows the Key Browser on the left and the Property Inspector on the right. The Key Browser has a tree view with 'Parameters', 'Data Sources', and 'Built In'. Under 'Data Sources', there is a folder named 'Equipment' containing 'EquipmentDescription', 'EquipmentIDNumber', and 'EquipmentName'. The Property Inspector has two tabs: 'Configure TableGroup' and 'Properties'. The 'Configure TableGroup' tab is active, showing a table with three columns: 'Equipment ID', 'Equipment Name', and 'Description'. The table is titled 'Equipment Downtime Details'. The 'Equipment' table is selected in the Key Browser, and its properties are shown in the Property Inspector. The 'Key' field is set to 'Equipment' and the 'Start on new page' checkbox is checked.

3. Now that we have a base table, click the **Make TableGroup** button at the bottom of the Property Inspector.

4. Once you make your TableGroup, you're ready to create a Child Table. In the **ConfigureTable Group** tab, make sure the Equipment table is selected and click the plus icon **+** and select '**Add child table.**' The table you create becomes a child of the Equipment table. By default, the child table will be called **Objects**.

5. A Child Table has its own datasource and uses a nested query to pull data from the database. Double click on the child table to select it and see the details for that table.

NOTE

You can switch back to the Configure TableGroup tab by clicking on the Table Group button **Table Group** in the lower left of the Table component.

6. Drag the child datasource (i.e., EquipDowntime) from the Key Browser to the **Key** field in the Configure TableGroup tab.

7. Next, add a header row with column name text:

- **Downtime Cause**
- **Downtime (Minutes)**

8. Drag your columns from the Key Browser to the appropriate columns in your table.

The screenshot shows the report design tool interface. On the left, the **Key Browser** pane displays a tree structure under **Data Sources** with **Equipment** expanded, showing fields like **EquipmentDescription**, **EquipmentIDNumber**, **EquipmentName**, **EquipDowntime** (highlighted), **DowntimeCause**, and **DowntimeMinutes**. Below it, the **Property Inspector** shows the **EquipDowntime** table selected, with the **Key** property set to **EquipDowntime** and a green checkmark indicating it is the primary key. The main design area on the right shows a report titled **Equipment Downtime Details** with a table structure. The table has a header row with **Downtime Cause** and **Downtime (Minutes)** columns, and a body row with **@DowntimeCause@** and **@DowntimeMinutes@** placeholders. The table is labeled **EquipDowntime Details** and is part of a **Table Group**.

9. Go to the **Preview** panel to view the report. You'll notice the Child Table is now embedded in the parent table.

Equipment Downtime Details

Equipment ID	Equipment Name	Description
25489	Conveyor Line	Transfers product
	Downtime Cause	Downtime (Minutes)
	Backup	22
	Scheduled Maintenance	12
55684	Labeler	Makes labels
	Downtime Cause	Downtime (Minutes)
	Out of labels	25
	Stuck labels	15
88456	Palletizer	Makes pallets
	Downtime Cause	Downtime (Minutes)
	Misalignment	38
	Misalignment	55
626145	Filler	Fills tanks
	Downtime Cause	Downtime (Minutes)
	Overflow	50
	Scheduled Maintenance	40

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Images in Reports

You can spice up your reports by adding images, whether it's your corporate logo, or embedding icons in tables to help users find data quickly. Images can be added to reports in several different ways:

- Drag and drop an image from your desktop/computer to the report - No data key is required.
- Drag an image from the Image Management tool to a report - No data key is required.
- Drag an Image component from the Report Design Palette to a report. This option requires that you enter a path from wherever the image is located (i.e., local drive, shared drive, or webpage) to the image in the Key property. You can either manually type in the path or reference a parameter, thus, making it static or dynamic.

Data Keys can help you make the images in reports more dynamic. By dragging a parameter or a datasource column from the Key Browser into the Key property of the Property Inspector, a data Key can resolve to a byte array, such as an image retrieved from a database, or a URL that points to an image on a webpage, or an image file stored on a shared drive. Reports are generated on the Gateway so the image source needs to be accessible from the Gateway computer.

Keychain Expressions can not be used in the data Key, but the Key can be a parameter or Data Source that dynamically constructs a path to an image. This allows you to easily change images in reports without changing the actual image component.

! INFO

Whether using a parameter or typing in a path name in the Key property, always use double quotes around the path name.

Before we dive-in to show you how to configure the paths for your images, please read the Notes contained in this section carefully because path names require a specific format. There are also a few helpful hints about configuring images in reports.

Image Formats

The Report Module supports the following image file formats:

- .gif
- .tiff
- .jpeg
- .jpg
- .png
- .bmp
- .pdf

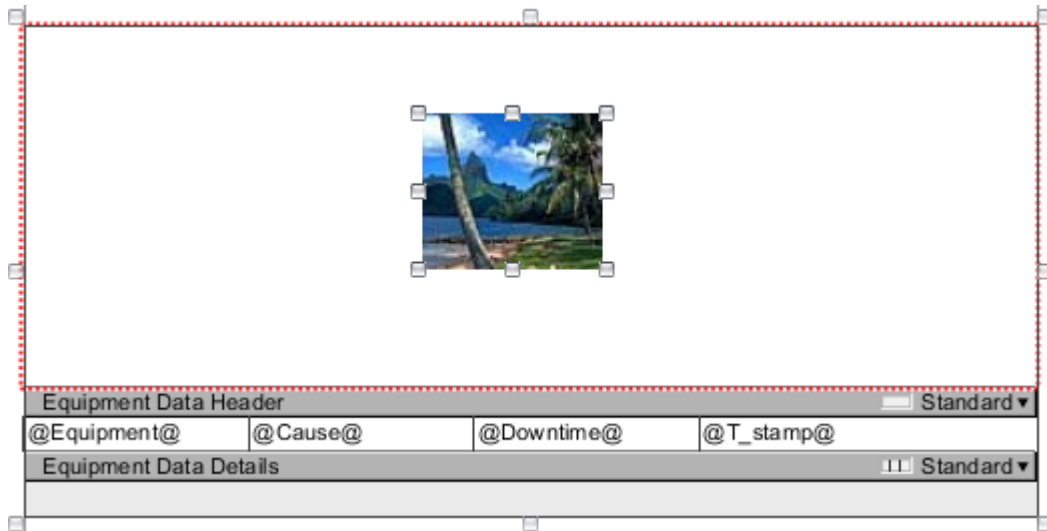
Drag and Drop Images

You can use the drag and drop feature to drag images from a local computer, shared drive, webpage, or the Image Management Tool to add images on a report. The report will display blue highlights around your report when you drop your image onto the report. No data key is required, and the dropped image will be used. Below, you can see a table header highlighted to drop an image in.



Equipment Data Header			
@Equipment@	@Cause@	@Downtime@	@T_stamp@
Equipment Data Details			

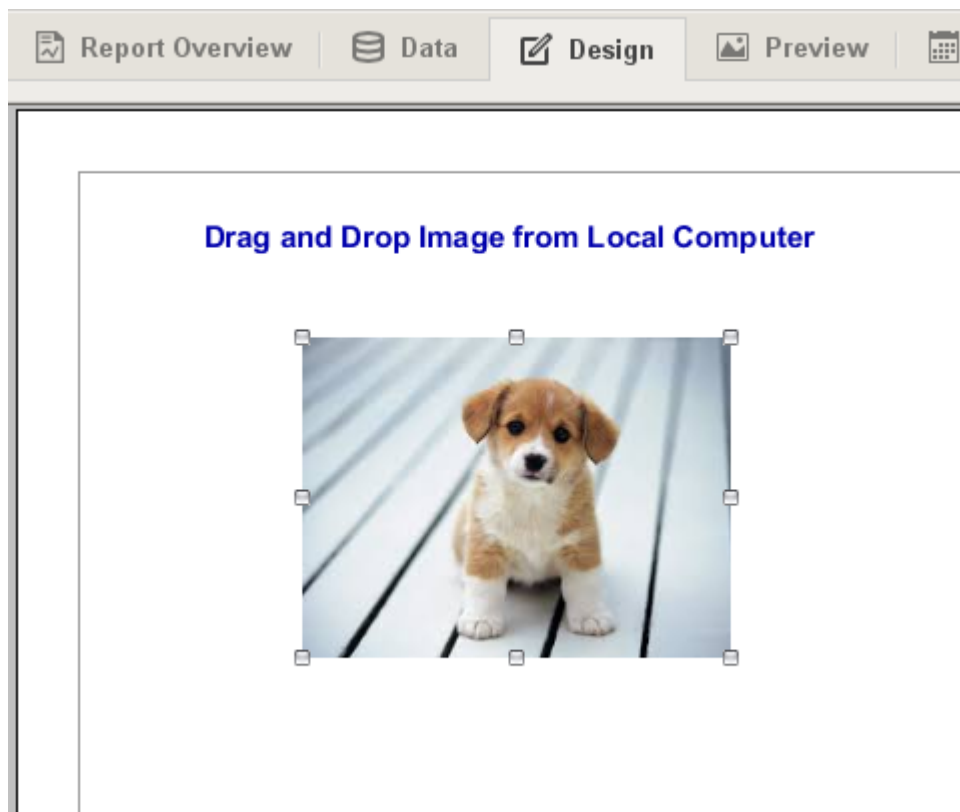
If you associate a key with an image component, the original placeholder image (as shown below) will still be seen in the Design panel, but will be replaced with the image associated with the data Key in the Preview panel. If the key does not link to a valid image, no image will be shown in the Preview panel.



Drag and Drop an Image from a Desktop

If you have a collection of images stored on your local computer, you can easily drag and drop any image into a report. No Key is required. You'll notice the **Key** property is blank. This doesn't mean you can't associate a Key with an image component. You most definitely can! The image will be replaced with the image associated with the Key when the report is generated. If you're going to use images from a local computer, it's a good idea to import those images to the Image Management Tool to make them visible and available to all project users.

To drag and drop images into a report from your desktop, head to the **Design Panel**. Find an image on your local computer, and drag the image directly onto the report. A new Image component will be created, and will be filled with your selected image. Note that the resulting Image component's Key property will be blank. This is expected behavior, as the image is **embedded directly on the page**, and not dependent on a key or file path.

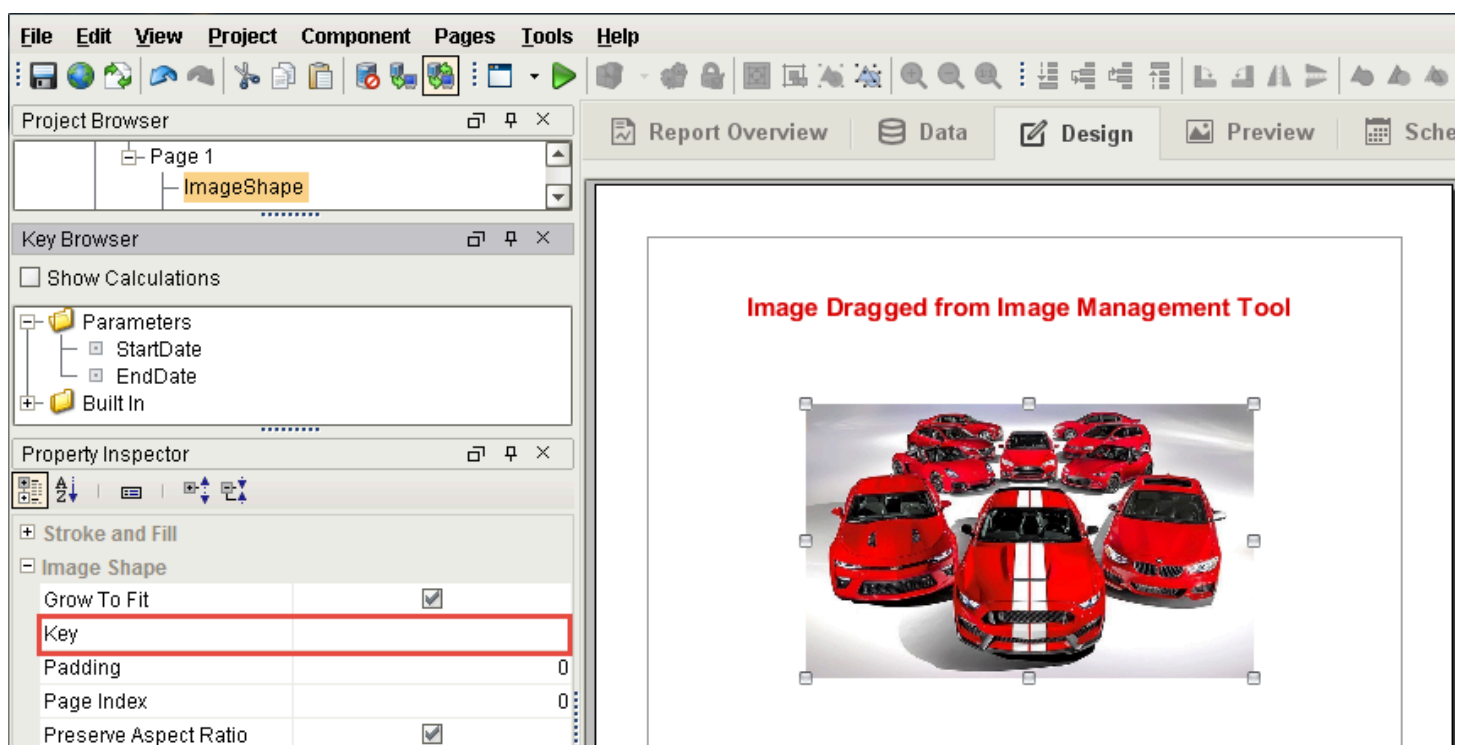


Using the Image Management Tool

Images may also be dragged and dropped directly from the **Image Management tool** (**Tools** menu in the Designer > **Image Management**).

Go to your report and open the **Design** panel. Drag an image from the Image Management tool to your report. You'll notice that the Key property is blank. This is expected behavior.

Your image will be displayed directly in both the Design and the Preview panels.



The Key Property

Using the Image component is one way of adding an image in a report. You can make the image path static or dynamic by entering the path name manually, or referencing a parameter. The following sections describe how to setup the image path to use both static and dynamic data keys.

NOTE

- Python is used to handle expressions when creating new parameters. Because the backslash has a special meaning in Python, formatting a path name for local drives and shared drives is a little different. Each backslash must be represented by two backslashes, for example, place four backslashes at the beginning of a network path and two backslashes between folders in the path. Here are a few examples;
 - Accessing a local drive - "\\C:\\Images\\veggies.jpg"
 - Accessing a shared drive - "\\ComputerName\\Folder\\Images\\image.png"
- URLs from a webpage, use a forward slash, so use the link as is - "<http://i.stack.imgur.com/WCveg.jpg>"

Static File Path

A static file path is created by dragging an Image component from the Report Design Palette into the Design panel, and manually typing the image path name into the Key property. The image path can point to a drive on a user's local machine, shared drive, or webpage. In this example, the image path is pointing to a user's local machine.

INFO

To paste a link into the Key property, use the **Ctrl V** command. Right clicking with your mouse to paste a link in the Key property is not an available function.

In the **Key** property, enter the path name where the image is located. You can type it in manually, or paste (**Ctrl-V**) the path name in the Key property making sure you have double quotes around the path name.

NOTE

The path name uses double backslashes separating the drive and folders (i.e., "**C:\\Images\\Images\\veggies.jpg**").

The placeholder image will be seen in the Design panel, and the image associated with the Key property will be displayed in the Preview panel or when the report is executed.

Property Inspector

Stroke and Fill

Image Shape

Grow To Fit

Key

Padding

Page Index

Preserve Aspect Ratio

Report Overview

Data

Design

Preview

Schedule

Placeholder Image

	Vegetable	Qty	
Vegetables Header			
	@Vegetable@	@QTY@	
Vegetables Details			

Go to the **Preview** panel and you'll see the placeholder image was replaced by the image associated with the Key property.

Image Associated with Key Property

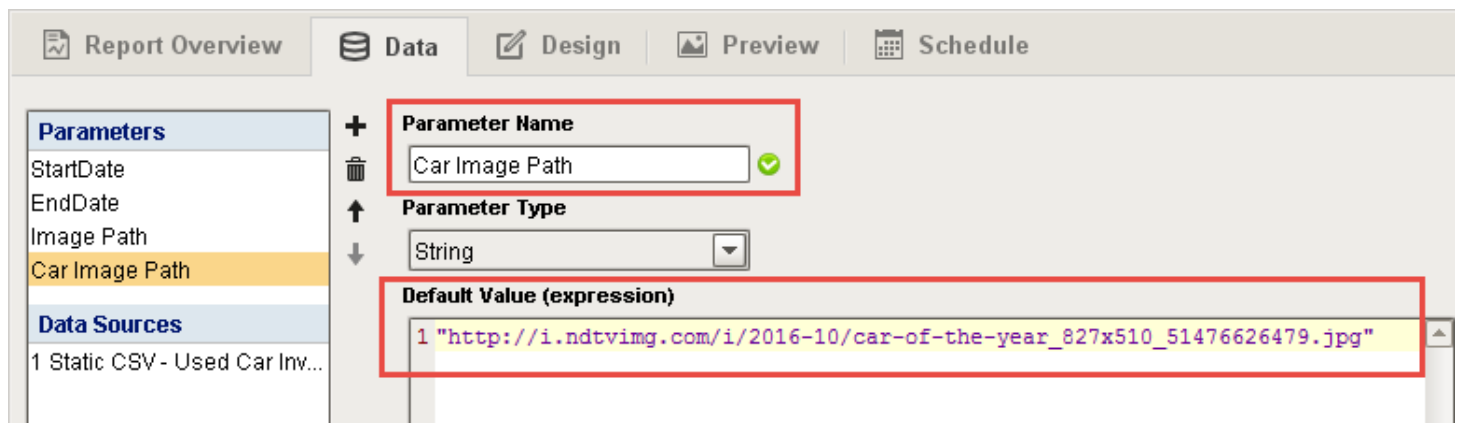
Vegetable	Qty
Squash	10
Zucchini	18
Carrots	25
Potatoes	8
Beans	33

Dynamic Key Property

The Key property can be a parameter that dynamically constructs a path to an image which allows you to easily change images in reports without changing the actual image component. You can make an image path dynamic by dragging an Image component from the Report Design Palette into the Design panel, and creating a parameter for the image path. The expression for the Image Path parameter can be a path to a local drive, shared drive, or webpage.

Note that reports **always** execute on the Gateway, so the file path is always relative to the Gateway.

To create a parameter, go to the Data panel, and click the plus **+** icon to add a parameter. In the expression block, enter in a link from a webpage using double quotes around the path name.



Go to the **Design Panel**, click on the placeholder image, and expand the **Parameters** folder in the **Key Browser**. Drag your new parameter (i.e., Car Image Path) into the **Key** property of the **Property Inspector**. The image associated with the parameter in the Key property will be displayed in the Preview panel or when the report is executed.

Key Browser

Show Calculations

Parameters

Data Sources

Used Car Inventory

Cars

QTY

Built In

Property Inspector

Stroke and Fill

Image Shape

Grow To Fit

Key

Car Image Path

Padding

Page Index

Preserve Aspect Ra...

Rectangle Shape

Basic Properties

Report Overview

Data

Design

Preview

Schedule

Used Car Inventory Report

August 2017



	Make	Quantity	
Used Car Inventory Header			
	@Cars@	@QTY@	
Used Car Inventory Details			

Go to the **Preview** panel and you'll see the placeholder image was replaced by the image associated with the parameter in the Key property.

Used Car Inventory Report

August 2017



Make	Quantity
Ford	13
Toyota	22
Mercedes	2
Buick	6
Lexus	7
Jeep	11

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

CrossTab and Simple Tables

In the other sections of the Report Table, we talked about creating tables using the Table component, commonly referred to as Standard Tables. Standard Tables focus on a dynamic number of rows with static columns, however, there are other table types that behave a little differently. These other table types are the Simple Table and CrossTab Table.

The CrossTab Table, although similar to standard tables and charts, are commonly used to summarize the relationship between two categories of data by showing summaries of cross sections of the datasource. The Simple Table allows you to easily create a grid-like table structure with the rows and columns being dynamic. Both table types are described below.

WORKING WITH SIMPLE AND CROSSTAB TABLES

The examples on this page assume you already have datasources created. If not, you can copy the data from the Code Blocks in each example to create your own datasources to be used with the following Simple Table and CrossTab Table examples.

CrossTab Table



INDUCTIVE
UNIVERSITY

CrossTab Table

[Watch the video](#)

The Cross-Tabulation or CrossTab component is a tabular data element like the [Table](#) and [Charts](#). Also known as Contingency Tables, they are commonly used to summarize the relationship between two categories of data by showing summaries of cross sections of the data source. To be useful, crosstab data should have the following:

- Lots of repetitious data. Sums, Averages, and other Aggregating functions are well represented by CrossTab Tables.

- A data source that provides at least two columns of data which are repetitious compared to the number of rows.
- One or more columns that represent a value that requires calculation. Examples are: summing money, displaying average response times, counting occurrences, etc. These calculations may be provided as columns or calculations of the data source, or as [Keychain Expressions](#).

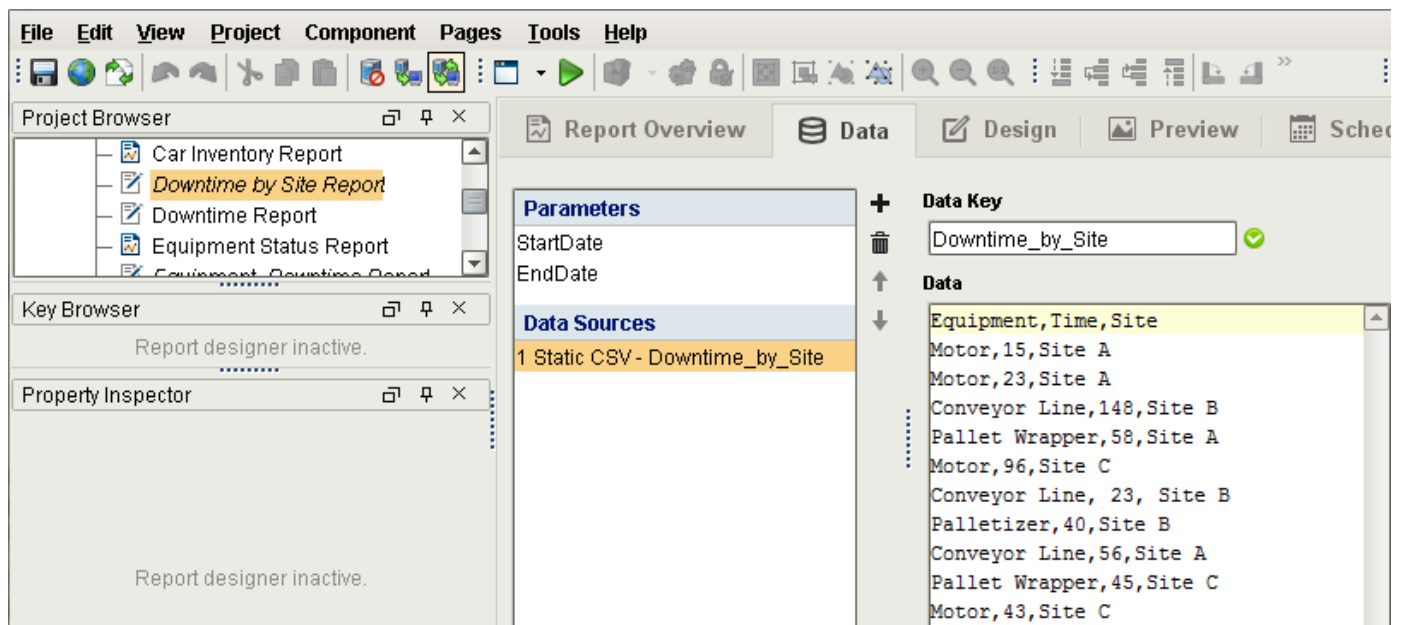
The CrossTab component is much simpler than the Table component. By default, it just shows a single cell. This is usually configured with an aggregate key, like "@total.getAmount@". After that, grouping keys are dragged to the horizontal and vertical axis.

The objective of this CrossTab example is to show the number downtime time events by site and equipment. Let's give it a try!

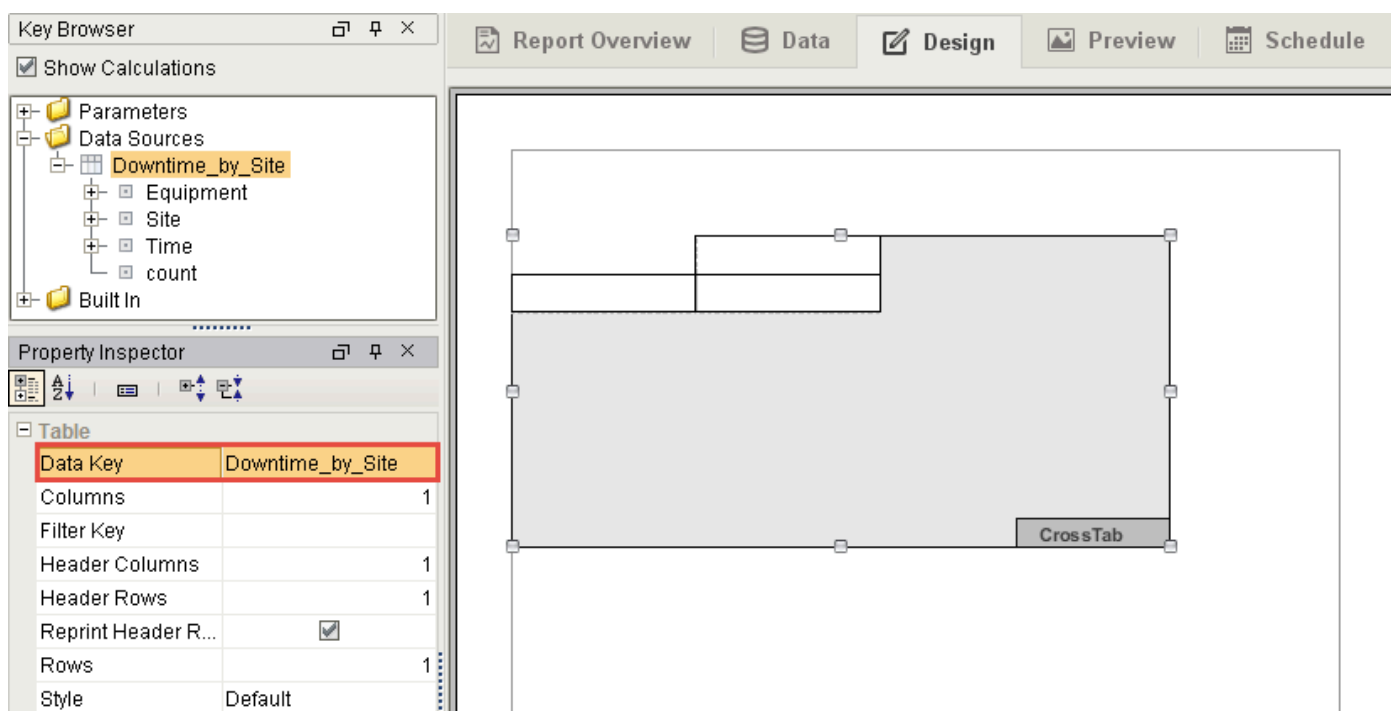
1. In the **Data** panel, create a CSV data source named **Downtime_by_Site**. You can copy the data from the Code Block below to create your own CSV Data Source.

Downtime_by_Site

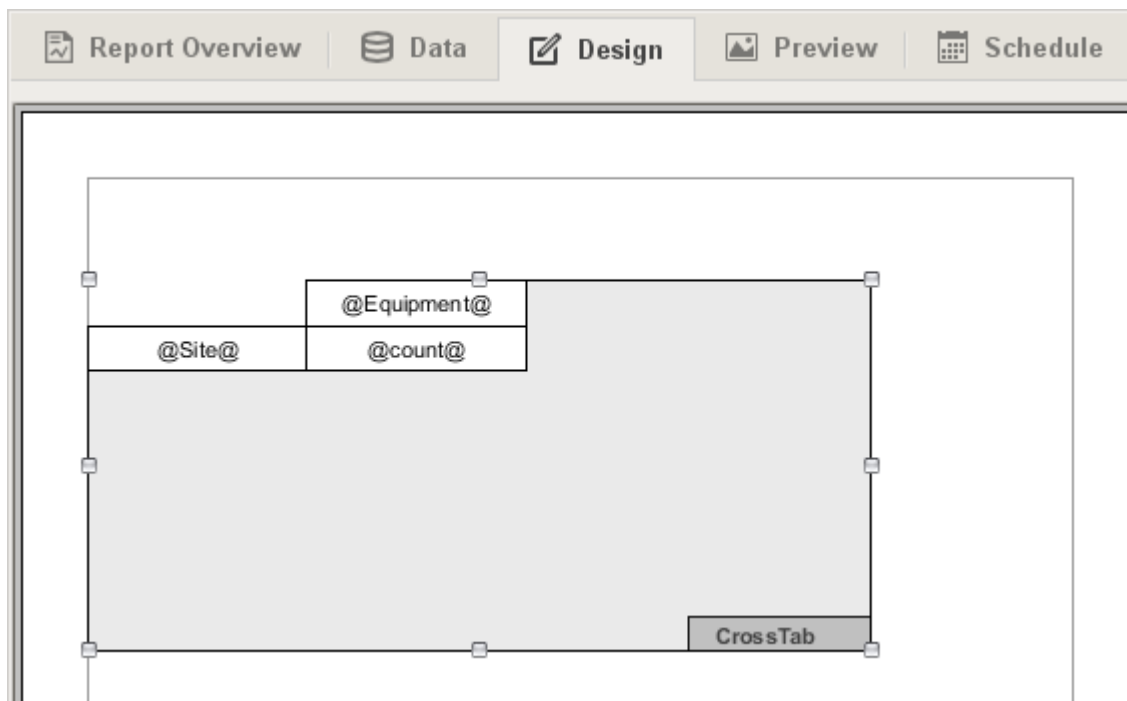
```
Downtime_By_Site
Equipment,Time,Site
Motor,15,Site A
Motor,23,Site A
Conveyor Line,148,Site B
Pallet Wrapper,58,Site A
Motor,96,Site C
Conveyor Line, 23, Site B
Palletizer,40,Site B
Conveyor Line,56,Site A
Pallet Wrapper,45,Site C
Motor,43,Site C
```



2. In the **Design** panel, drag a **CrossTab** component  to your report.
3. With the CrossTab component still selected, drag-and-drop your Data Key (i.e., Downtime by Site) from the **Key Browser** to the **Data Key** property in the **Property Inspector**.



4. In the **Key Browser**, set the **Show Calculations** property to **true**.
5. Next, add some data keys to the cells in the CrossTab Table component. From the **Key Browser**, drag the '@Equipment@' to the top cell, '@Site@' to the leftmost cell, and '@count@' to the remaining cell. This will show the downtime count of each piece of equipment by site.



6. Switch to the **Preview** panel and your CrossTab Table will display the results.

	Motor	Conveyor Line	Pallet Wrapper	Palletizer
Site A	2	1	1	0
Site B	0	2	0	1
Site C	2	0	1	0

Simple Table



INDUCTIVE

UNIVERSITY

Simple Table

[Watch the video](#)

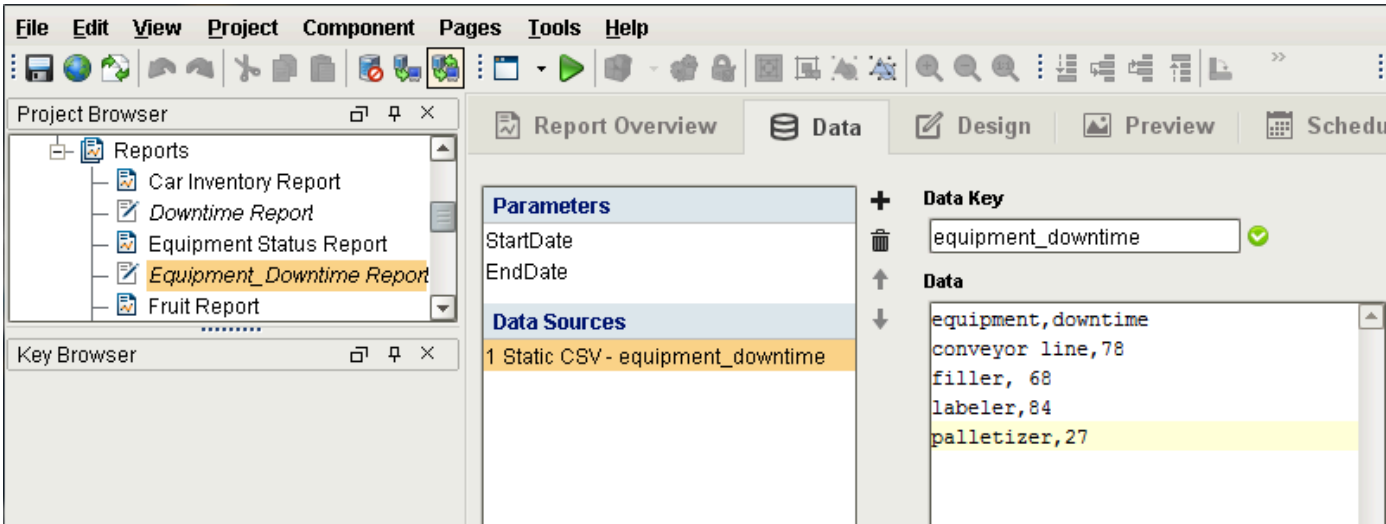
The Simple Table is a grid-like table structure that dynamically creates new rows and columns for rows returned by the Data Keys on the component. With the Simple Table you can very quickly add a table inside a report.

The objective of this Simple Table example is to show each piece of equipment including the total amount of downtime, occurrences, and the average duration of each occurrence, broken down by each piece of equipment.

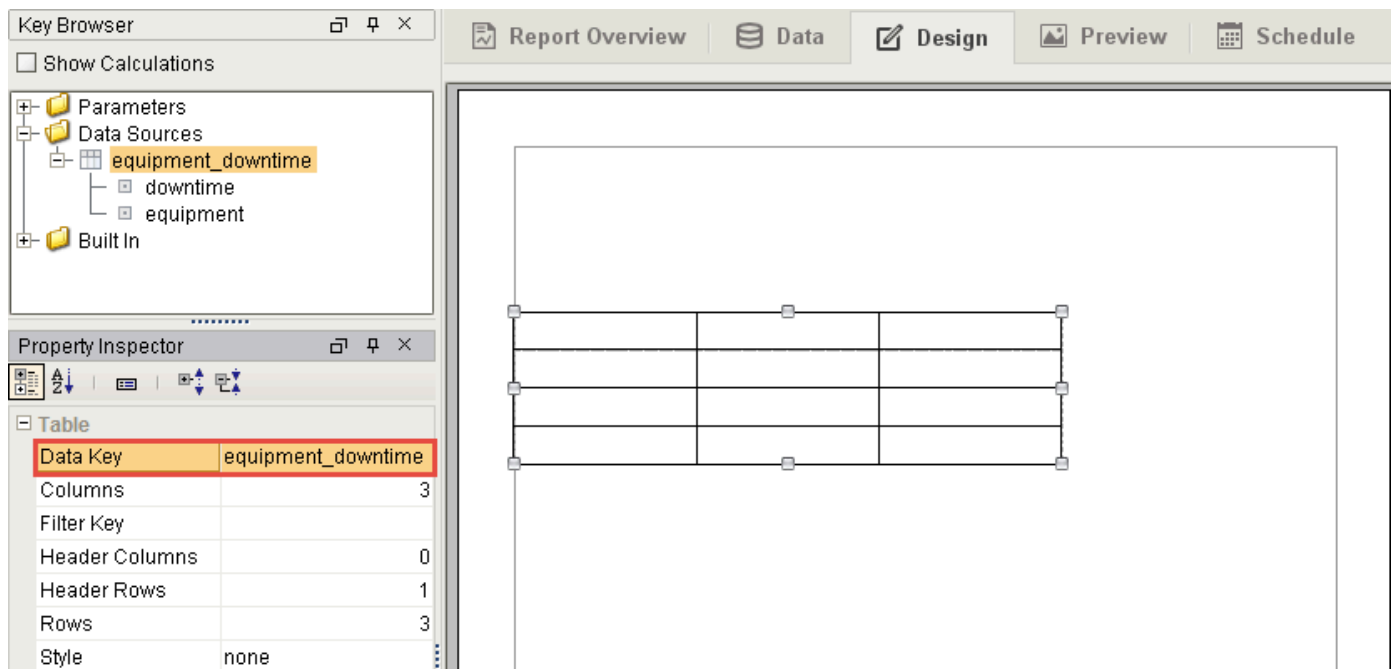
1. In the **Data** panel, create a CSV data source named **equipment_downtime**. You can copy the data from the Code Block below to create your own CSV Data Source.

```
equipment_downtime

equipment_downtime
equipment,downtime
conveyor line,78
filler, 68
labeler,84
palletizer,27
```



2. In the **Design** panel, drag a **Simple Table** component  to your report.
3. With the Simple Table component selected, drag and drop your Data key (i.e, equipment_downtime) from the **Key Browser** to the **Data Key** property of the **Property Inspector**.

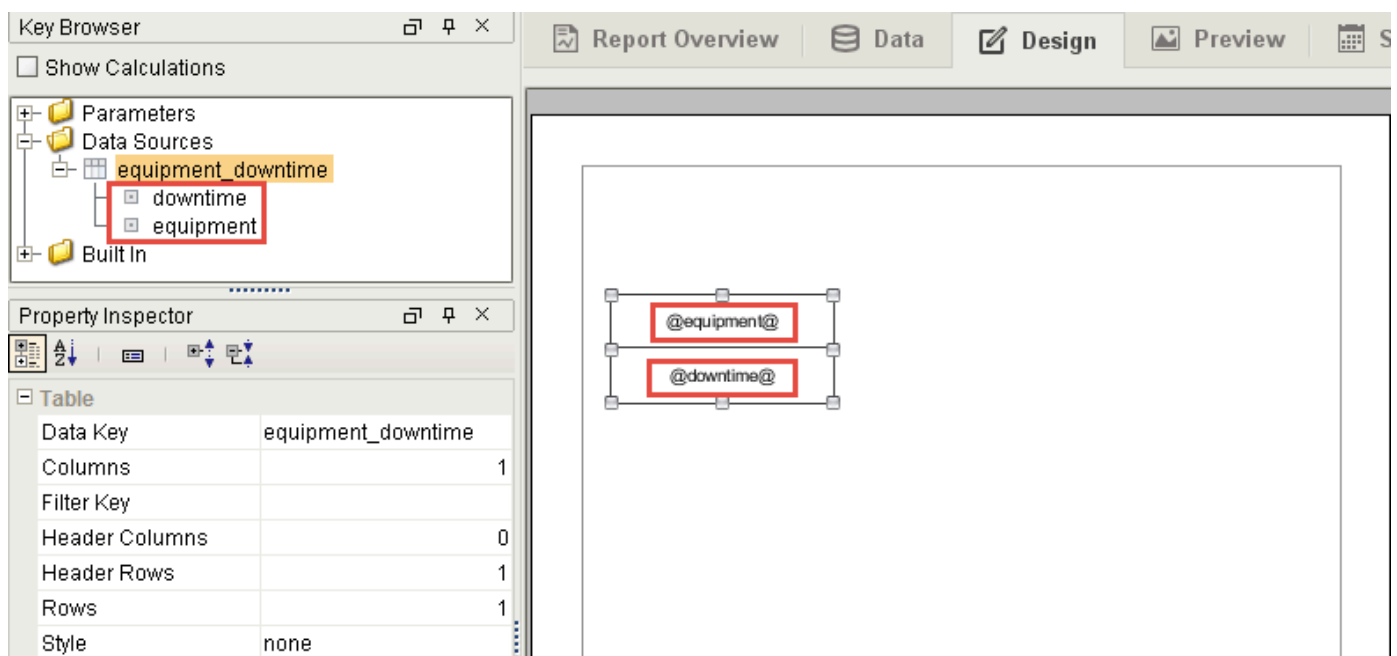


- From the **Key Browser**, drag the '@equipment@' key to the top row of the Simple Table, and the '@downtime@' key to the second row.

HEADER ROW

It's important to note that the top row of the Simple Table is a **Header Row**. In the Property Inspector, there is one Header Row, but you can always add more rows by changing the value, or even setting it to '0' to remove the Header Row.

- In the **Property Inspector**, change the number of **Rows** and **Columns** to 1. If you need to add or remove columns or rows, you can always change the Column and Row property values. Because the data keys determine how many columns and rows get created in your table, the Simple Table will add a new column for each value in the 'equipment' key, which will be visible on the **Preview** panel.



6. Next, switch to the **Preview** panel to check out your report. You can always go back to the **Design** panel to edit your table and update your report format.

Report Overview

Data

Design

Preview

Schedule

Simple Table

conveyor line	filler	labeler	palletizer
78	68	84	27

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Common Reporting Tasks

This section contains examples for items we've identified as "common" tasks: undertakings that many users are looking to utilize when first starting out with a specific module or feature in Ignition.

Additionally, this section aims to demystify some of the more complex or abstract tasks that our users may encounter.

The examples in this section are self-contained explanations that may touch upon many other areas of Ignition. While they are typically focused on a single goal or end result, they can easily be expanded or modified after the fact. In essence, they serve as a great starting point for users new to Ignition, as well as experienced users that need to get acquainted with a new or unfamiliar feature.

Below is a list of common tasks related to this section of the manual.

Report Workflow

Creating a sample report from start to finish. The [Reports Workflow Tutorial](#) will take you through all the different steps on how to bring in data, design a page, and schedule your report. This is a great place to start if you have used the Report module in the past and need a refresher, or if you are new to it and prefer concrete examples over the normal reference style pages of the manual.

Generate Barcodes

Creating sheets of product labels. The report module can quickly and easily generate [Labels with Embedded Barcodes](#). Using one of the special components in the Report Designer components list, you can easily set up standard sizes of paper to print multiple labels from a dataset and even include scan-able barcodes and QR codes. Once printed, these labels can be used for shipping, product identification, or tracking.

Converting Legacy Reports

Convert old reports to the current format. The report module was updated in 7.8 and any reports older than that were designed in a different way. The current report module has a built in converter to **Convert Legacy Reports** to the new reporting format so you can start scheduling those reports without recreating them from scratch. This example demonstrates the simple conversion process to get you started in the "new" format.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Tutorial - The Report Workflow



INDUCTIVE
UNIVERSITY

Simple Report

[Watch the video](#)

Steps to Amazing Reports

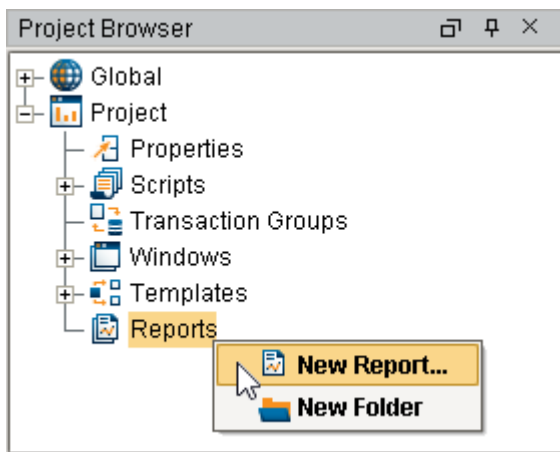
There are three fundamental steps to creating Reports:

1. Gather Data
2. Design the Report
3. View or Schedule Delivery

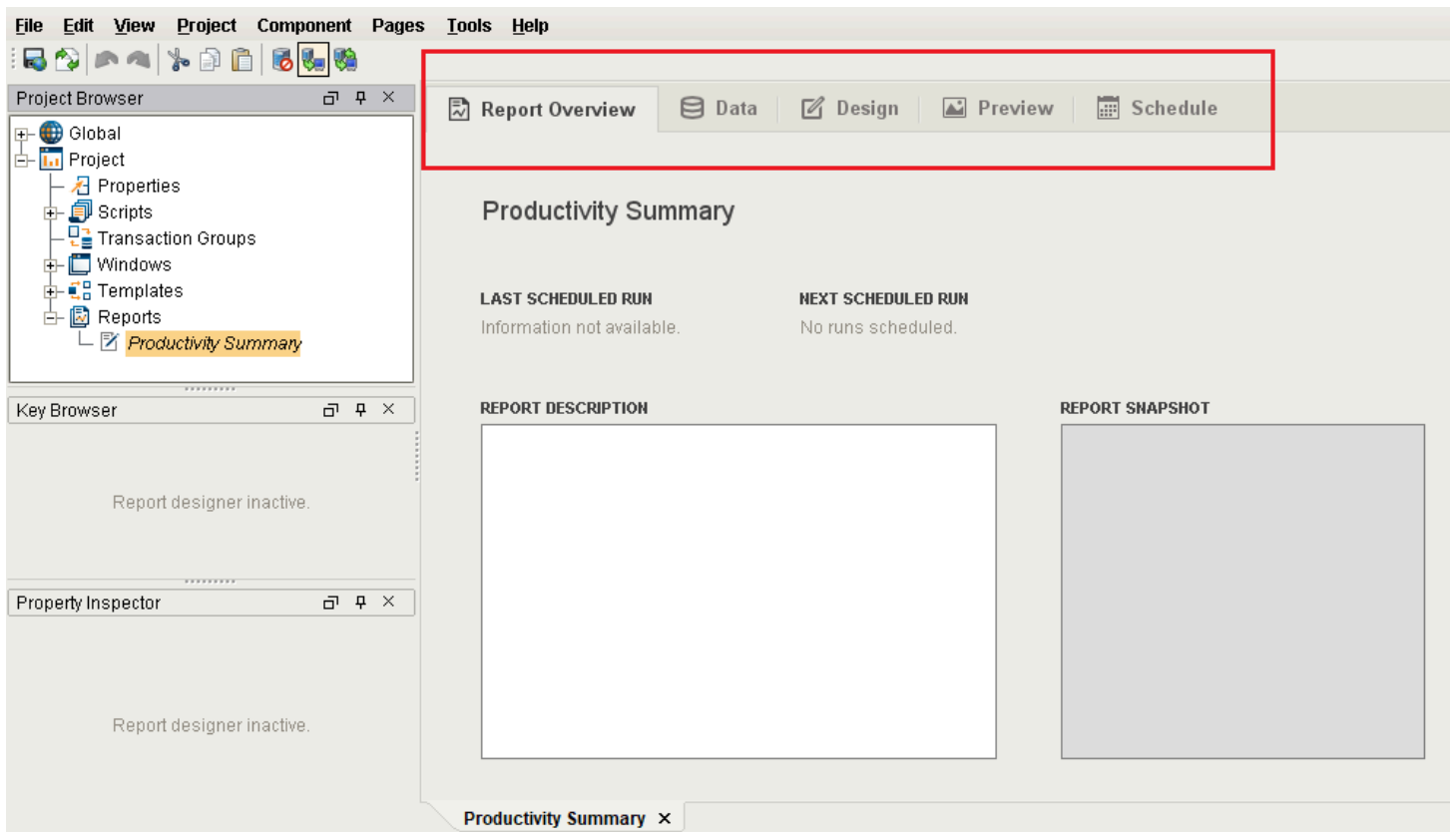
Simple, right? This workflow is really the essence of what it means to create a report in the Reporting Module. Conveniently, the [Report Workspace and Workflow](#) interface makes it simple to complete each step and view your result. This tutorial provides a walkthrough of creating a simple report, and reinforce some of the concepts and terminology involved. If you are already familiar with the basics of how to create a Report, you may be interested in the [Reporting Reference in the Appendix](#).

Creating Your First Report

To create your first report, you will need the [Reporting Module](#) installed. Once installed, launch the [Ignition Designer](#), and let's get started! To create a new report, right click on the Reports node in the **Project Browser**, select **New Report**, and enter a name for your report.



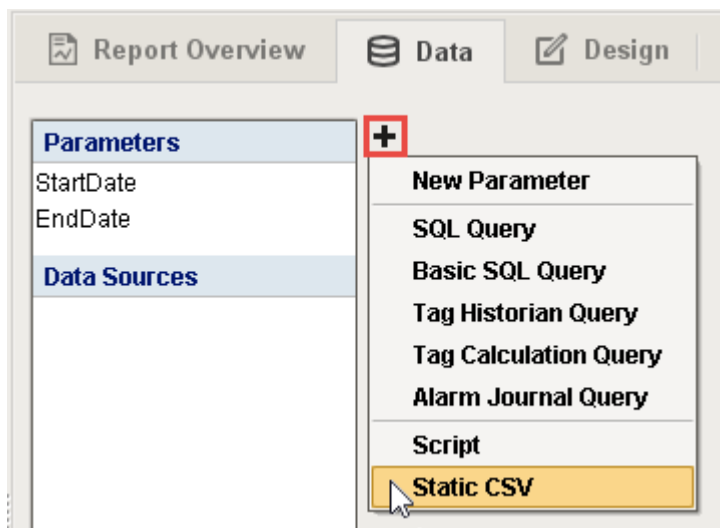
The **Report Workspace** will open, and you'll be in the **Report Overview Panel**. The top of the workspace has a series of stylized panels which guide you through the creation of a new report. It's no coincidence that our Report Workspace is laid out in such a way that lets you intuitively follow the three fundamental steps to creating reports with a little help along the way: gathering data, designing reports, and setting up schedules to run and distribute reports.



Setting Up Data Sources

For any Report, you will first need to specify one or more Data Sources in the **Data panel**.

1. Click the plus icon **+** to add a simple **Static CSV** Data Source, but if you would like to use data from a database connection, feel free to add a **SQL Query data source** instead.

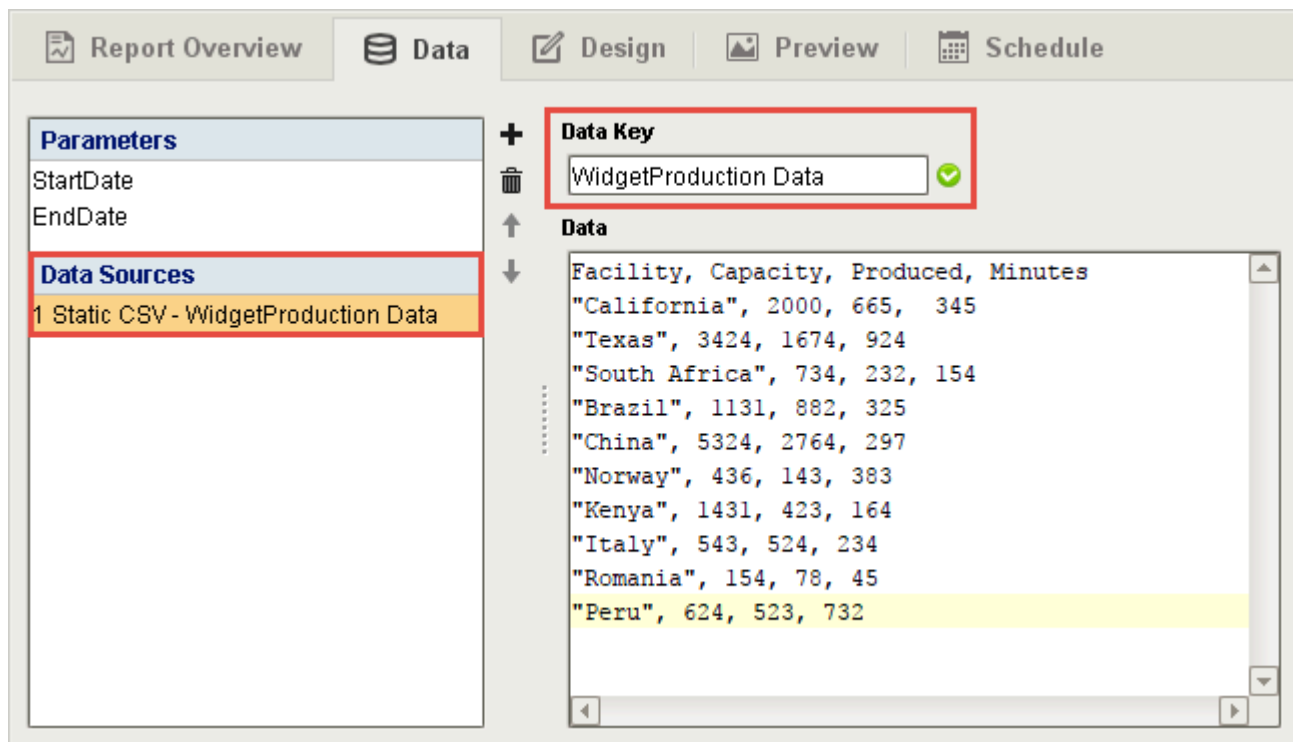


2. With the **Static CSV** Data Source editor open, copy and paste the CSV data below into your Data Source editor, and give your data a meaningful name in the **Data Key** field: we've named ours **WidgetProduction Data**. This example models data collected from all the International Widget Factories, complete with production capacity, number of widgets produced, and the number of minutes it took to produce them.

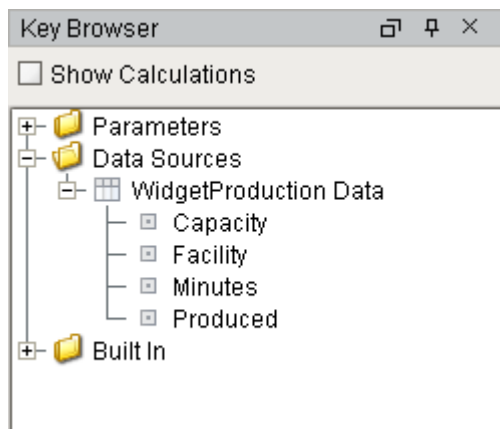
Static CSV Data Source

```
Facility, Capacity, Produced, Minutes
"California", 2000, 665, 345
"Texas", 3424, 1674, 924
"South Africa", 734, 232, 154
"Brazil", 1131, 882, 325
"China", 5324, 2764, 297
"Norway", 436, 143, 383
"Kenya", 1431, 423, 164
"Italy", 543, 524, 234
"Romania", 154, 78, 45
"Peru", 624, 523, 732
```

Here is what the data looks like in the Static CSV Data Source editor:



- Next go to the **Design panel**, and expand the **Data Sources** folder in the **Key Browser**. You'll see that each column of data in the Static CSV data source is represented by its own Data Key. These Data Keys are automatically generated based on the table of data fed to the report system. A key may have subkeys, or 'child-keys' Accessing the data from a child-key is accomplished using the path of a key in relation to its parent, sometimes referred to as A Keychain is simply the path to your data key using 'dot notation'. For instance, the Keychain dragged into a report for our **Facility** key would be `@WidgetProduction Data.Facility@`.



💡 THE POWER OF DATA KEYS

You can have multiple keys in any text field. In addition, you can use multiple keys and common numeric operators to do calculations within the @ symbols! These are called Keychain Expressions. Expressions are not something this introductory tutorial will cover, but you can head over to the data key documentation page for more information.

Now that you have some data, let's create a simple design and see how the Data Keys can be used to create dynamic reports.

Design the Report

The Design Panel is where we start building the report. This section of the report has many components, such as tables and charts, that can display the results of your data sources and parameters. Components in a report are typically assigned one or more keys, which then feeds the results of our data sources and parameters directly to the component. To create comprehensive reports, you will likely find yourself combining different components. Fortunately, this is pretty easy to do using the visual Designer, so let's continue with your first report and see how easy it is.

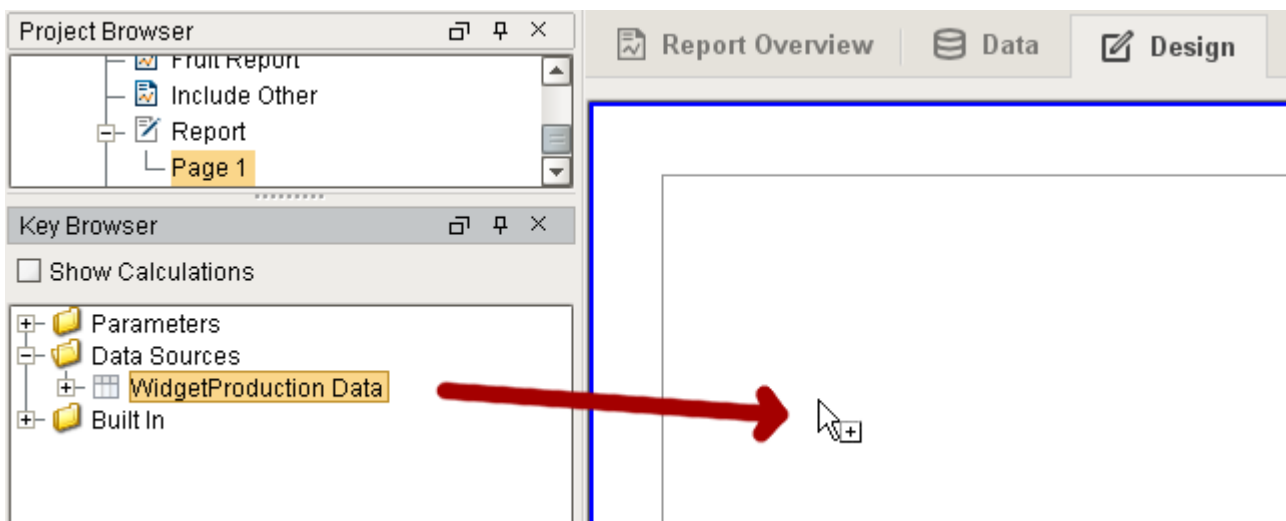
For this report, we will aim for the following requirements:

1. A **Table** that gives a summary of how many widgets each factory produced, and the totals for each data column.
2. Factory efficiency on a units/minute basis which needs to be calculated.
3. A Header / Title for the report.
4. Page numbers in case the report gets to be too long.
5. A **Bar Chart** that visualizes widget production. This will use a separate data source that summarizes the Table's data.

Adding a Table

Let's add a table and Datasource.

1. In the **Design panel**, drag and drop the **WidgetProduction Data** key from the Key Browser onto the page. When you let go of the mouse button, this will create a Table component.

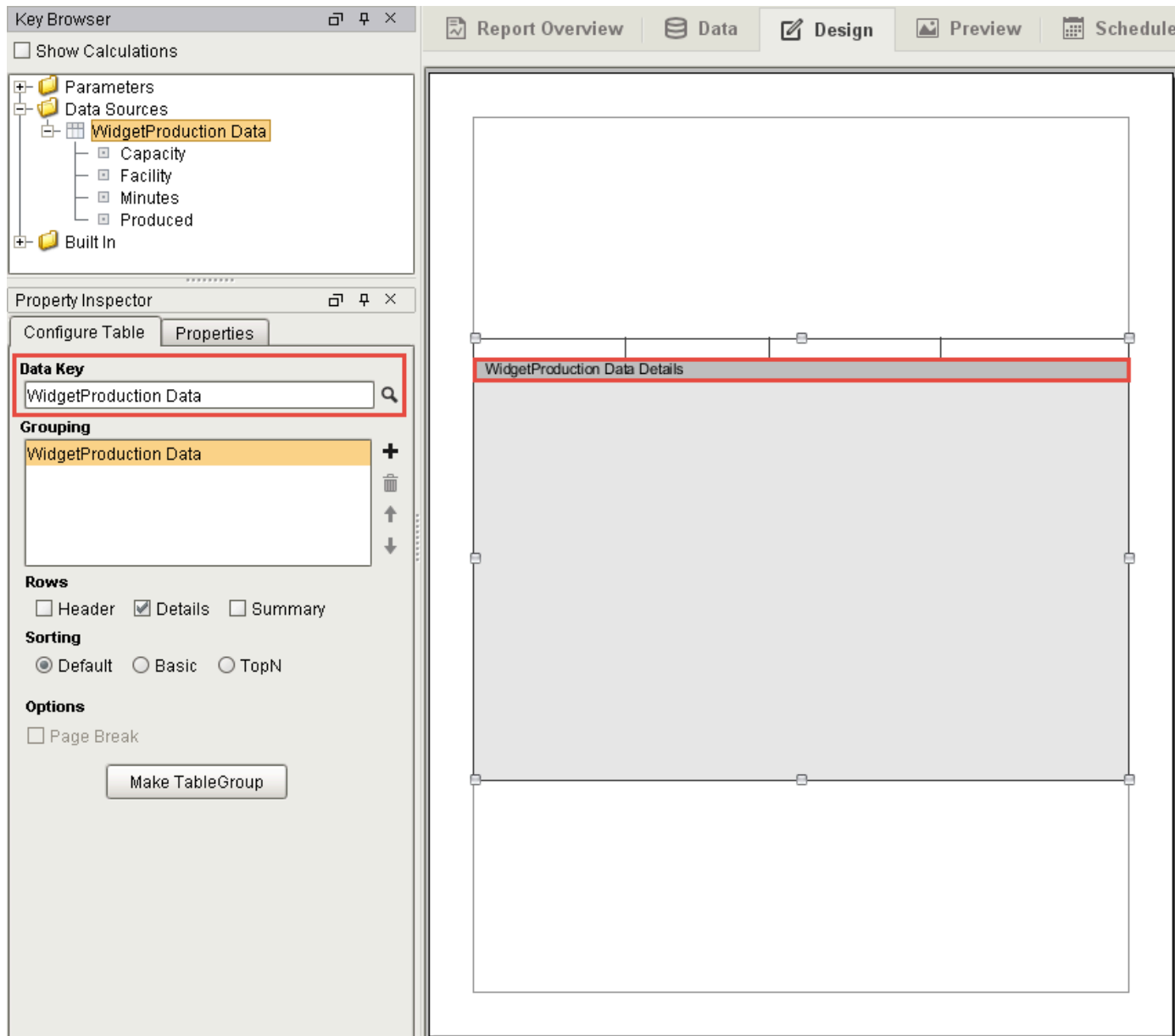


2. With the Table selected, in the bottom left corner of the **Property Inspector**, you will see the **Configure Table** tab. Note that the Data Key property on the table was automatically assigned to our **WidgetProduction Data** data source. Additionally, you'll see the Details Row (the dark gray

bar on the table component) displays the name of our data source. This means that the table was property assigned a data source.

TIP

Alternatively, you may drag a Table component from the Component Palette on the right side of the interface. If you take this approach, you must manually assign a data source by interacting with the **Data Key** property on the **Configure Table** tab: click the **search**  icon next to the Data Key property, and select the **WidgetProduction Data** data source. It should look like the image below.



Configuring Data for your Report

Now, let's add some keys to the table.

1. In the **Design panel**, drag and drop data keys from the **Key Browser** (i.e., Facility, Capacity, Minutes, Produced) into the **Details** columns. You'll notice that the keys you dropped are

surrounded by '@' symbols. The @ symbols tell the report engine that text inside is a key, and it should try and find the key's value when the report is generated.

- Next, add a header for each of the columns by enabling the **Header** checkbox under the **Configure Table** tab of the Property Inspector. Next, select each header column, type in your header name, and if you like, you can make it stand out by bolding the text in the **Text Editor** pane.
- Go to the **Preview panel** to check the data in your report.

The screenshot displays two panels from a report generation tool. The left panel is the 'Design' view, showing a table structure with a header row and a details row. The header row contains four columns: 'Facility', 'Capacity (Units)', 'Run Time (Min)', and 'Units Produced'. The details row contains four columns: '@Facility@', '@Capacity@', '@Minutes@', and '@Produced@'. The right panel is the 'Preview' view, showing the rendered table with data for various facilities.

Facility	Capacity (Units)	Run Time (Min)	Units Produced
California	2000	345	665
Texas	3424	924	1674
South Africa	734	154	232
Brazil	1131	325	882
China	5324	297	2764
Norway	436	383	143
Kenya	1431	164	423
Italy	543	234	524
Romania	154	45	78
Peru	624	732	523

Let's take it a step further. Say you would like to calculate the efficiency of units produced per minute of production runtime. You can use the **keychain expression** `@Produced/Minutes@`, which will return the value of the Produced key divided by the value of the Minutes key. Note that this calculation is executed for each row, so it is always taking the current rows values to determine the quotient. To do this, you need to add another column in the Details Row and Header Row, using the following steps:

- With the Table selected, double click the **Details Row** to select it. This opens the **Property Inspector** containing all the Details properties. Under the **Properties tab**, change the **Column Count** property from '4' to '5'. The new column will be added on the right side of your table. You may have to adjust your column widths to see it.
- Click on the new cell and enter '`@Produced/Minutes@`' (Refer to the screenshot in the Design panel below).
- Double click the **Header Row**, and add a column by changing the **Column Count** property from '4' to '5'.
- In the **Table**, adjust your column widths, and enter '**Efficiency - Units/Min**' in the new Header column.

Lastly, let's add a total for the number of widgets produced from all facilities to your report using the following steps:

1. With the **Table** selected, mark the Summary checkbox in the Configure Table tab of the Property Inspector. A new Summary row will appear.
2. Double click on the new **Summary Row**, and under the **Properties** tab, change the **Column Count** property from '4' to '2'.
3. In the **Key Browser**, set the **Show Calculations** property to 'true' making the Built-in keys available.
4. Expand the Datasources → WidgetProduction Data → Produced object. Drag the 'total' key from **Produced** to the second column of your new **Data Summary** row.
5. Select the cell you just added to and type the following text in front of your key: '**Total Widgets Produced**'. In the Text Editor you can make it bold so it stands out.
6. Go to the **Preview panel** to check your report. You should see all your data in the report.

Key Browser

Show Calculations

Parameters

Data Sources

WidgetProduction Data

Capacity

Facility

Minutes

Produced

value

total

average

max

min

Running total

Running average

Running count

count

Built In

Property Inspector

Configure Table

Properties

Data Key

WidgetProduction Data

Grouping

WidgetProduction Data

Rows

Header

Details

Summary

Sorting

Default

Basic

TopN

Options

Page Break

Make TableGroup

Report Overview

Data

Design

Preview

Schedule

Facility	Capacity (Units)	Run Time (Min)	Units Produced	Efficiency - Units/Min
WidgetProduction Data Header				
@Facility@	@Capacity@	@Minutes@	@Produced@	@Produced/Minutes@
WidgetProduction Data Details				
Total Widgets Produced: @total.Produced@				
WidgetProduction Data Summary				

Report Overview

Data

Design

Preview

Schedule

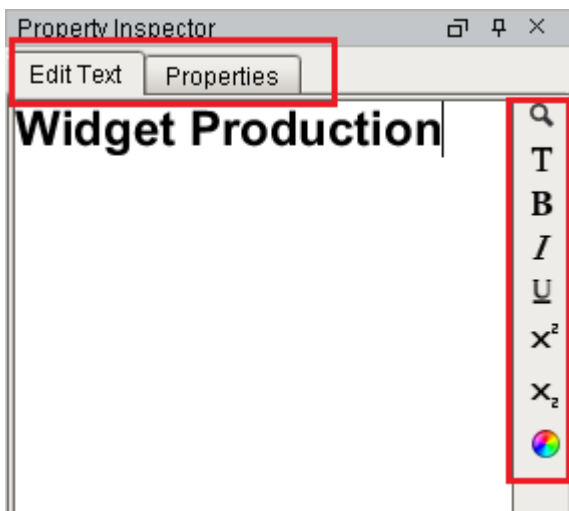
Facility	Capacity (Units)	Run Time (Min)	Units Produced	Efficiency - Units/Min
California	2000	345	665	1.93
Texas	3424	924	1674	1.81
South Africa	734	154	232	1.51
Brazil	1131	325	882	2.71
China	5324	297	2764	9.31
Norway	436	383	143	0.37
Kenya	1431	164	423	2.58
Italy	543	234	524	2.24
Romania	154	45	78	1.73
Peru	624	732	523	0.71
Total Widgets Produced: 7908				

Adding a Header, Date, and Page Numbers to a Report

Header

The **Text Shape** is great to use for a report title, although it can be used anywhere in the report to add text to the page.

1. To activate, click on the **Text Shape**  icon on the Component Palette.
2. Once selected, click and drag on the top of the page to create a Text Shape. Give your report a title by typing into the field of the shape.
3. You'll notice the **Edit Text** tab in the lower left corner of the **Property Inspector**, along with some buttons that let you customize the look and layout of the text. This configuration area will change depending on the selected component on the report. In the same area, you'll notice there is a tab titled **Properties**. The Properties tab provides access to a component's various properties. Feel free to experiment with the settings like font in the editor or property table to customize your title and/or text shapes.



Date and Page Numbers

In addition to a title, let's add some metadata, such as the date the report was generated. However, you don't want to type the date into the text field, because all reports you run (could be today, tomorrow or next year!) will show what date you typed. Instead, you want the date to reflect the day the report was generated.

1. If you look at your **Key Browser**, you'll see a folder called **'Built In'**. Expand this folder and you'll see a number keys that are common in reports.
2. Drag the **'Date'** key and drop it on the report just below your title. This key represents the date and time the report is executed.
By default, Text Shapes initially show just the date, but this can be modified by via the **Date Format** property, which is located under the **Properties** tab when the Text Shape is selected. This example will use the default Date Format, but feel free to modify this.
3. Next, drag the **'Page of PageMax'** key to the bottom of your report.

Adding text on a page shared with a repeating component (such as a table) will add the text to all pages created by the component. In some cases, like page numbers, date, and title at the top, this is desirable.

Here is what the report look likes in the Design and Preview panels after the header and page numbers were added to the report:

Key Browser

Show Calculations

Parameters

Data Sources

Built In

Report

Date

Page

PageMax

Page of PageMax

PageBreak

PageBreakMax

PageBreakPage

PageBreakPageMax

Row

Property Inspector

Edit Text

Properties

@Page@ of @PageMax@

Q

T

B

I

U

X²

X₂

Color

Report Overview

Data

Design

Preview

Schedule

Widget Production

@Date@

Facility	Capacity (Units)	Run Time (Min)	Units Produced	Efficiency - Units/Min
WidgetProduction Data Header				
@Facility@	@Capacity@	@Minutes@	@Produced@	@Produced/Minutes@
WidgetProduction Data Details				
Total Widgets Produced: @total.Produced@				
WidgetProduction Data Summary				

@Page@ of @PageMax@

Report Overview

Data

Design

Preview

Schedule

Widget Production

July 7, 2017

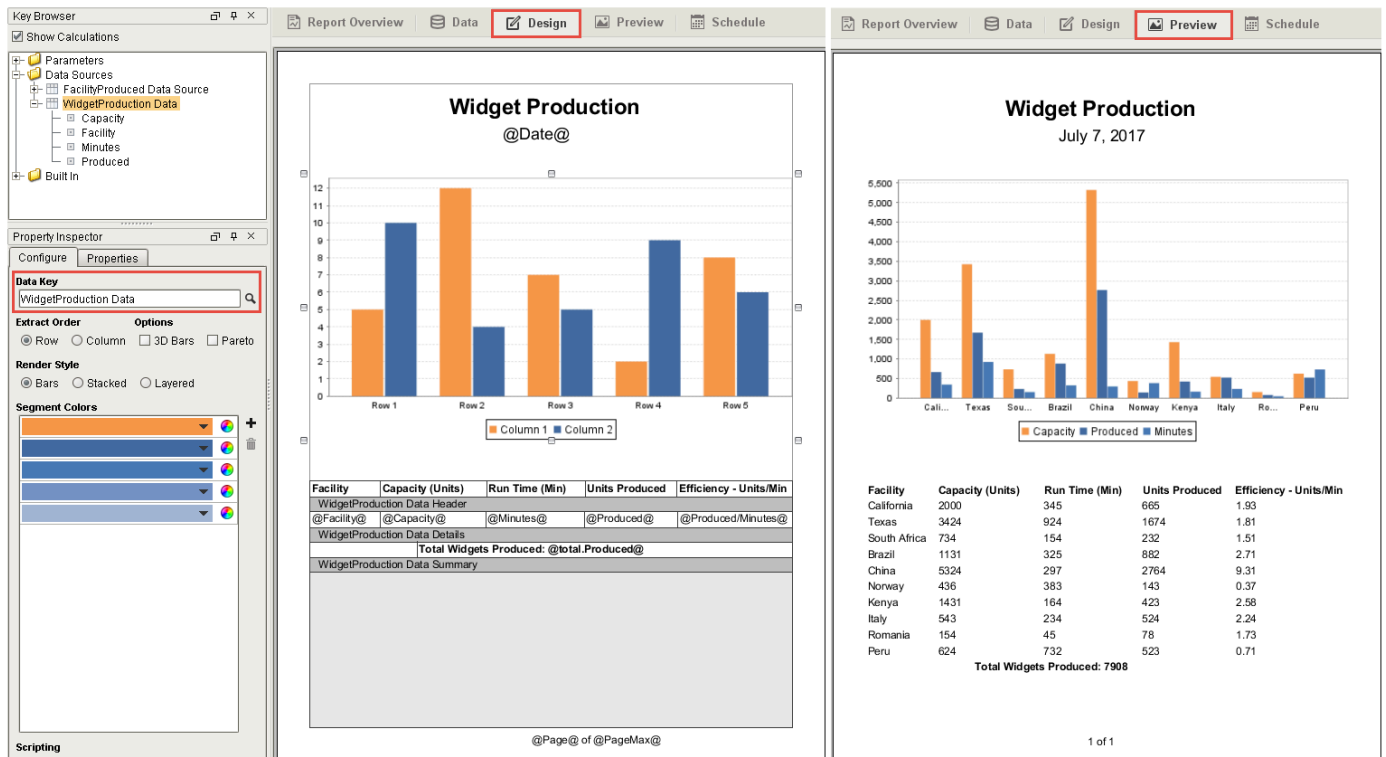
Facility	Capacity (Units)	Run Time (Min)	Units Produced	Efficiency - Units/Min
California	2000	345	665	1.93
Texas	3424	924	1674	1.81
South Africa	734	154	232	1.51
Brazil	1131	325	882	2.71
China	5324	297	2764	9.31
Norway	436	383	143	0.37
Kenya	1431	164	423	2.58
Italy	543	234	524	2.24
Romania	154	45	78	1.73
Peru	624	732	523	0.71
Total Widgets Produced: 7908				

1 of 1

Adding a Chart to the Report

To finish the report, we just need to add a chart. You may need to resize the table a little bit and add a chart right under the page header.

1. In the **Design** panel, drag a **Bar Chart** component from Report Palette into the space above the table. You will need to resize the Bar Chart and the Table to your report page.
2. Next, drag the **Data Source** (i.e., WidgetProduction Data) from the **Key Browser** to the **Data Key** on the **Configure** tab.
3. Go to the **Preview** panel to see how your report looks. Instantly, you'll notice the Bar Chart adds a bar for each column in the data, where our goal for this report is to only show the number of widgets produced. If you want to change the colors of the bars, go back to the Design panel and edit the **Segment Colors** under the Configure Tab.



Edit Data Sources to Accommodate Report Requirements

If we were using Query data sources, we would create an additional data source or use a nested source to get the two columns we want. For this example, let's create a subset of our original dataset (i.e., WidgetProduction Data) so we only get the data we are interested in.

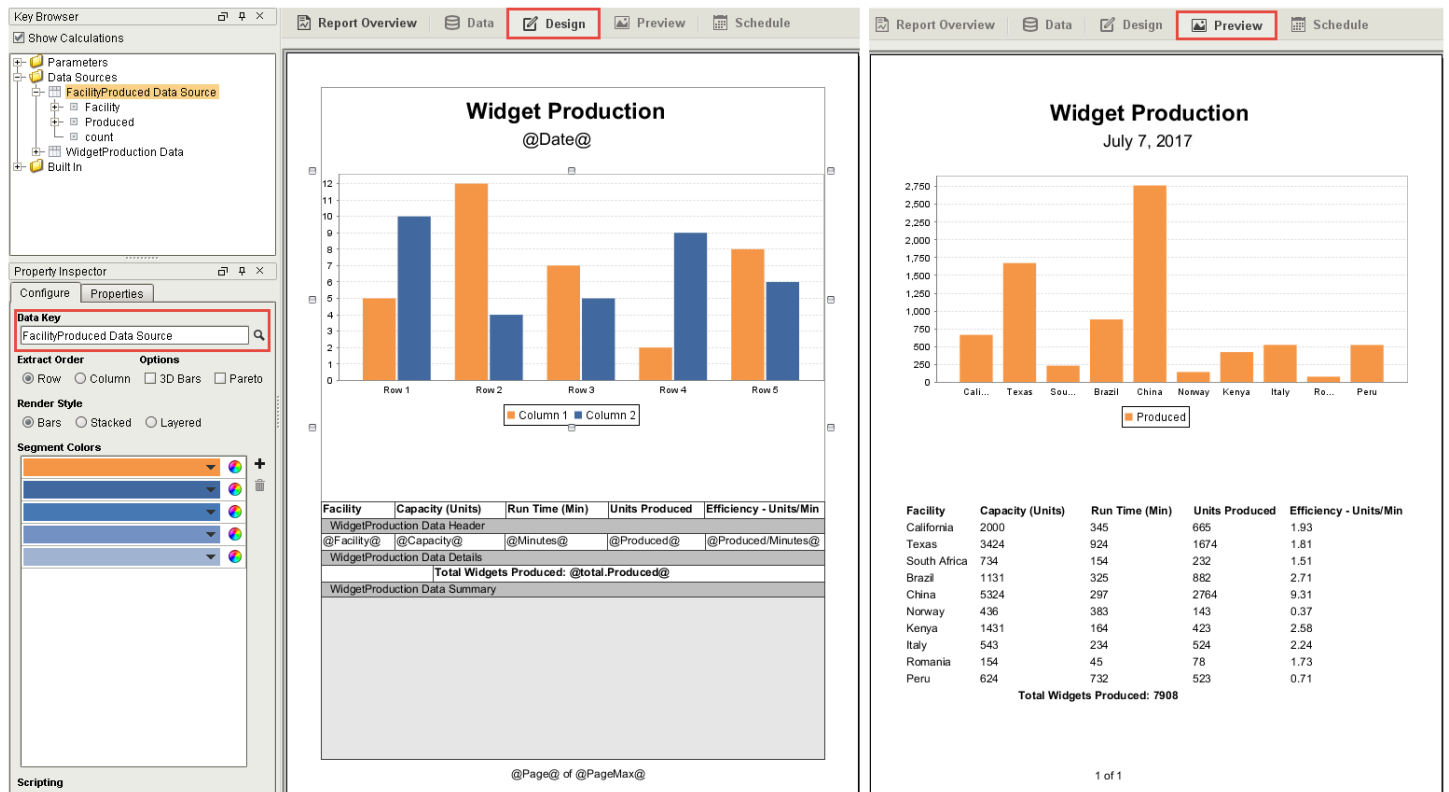
1. In the **Data** panel, add another **Static CSV** data source (i.e., **FacilityProduced Data Source**) that includes only the Facility and Produced columns. For this example, copy the following code for your data source:

FacilityProduced Data Source

```
Facility, Produced
"California",665
"Texas", 1674
"South Africa", 232
"Brazil", 882
"China", 2764
"Norway", 143
"Kenya", 423
"Italy", 524
"Romania", 78
"Peru", 523
```

An easy way to do this is to select the chart and drag the '**FacilityProduced Data Source**' key onto the **Data Key** field in the Property Inspector. Now when we generate our report with the new data, we

only get a Bar Chart with the items produced for each facility.



Viewing Reports in Runtime

Once your report is created, you can view the report from the Ignition runtime or Designer's Preview Mode with the **Report Viewer** component. You can add this component to any Vision window and select the name of your report from the Report Path parameter dropdown.

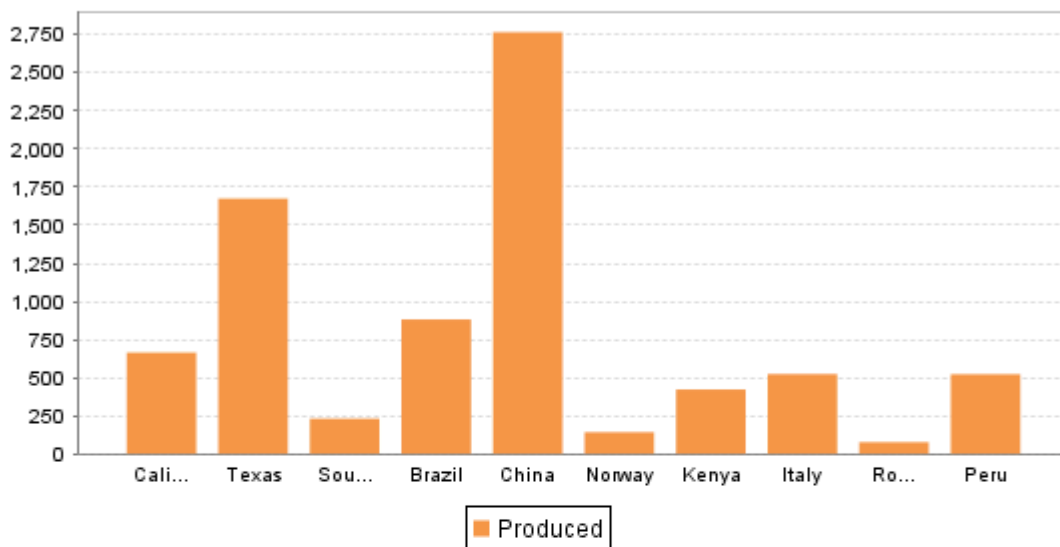
Saving a report from the Report Viewer is simply a matter of right clicking on the report in the Ignition runtime or Preview Mode of the Designer, and selecting the format you wish to save it as. Selecting from the menu will open a Save or Print dialog in the Client window as well as in Preview Mode.

The Report Viewer component supports the following options:

- **Print** - send your report to a connected printer
- **PDF** - formatted for viewing in PDF Viewers
- **HTML** - an HTML document viewable in your web browser
- **PNG** - save as an image

Widget Production

July 6, 2017



Facility	Capacity (Units)	Run Time (Min)	Units Produced	Efficiency - Units/Min
California	2000	345	665	1.93
Texas	3424	924	1674	1.81
South Africa	734	154	232	1.51
Brazil	1131	325	882	2.71
China	5324	297	2764	9.31
Norway	436	383	143	0.37
Kenya	1431	164	423	2.58
Italy	543	234	524	2.28
Romania	154	45	78	1.73
Peru	624	732	523	0.71

Total Widgets Produced: 7908

Print

Save as PDF

Save as HTML

Save as PNG

1 of 1

Creating a Schedule

Once a report is designed, you can have a list of scheduled times and actions that will execute automatically at specified times using the [Report Schedule](#) functionality.

Use the following steps to create a schedule for your report:

1. In the **Schedule panel**, click on the plus icon **+** on the right side of the panel. A new row is added to the table. You'll notice the user interface is split into two sections. On the top is a table which

contains a list of all the schedules for your report as well as the Actions that will occur following report generation.

2. The **Schedule tab** is where you setup the schedule for your report to run. The UNIX Crontab format is used to setup schedules, but there are some common schedules available in the dropdown list to select from. To set a schedule, select a row in the Schedule Table. Once the row is selected, the Schedule section becomes editable.

The screenshot shows the 'Schedule' tab of a report configuration interface. At the top, there's a navigation bar with tabs: 'Report Overview', 'Data', 'Design', 'Preview', and 'Schedule'. Below this is a table with two columns: 'Schedule' and 'Actions'. The first row in the table is 'At 12:00 AM'. Below the table are three tabs: 'Schedule', 'Parameters', and 'Actions'. The 'Schedule' tab is active, showing a 'Common Settings' dropdown set to 'Once Per Day (0 0 ***)' and several input fields for 'Minutes', 'Hours', 'Days', 'Months', and 'Weekdays'. Each field has a green checkmark and a dropdown menu. The 'Options' section at the bottom has a checked 'Enabled' checkbox.

3. To create a new schedule, select a schedule from the Common Settings dropdown list, or choose from one of the Common Settings and customize it using any of the selection boxes. In this example, the **Hours** dropdown was set to **Every 12 hours (*12)**, which will run the report daily every time the current hour changes to 12 (i.e. midnight and noon). Now that a schedule has been created, we need to specify an Action to occur at the scheduled times.

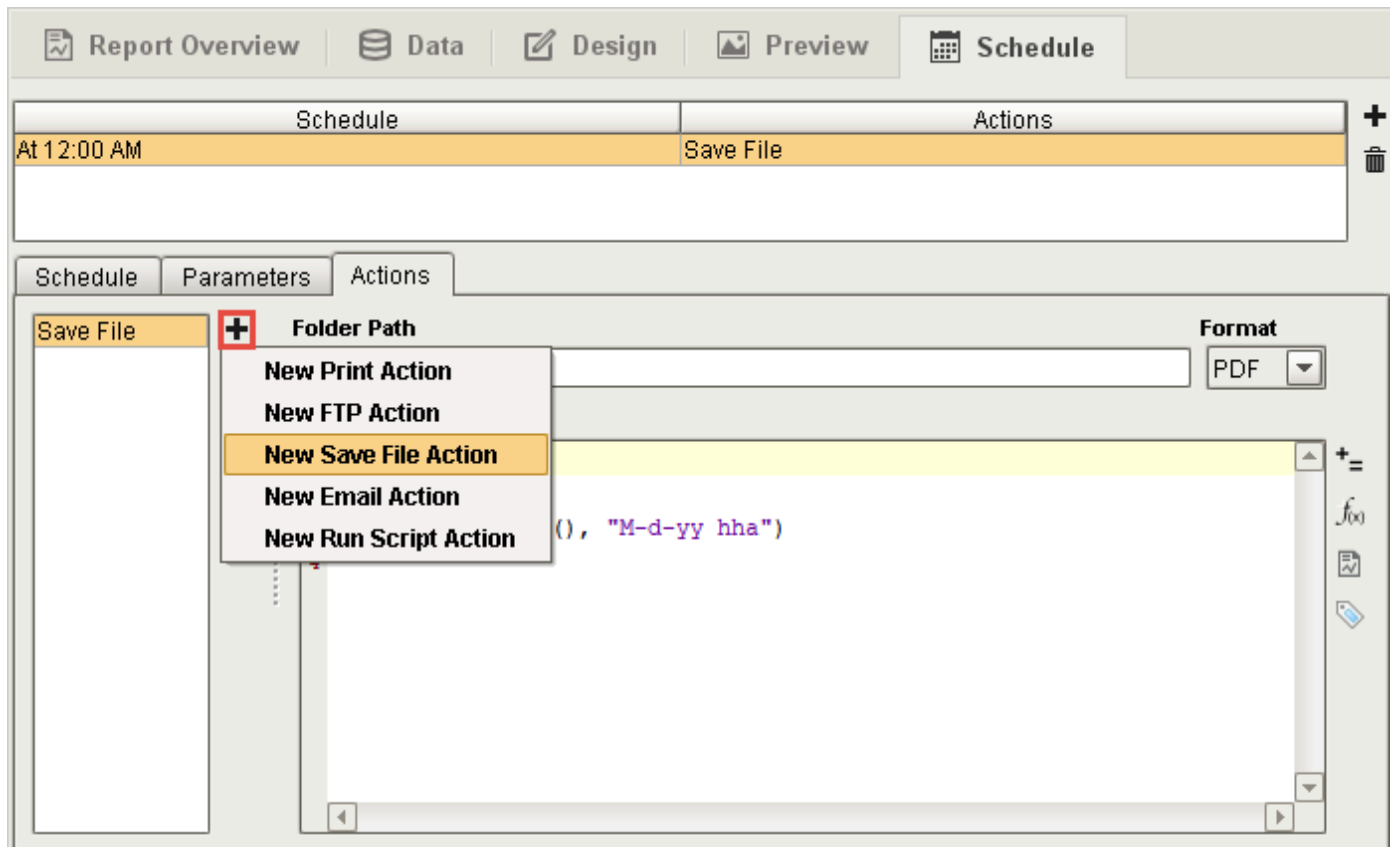
Scheduling a Save Action

The **Save File Action** saves a copy of the report to any folder Ignition has access to whether it's on your local computer or a shared drive.

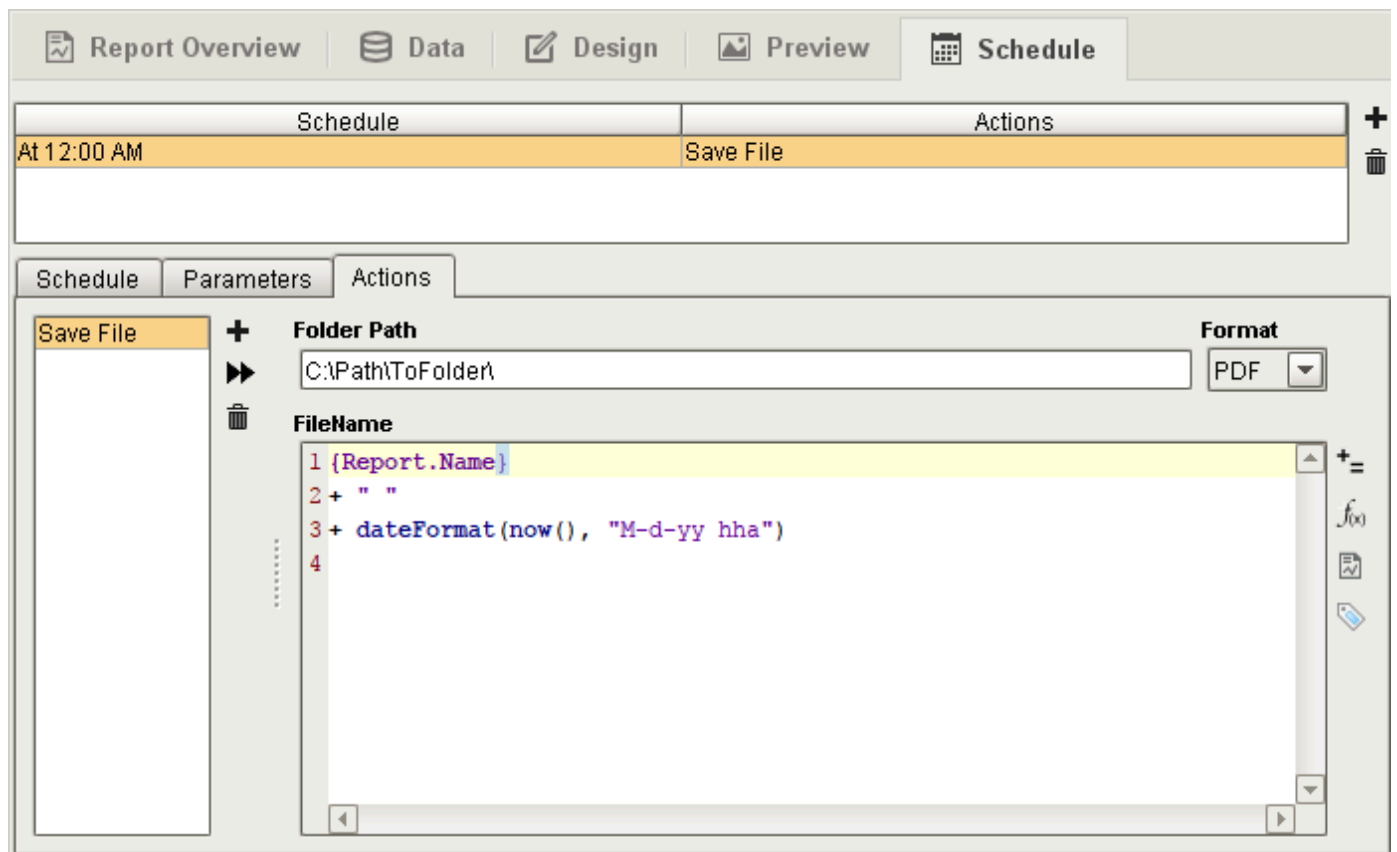
Here are a few simple steps to saving a report:

1. In the **Schedule** panel, create a **Schedule** to automatically save a report by clicking on the plus icon **+**, if you don't already have one.
2. Next, click on the **Actions** tab.

3. Click on the plus icon **+**, and select **New Save File Action** from the dropdown list.



4. Enter the **Folder Path** where you want to save your report to. You can save them to your local computer or a shared drive. Here are a couple of examples of a folder path for a local computer and a shared drive: "**C:\Reports**" and "**Share_Drive\Folder**". This path is always relative to the Gateway, so you should always take the Gateway server's operating system into consideration when specifying the path.
5. Select the file **Format** from the dropdown list that you want to save the report as. PDF is a very common format.
6. By default, your file will be saved with your report name followed by month, day, year, and hour. If you prefer not to use the default filename, you can change it.



7. Save your project. You've just completed configuring a schedule! You can test out the action by clicking on the **Run Immediately** button (▶▶)

Disabling the Schedule

Now that your schedule is complete, you may want to disable the schedule so that a new copy of this example report is not created every 12 hours. The **Schedule** tab has an **Enabled** checkbox that can be disabled to prevent the schedule from occurring.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Labels with Embedded Barcodes

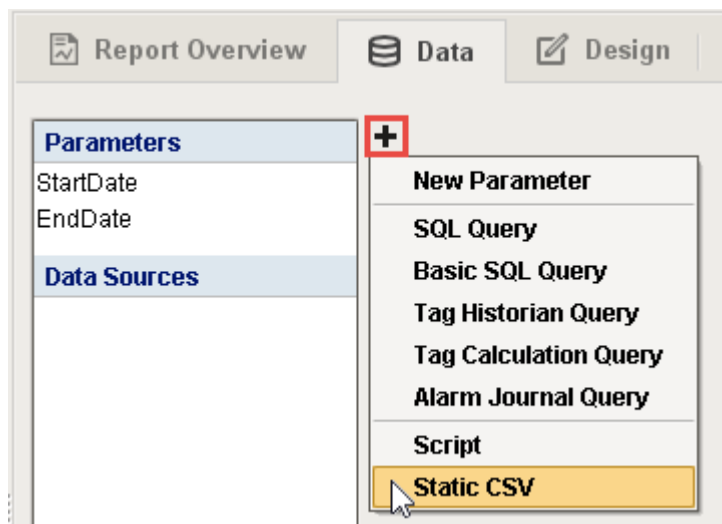
Using labels with embedded barcodes is very common practice. Barcodes are typically used to print out mailing and shipping labels, and can be used to identify and track almost anything. When creating any type of label, you can specify your own dimensions or use standard Avery label sheets. The size of the labels component is based on the values in the Label Attribute section of the Configure Labels tab. When you update any of the Label Attributes (i.e., Rows, Columns, Width, Height, and Spacing), the labels automatically resize based on the values that were specified.

The best way to explain how to setup labels with embedded barcodes is to use an example. Let's create a set of custom labels for the University Research Lab to better track the DNA Primers used in experiments. We want the labels to contain information about each Primer and where it belongs. In addition, we want the barcode to contain the Primer name, number of the lab, and the location within the lab that the samples belong to. We also want the labels to include a sequence number to identify the process order for each step in the experiment. A barcode will be embedded into the template label to create QR Codes, and a QR Code Scanner will be used to track the Primers.

Setting Up Data Sources

Before setting up any labels, you first need to specify a **Data Source** in the **Data panel**.

1. Click the plus icon **+** to add a **Static CSV** Data Source. This example will use the static data listed below, but could easily substitute real data from a database by using a **SQL Query Data Source** instead.

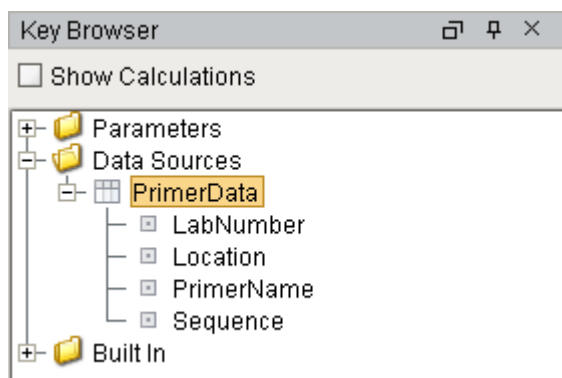


2. With the **Static CSV** Data Source editor open, copy and paste the CSV data below into your Data Source editor, and give your data a meaningful name (i.e., PrimerData) in the **Data Key** field.

PrimerData Datasource

```
PrimerName, Sequence, LabNumber, Location
16F, CGG TTA CCT TGT TAC GACT T, 2, Cooler 2
3A0X, GCA AAT GGC ATT CTG ACA TCC, 3, Freezer
EGFPC1R, CAT TTT ATG TTT CAG GTT CAG GG, 2, Cooler 2
pGLrev, CTT TAT GTT TTT GGC GTC TTCC, 1, Cooler 1
M13R, CAG GAA ACA GCT ATG ACC, 3, Freezer
SV40-promoter, TAT TTA TGC AGA GGC GAGG, 1, Cooler 1
pBabe5, CTT TAT CCA GCC CTC AC, 2, Cooler 2
EGFPC1R, CAT TTT ATG TTT CAG GTT CAG GG, 2, Cooler 2
pGLfor, GTA TCT TAT GGT ACT GTA ACT G, 3, Walk-in Shelf A
EF-1xForward, TCA AGC CTC AGA CAG TGG TTC, 1, Cooler 1
```

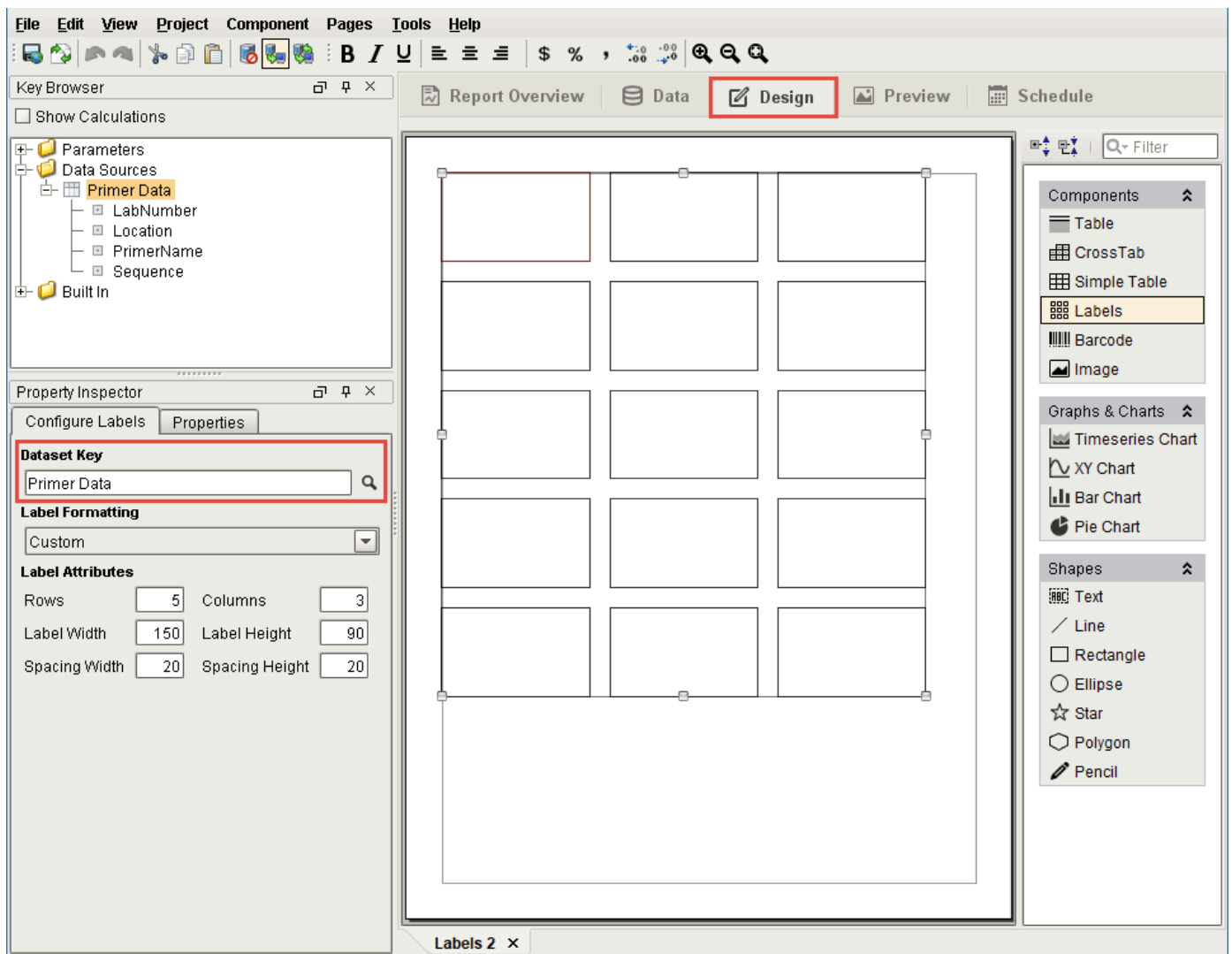
3. Next, go to the **Design panel** and expand the **Datasources** folder in the **Key Browser**. You'll see that each column of data in the Static CSV datasource is represented by its own Data Key.



Configuring the Label and Embedded Barcode

Now that you know what information you want on your labels, you have to decide what you want your labels to look like.

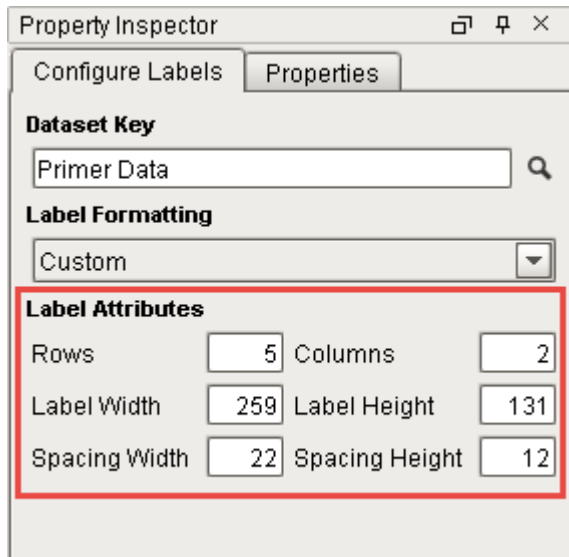
1. In the **Design** panel, drag a **Labels** component from the Report Component Palette to your workspace. By default, 15 labels will display in your workspace.
2. Specify the Data Key that maps to the Data Source that you want to drive the label. Drag your **Datasource** (i.e., PrimerData) to the **Dataset Key** in the Configure Labels tab of the Property Inspector.



3. While the Label component is selected, edit the number and size of the labels in the **Configure Labels** tab. The size of the label is based on the values for the Label Attributes: Rows, Columns, Label Width, Label Height, Spacing Width, and Spacing Height. When any of the Label Attributes are changed, the labels on the page are automatically resized based on these values.



To commit values to any of the Label Attributes, enter a value and hit Enter.



The screenshot shows the 'Property Inspector' window with two tabs: 'Configure Labels' and 'Properties'. The 'Properties' tab is active. Under the 'Dataset Key' section, 'Primer Data' is entered in the search field. Below that, the 'Label Formatting' section has a dropdown menu set to 'Custom'. The 'Label Attributes' section is highlighted with a red rectangular box and contains the following settings:

Attribute	Value	Attribute	Value
Rows	5	Columns	2
Label Width	259	Label Height	131
Spacing Width	22	Spacing Height	12

❗ TEMPLATE LABEL

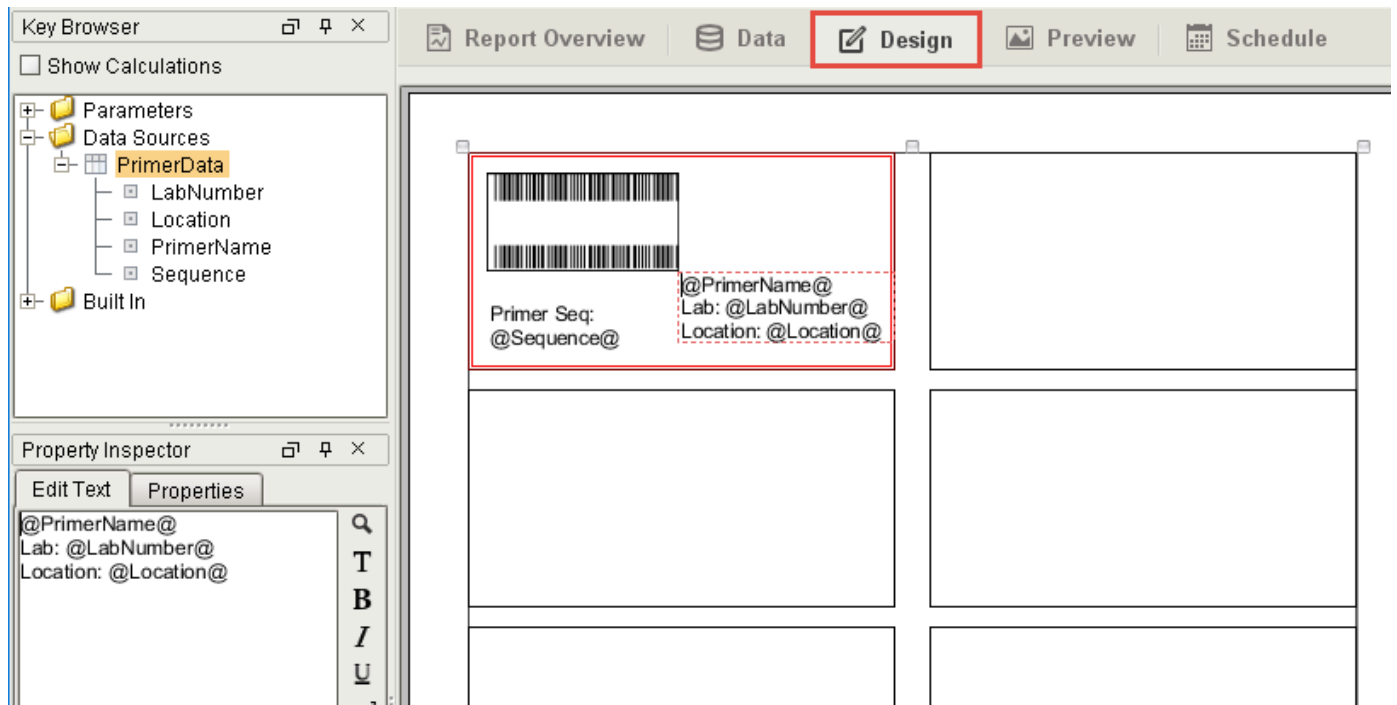
The top-left label in the Label component is the template label. Here is where you'll configure your label. To edit a component or shape within a template label, super-select it by double clicking on it. When you Preview and Print your labels, the template label pulls all the data from the dataset and populates all your labels.

- Next, drag a **Barcode** component and place it in the upper left corner of the template label.
- Now you're ready to drag your data keys (i.e., PrimerName, LabNumber, and Location) from the Key Browser to the label template.

💡 TIP

Once you drag the first data key to your label template, the **Edit Text** tab will open. You can drag the other data keys directly into the Edit Text tab so they make one shape. This makes editing a little easier.

Next, drag the **Sequence** data key into the lower left corner of the label.

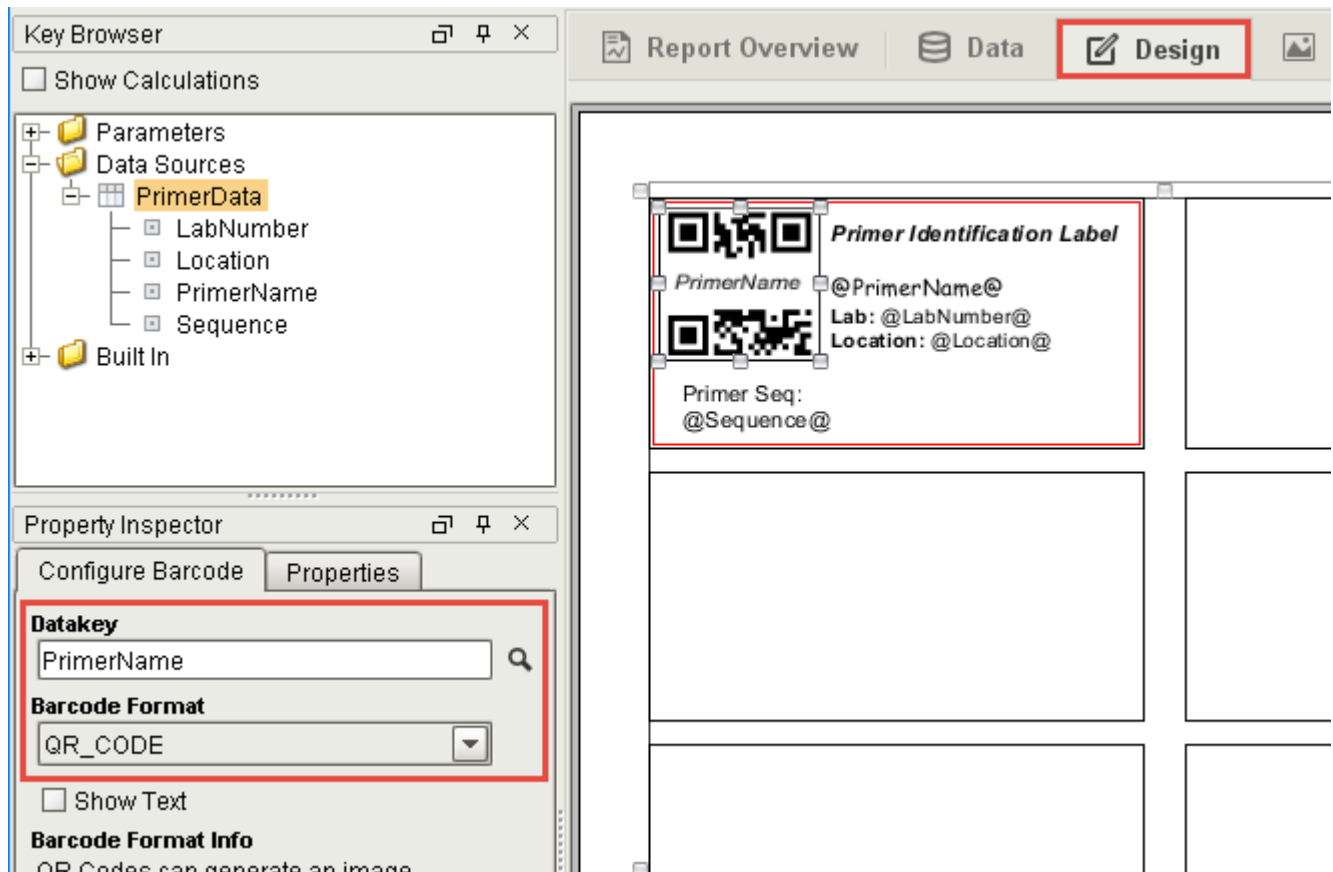


6. Go to the **Preview panel** to verify if all your data populated all the labels, and check how all the data fits on the label because you may need to resize the barcode or text components.
7. Go back to the **Design panel**, click on the **barcode** component, and specify the data key you want to map to the Data Source to drive the barcode. In this example, we want the barcode to encode the PrimerName to the Data Key field of the barcode.
Next, choose a **Barcode Format** from the dropdown list based on your label requirements (i.e., product type, number of characters required, label spacing, etc.). For this example, select the QR_CODE barcode format.

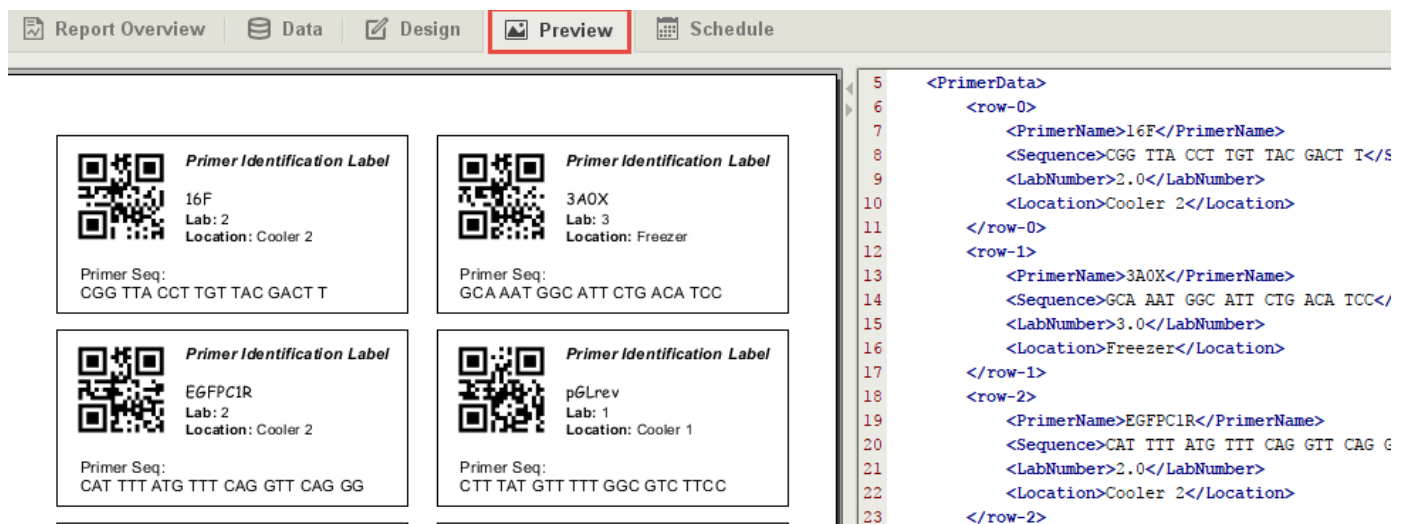
NOTE

If you want to push multiple fields into the barcode text, you can concatenate them together in the Data section of the report.

8. Add a Title for your label by dragging a **Text Shape** component to the top of the label. Enter a title inside the text box (i.e., Primer Identification Label).
In addition, in the Edit Text tab, you can mix plain text and data keys, change the font, size, and style, and even bold text that you want to stand out.



- Go to the **Preview Panel** to view the finished design layout and the HTML code. It's not uncommon to go back to the Design panel to resize components and shapes on the label several times so they fit correctly and the data is readable. This may take several iterations.



Printing Labels

You can print your labels by creating a **Print Action** on the Schedule tab, and run it by clicking the "Run Immediately" button. If you receive new shipments of Primers on a set schedule, you can setup the labels to print automatically. For this example, we want to simply print our labels on demand.

ScheduleParametersActions

Print

+

»

Primary Printer

Default Printer

Backup Printer

None

Print Mode

Vector

Raster

300

dpi

Copies

1

Options

Print on both sides

Collate

Use AutoLandscape Mode

Page Orientation

Portrait

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Converting Legacy Reports



INDUCTIVE
UNIVERSITY

Converting Legacy Reports

[Watch the video](#)

Overview of Converting Legacy Reports

The complete rebuild of the Reporting Module in Ignition 7.8 brought many improvements that would have been impossible to add to the legacy module. To preserve report functionality and prevent problems with backward compatibility, any existing reports will function as they always have. To get the most out of your reports and enable new functionality in such as Scheduled Reports, you'll have to convert your old Vision Report Panel components to new Report Resources. In an effort to minimize barriers, we created a Report Conversion process that will attempt to convert Ignition 7.7- reports into 7.8+ reports.

We encourage users to convert their reports to the new format if they feel they would benefit from the added functionality, but in doing so, it's important to keep some things in mind:

Some Components have Imperfect Conversion

There are a number of components that have been upgraded or completely rebuilt. Due to the changes, some components and/or configurations may not convert perfectly to the newer module. Specifically, the upgrade to the Barcode component and addition of 2D barcodes utilizes a new encoding system that does not have perfect parity with the legacy encoder. The new component does not explicitly support encoders for some of the Narrow and Extended codes, as well as MSI. Reports which require the Narrow or Extended Code 39/Codabar or MSI barcodes will not convert perfectly. Lastly, the changes to the Charts brought many improvements, but will look a little different and may require some configuration.

Data Sources will need Configuration

Data Sources are a huge improvement in the new module. It's now far easier to collect and use data from nearly anywhere in Ignition. Unfortunately, the custom properties in the Legacy Report module do not directly map over to the new Data Sources. When a report is converted, its custom properties will be converted to Parameters. Parameters are great in that they allow for a quick conversion and enable things like Scheduled Reports, but Data Sources will need to be manually configured if desired.

These are the two major caveats to be aware of when converting. The conversion tool has been tested with a variety of legacy reports, but there may be additional factors preventing conversion. We encourage users to either visit the Inductive Automation forum or contact our support department, and let us know – improvements to the conversion tool can only occur if we are made aware of errors!

Conversions are Non-Destructive

With the potential issues covered, it's important to note that report conversions do not destroy or alter the original report. If the conversion isn't successful for some reason, the new report can simply be deleted and the old one exists as it always did. If the conversion is successful, the old report can be saved to an unused window or exported as a backup before being deleted. Conversions may not always be perfect, but there is no risk in trying.

How to Convert

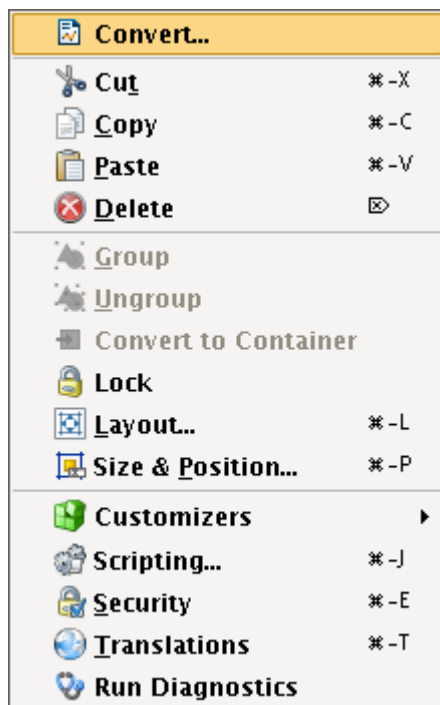


INDUCTIVE
UNIVERSITY

**Sample Legacy Report
Conversion**

[Watch the video](#)

Converting a legacy report is quite simple. In the Ignition Designer, open the window where the Report Viewer component exists. If you right click on the component, the new **"Convert..."** option is at the top of the popup menu. Selecting it will start the conversion process which first prompts for the "OK" to proceed, then asks what the new Report Resource should be called.



Upon conversion, a new Report Viewer will be added to the existing window in addition to the old report. If you check the Project Browser, you'll see the new report under the Reports tree node. Open the newly created Report Resource to make any edits, to add data sources, or add Scheduled Actions!

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Report Schedules



INDUCTIVE
UNIVERSITY

Scheduling Basics

[Watch the video](#)

Execute and Distribute Reports Automatically

Perhaps one of the most compelling features of the Reporting module is the Scheduling system. Once a report has been designed, it can be run and delivered automatically without using a client. Since reports run on the Gateway, it gives you total flexibility when reports should run, and how they should be delivered.

Scheduling a Report

For any Scheduled Report, there are three settings that need to be configured:

1. When the **Scheduled** report executes (i.e., date, time, and frequency). There can be multiple schedules on a report, and each schedule can be refined down to a minute resolution.
2. What **Parameters** will be used to determine the report data. Can use the default parameter values or have new values passed in.
3. The **Actions** that occur following report generation. There can be multiple actions per schedule.

SCHEDULES EXECUTED ON THE GATEWAY

It is important to note that schedules execute on the Ignition Gateway, not in the client. This means that schedules should be created to Gateway local time, not Client or Designer time. Reports can also be run in the client using the [Report Viewer component](#).

Schedule Table

The Schedule Table is at the top of the Schedule tab and will display a list of all currently configured schedules and actions. A single report can have multiple scheduled times and actions configured, allowing it to be saved with different parameters, at different times, and with different actions.

Creating a scheduled report is easy. To add a new schedule click on the plus icon  on the top-right corner of the Schedule panel. In doing so, you've created a new Schedule which can now be configured. To remove any rows, simply select the row in the table and click the delete button  on the right side of the panel. To configure the schedule, select it, and fill in the tabs below the table: **Schedule**, **Parameters**, and **Actions**.

Schedule Tab

The **Schedule** tab is where the scheduled time is set for the report to run. Schedules are driven via Cron formatted strings, a popular scheduling format used in computing. If you aren't familiar with Cron, don't worry, we've created an intuitive user interface that allows you to set schedules easily without having to learn anything new. The Schedule UI provides some convenient pre-made **Common Settings**. If there isn't a setting for you in the Common Settings combo box, choose one that is close and then simply customize it using any of the selection boxes below. For more information on the UNIX crontab scheduling, see the [Scheduled Backups](#) section.

You can also enable or disable the schedule by checking the **Enabled** option.

Report OverviewDataDesignPreviewSchedule

Schedule	Actions
At 0:00 AM, on day 1 and 15 of the month	

ScheduleParametersActions

Schedule

Common Settings

On the 1st and 15th of Every Month (0 0 1,15 **)

Minutes

0

:00 (0)

Hours

0

Midnight (0)

Days

1,15

On the 1st and 15th of the Month (1,15)

Months

*

Every Month (*)

Weekdays

*

Every Day (*)

Options

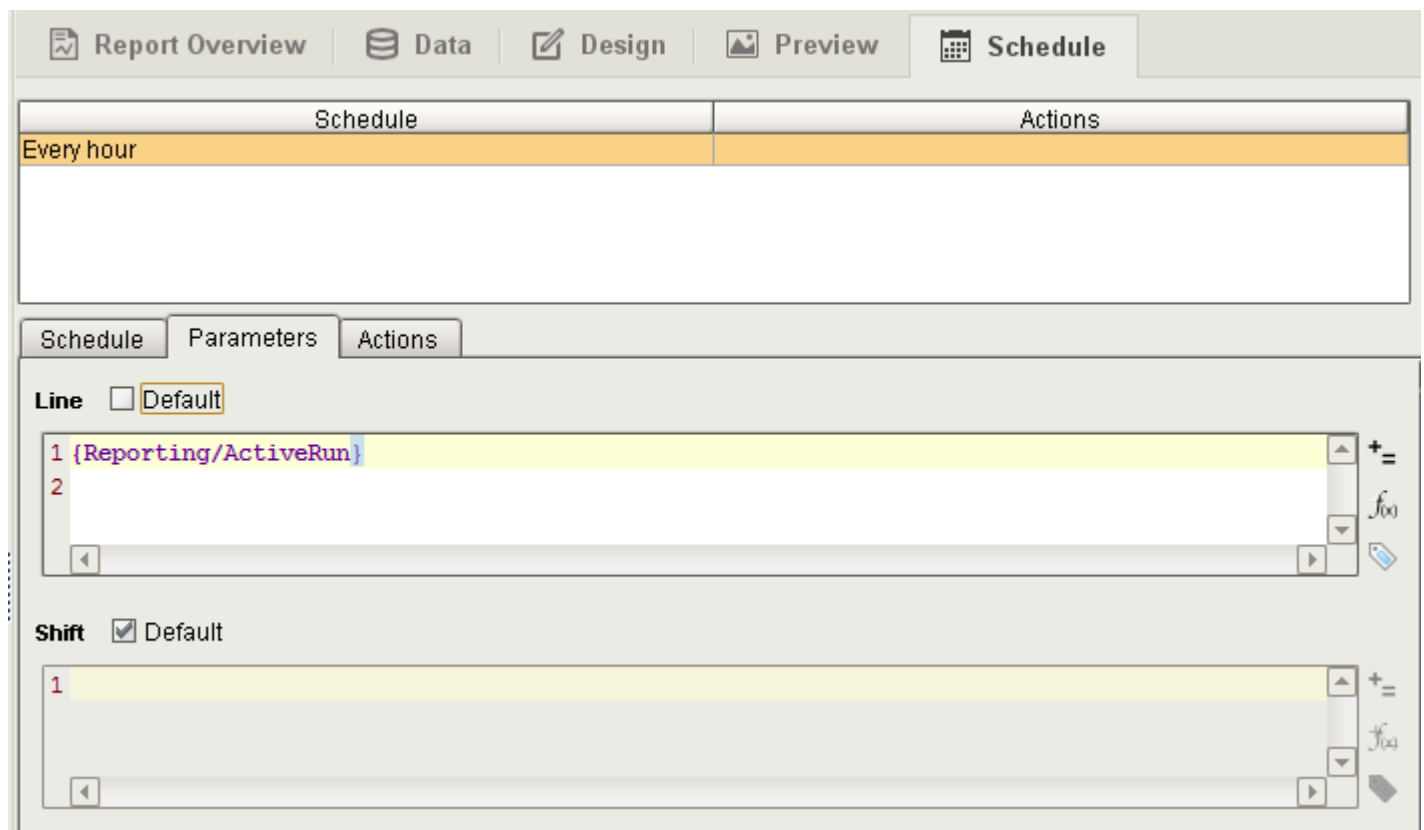
☒ Enabled

Parameters Tab

The **Parameters** tab is where you can override the default values of your parameters when the schedule runs. You can alter these default values to tailor scheduled reports without having to change the Report's design or configuration. For instance, imagine a report which summarizes how many widgets a factory produced during a given shift. Rather than create a separate report for each shift, we can create multiple schedules (one for each shift) and simply alter a shift parameter. Using Parameters and Schedules, users can avoid creating multiple reports while keeping projects more maintainable.

Each parameter of the report will be listed, and they can either be set to their default parameter value by selecting the checkbox, or can be customized by deselecting the checkbox and specifying a parameter value to pass in at the time the scheduled action executes.

In this example, the parameter "**Line**" is being overridden to the value of the "**Reporting/ActiveRun**" Tag.



Actions Tab

The **Actions** tab is where you can setup actions that will run following the generation of a scheduled report. There can be any number of Actions associated with a Schedule, covering virtually any requirement for automatically storing, distributing, or notifying upon completion of the report. Each action has its own custom configuration interface to make adding and editing Actions simple. To learn more about how to configure Actions, refer to [Scheduling Actions](#).

The Actions you can perform following report generation are the following:

- **Print File** - Configure print settings which execute when the report is generated. This is often used to create hard copies of reports automatically.
- **FTP** - Send your report to a file server for backups or storage. Automatically backs up your reports to a service.
- **Save File** - An easy way to save a report to a location on your local Gateway computer or shared network drive.
- **Email** - Your report can be emailed to a list of email addresses or users with specific roles. You can configure the Subject, Filename, and Body of the email.
- **Run Script** - A Run Script Action provides the ability to fully customize how a finished report is handled. The Run Script Action provides the report's data as well as the bytes generated according to the **Format** option in the configuration panel.

With any of these scheduling actions, you also have the option of running the report immediately, by clicking the double arrow **▶▶** next to the action.

The screenshot displays the 'Report Scheduler' interface with the 'Schedule' tab selected. The top section shows a schedule of 'Every 12 hours' and the action 'Save File, Email'. Below this, the 'Actions' tab is active, showing a list of actions on the left: 'Save File' and 'Email'. The 'Email' action is selected and has a double arrow icon next to it, indicating it is set to run immediately. The configuration panel for the 'Email' action is shown on the right, with the following fields:

- From Address:** user.name@mail.com (with a green checkmark icon)
- Subject:** 1 {Report.Name}
- Attachment Filename:** 1 {Report.Name}, 2 + " - ", 3 + dateFormat(now(), "M-d-yy ha"), 4 + ".pdf"
- Body:** 1 "Report attached"
- Mail Server:** SMTP (with a refresh icon and a link to 'Create new server')
- Format:** PDF
- Retries:** 1
- Recipients Source:** User Roles
- User Source:** default (with a refresh icon)
- Recipient Roles:** Administrator, Prod_Sups (with a green checkmark icon)

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

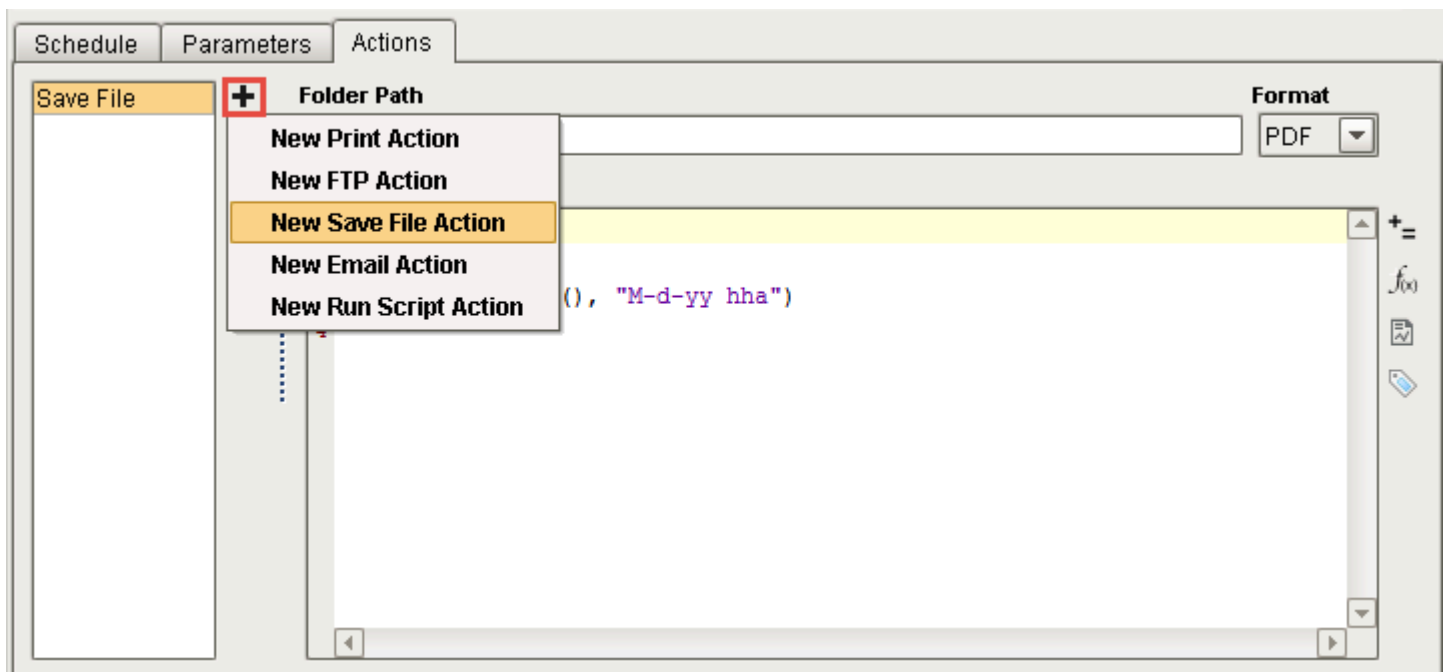
Scheduling Actions

Scheduling Actions

Actions can be configured to run once the report has generated at the scheduled time. Each action has its own custom configuration interface to make adding and editing Actions simple. The Actions you can perform following report generation are Print, FTP, Save, Email, and Run Script. You can even have multiple actions on the same schedule. So you can save the report to the hard drive, as well as email it out to multiple users.

Before creating any Scheduling Actions, you must first [create a schedule](#).

To create an Action, click the **Actions** tab, click the plus icon **+** and select an **Action**. Actions can be deleted using the trash can button , and executed immediately using the double arrow button .



Print Actions

The Print Action is used to send a report to a printer that is accessible from a computer Ignition is installed on. Here are a list of property descriptions for the **Print Action**.

Property Name	Description
Primary Printer	The primary method of printing the report
Backup Printer	A backup method of printing the report. Will print using this option if the Primary Printer fails. [Optional]
Print Mode	The mode to print the report in. Can be either Vector or Raster. • Vector : Uses math to draw shapes using points, lines, and curves. The most common types of vector graphics are fonts and logos. PDF is a popular vector type. Vector based graphics like SVG image files show images with no pixelation when the size is changed. • Raster : Are composed of thousands of pixels or dots. Set the dpi (dots per inch). Common raster file format extensions are jpg, jpeg, png, tiff, bmp, and gif.
Copies	The number of copies of the report that will print.
Print on both sides	Will attempt to print on both sides of a sheet of paper, if supported by the printer.
Collate	Orders the pages so that a complete report prints before the next copy prints, if applicable.
Use AutoLandscape Mode	Evaluates the page dimensions and determines portrait or landscape orientation.
Page Orientation	The orientation of the page. Can either be Portrait or Landscape.

ScheduleParametersActions

Print

+

▶▶

🗑️

Primary Printer

Default Printer

Backup Printer

Adobe PDF

Print Mode

Vector

Raster

300

dpi

Copies

2

Options

☒ Print on both sides

☐ Collate

☒ Use AutoLandscape Mode

Page Orientation

Portrait

FTP Action

The **FTP Action** can be used to automatically upload your reports to a file server for backups or storage. Here are a list of property descriptions for the **FTP Action**.

Property Name	Description
Server Address	The server address where the report file will be transferred.
Port	The port of the file transfer.
Folder Path	The folder path that the report file will be transferred to.
Format	The file format of the report.
Username	The username that will be used to access the FTP server.
Password	The password that will be used to access the FTP server.
SSL	Will use SSL encryption if True.
Filename	The name of the report file. The Filename property is constructed using the expression language.

Schedule
Parameters
Actions

FTP

+

Server Address

ftp://MyCompany.com

Port

23

Folder Path

C:\Reports

Format

PDF

Username

user.name@mail.com

Password

☐ Use SSL

Filename

```

1 {Report.Name}
2 + " "
3 + dateFormat(now(), "M-d-yy hha")
4

```

Save File Action



INDUCTIVE
UNIVERSITY

Scheduling Actions - Save

[Watch the video](#)

The **Save File Action** will save a copy of the report to any folder the Ignition server has access to, such as a local folder or network shared drive. Here are a list of property descriptions for the **Save File Action**.

Property Name	Description
Folder Path	The folder path to save the report files to. This folder path is for the Ignition Gateway server.
Format	The file format of the report.
Filename	The name of the report files. The Filename property is constructed using the expression language.

The screenshot shows the 'Actions' configuration window. On the left, there's a 'Save File' button with a red '+' icon next to it. The main area is divided into 'Folder Path' (C:\Reports) and 'Format' (PDF). Below these is a 'FileName' text area containing the following code:

```

1 {Report.Name}
2 + " "
3 + dateFormat(now(), "M-d-yy hha")
4

```

There are also some icons on the right side of the FileName editor, including a function icon f(x), a checkmark, and a tag icon.

Email Action



INDUCTIVE
UNIVERSITY

Scheduling Actions - Email

[Watch the video](#)

The Email Action distributes a report via email when the report is finished executing. There is a Recipients Source property that allows you to send emails using either Email Addresses or User Roles. The 'From Address,' 'Subject,' 'Body,' and 'Attachment Filename' are all configurable. The Subject, Filename, and Body editors can utilize Expressions to dynamically add content or change names.

Email Server settings must first be configured on the Gateway webpage under **Configure > Networking > Email Settings** page, or in **Email Actions** and clicking the **Create new server** link. Once you create and save an SMTP profile, you can test your email settings for your mail server on the Gateway webpage under **Email Settings**.

i CREATING AN EMAIL SERVER

Before you setup any reports to be emailed, an email server must be configured. To create an email server if one doesn't exist, use the 'Create new server' link. This link will take you to Configure > Email Settings on the Gateway webpage. There, you will be able to create an SMTP server. For more information, refer to [Gateway Settings](#).

Here are a list of property descriptions for the **Email Action**.

Property Name	Description
From Address	The Email address from which the report is sent from.
Mail Server	The mail server to use to email the report. If one doesn't exist, click on the Create new server link. Refer to Email Settings for more information on that page.
Format	The file format of the report.
Retries	The number of retry attempts if the email that was sent failed to be delivered the first time.
Address Source	Will decide how email addresses are collected. Can be either Email Addresses or User Roles. See the source property tables below for more information. New in 7.9.8 A new ReplyTo Email function was added, allowing you to reply to email actions using the Email Address and User Roles. This simply adds those emails to the "ReplyTo" header of the email sent to the recipient list, so that if recipients choose to reply to that email, their reply is sent to those email addresses as well.
Subject	The subject of the Email. The Subject property is constructed using the expression language.
Attachment Filename	The name of the attached report. The Attachment Filename property is constructed using the expression language.
Body	The body of the email. The Body property is constructed using the expression language.

Email Address Source Properties

Property Name	Description
Addresses	A list of email addresses
Method	The corresponding method of what to do with the email addresses. Options are To , CC , BCC , and ReplyTo .}

User Roles Source Properties

Property Name	Description
Recipient User Source	The User Source to pull users from that match the Recipient Roles to get an email.
Recipient Roles	A list of roles to match with users. Any user that has any of the listed roles will get an email.
ReplyTo User Source	The User Source to pull users from that match the ReplyTo Roles that will be listed in the reply to of the email.
ReplyTo Roles	A list of roles to match with users. Any user that has any of the listed roles will have their email listed in the reply to of the email that gets sent out.

Configuring an Email Action

1. In the **Schedule panel**, create a **Schedule** to automatically email a report by clicking on the plus icon **+**, if you don't already have one.
2. Next, click on the **Actions** tab.
3. Click on the plus icon **+**, and select the **New Email Action** from the dropdown list.
4. Enter the sender's email address in the **From Address** field.
5. Select the **Mail Server** from the dropdown list. If one does not exist, click the **Create new server** link to create one.
6. Select the **Format** from the dropdown list.
7. Enter the number of **Retry** attempts in the event the email failed to be delivered the first time.
8. You can send emails to users using either Email Addresses or User Roles. Under **Address Source** enter either **Email Addresses** or **User Roles**. Note, email recipients can choose to reply

to the email if they prefer, since the email address is added to the 'Reply To' header of the email.

- i. **Email Addresses** - enter individual email addresses under in the **Recipient and ReplyTo Emails** area. To add multiple addresses, click the plus icon **+** on the right side of the window. Next, specify the **Method** of how to send the email: **To**, **CC**, **BCC**, or **ReplyTo**.

- ii. **User Roles** - select the User Source from the dropdown in the **Recipient User Source** field.
 - a. In the **Recipient Roles** field, begin typing a configured role and Ignition will validate it.
 - b. In the **Reply to User Source**, select the User Source from the dropdown. (Optional).
 - c. In the **ReplyTo Roles** field, enter the role(s) you want listed in the 'ReplyTo' header of the email. (Optional).

9. Enter in values for the **Subject**, **Attachment Filename**, and **Body** fields, or use the defaults.

Recipients and ReplyToEmails

The screenshot shows the 'Parameters' tab of an Ignition configuration window. The 'Email' action is selected. The 'From Address' is 'JohnDoe@inductiveautomation.com'. The 'Mail Server' is 'SMTP'. The 'Format' is 'PDF'. The 'Retries' is '0'. The 'Address Source' is 'Email Addresses'. The 'Recipient and ReplyTo Emails' table has one row: 'MaryDoe@inductiveautomation.com' with the method 'To'. The 'Subject' field contains '1 {Report.Name}'. The 'Attachment Filename' field contains '1 {Report.Name}', '2 + " - "', '3 + dateFormat(now(), "M-d-yy ha")', and '4 + ".pdf"'. The 'Body' field contains '1 "Report attached"'.

Address	Method
MaryDoe@inductiveautomation.com	To

Recipients Source - User Roles

The screenshot shows the 'Parameters' tab of an Ignition configuration window. The 'Email' action is selected. The 'From Address' is 'JaneDoe@inductiveautomation.com'. The 'Mail Server' is 'SMTP'. The 'Format' is 'PDF'. The 'Retries' is '0'. The 'Address Source' is 'User Roles'. The 'Recipient User Source' is 'default' and the 'Recipient Roles' is 'Operator'. The 'ReplyTo User Source' is 'default' and the 'ReplyTo Roles' is 'Administrator, Prod_Sups'. The 'Subject' field contains '1 {Report.Name}'. The 'Attachment Filename' field contains '1 {Report.Name}', '2 + " - "', '3 + dateFormat(now(), "M-d-yy ha")', and '4 + ".pdf"'. The 'Body' field contains '1 "Report attached"'.

Recipient User Source	Recipient Roles
default	Operator

ReplyTo User Source	ReplyTo Roles
default	Administrator, Prod_Sups

Run Script Action



INDUCTIVE
UNIVERSITY

Scheduling Actions - Run Script

[Watch the video](#)

The **Run Script Action** allows you to store your report in a database, provide special email code, or anything else you can think of. Run Script exposes the function `handleFinishedReport()` which gives you the report name and path, a mapping of the report parameters and datasets, and the bytes in whatever format you want.

Here are a list of property descriptions for the **Run Script Action**.

Property Name	Description
Run Script	An area where a script can be created to do something at the scheduled time.
Format	The file format that the reportBytes parameter should be.

A screenshot of the 'Run Script' action configuration window. The window has three tabs: 'Schedule', 'Parameters', and 'Actions'. The 'Actions' tab is active, showing a list of actions on the left with 'Run Script' selected. The main area displays the Python code for the `handleFinishedReport` function. The code is as follows:

```
def handleFinishedReport(reportName, reportPath, dataMap, reportBytes):  
    """  
    Provides an opportunity to perform script-based manipulation, review or  
    action on a generated Report following its execution. Scheduled Report  
    Actions execute on the Ignition Gateway and will have Gateway scope, file  
    paths, etc.  
  
    Arguments:  
        reportName: The name of the report for which this script should run  
        reportPath: The path of the report in your project  
        dataMap: A PyDictionary containing the data Objects that were supplied  
                  to the report  
        reportBytes: The generated Report's byte array in the file format  
                     specified in the Format selection box  
    """  
1  
2
```

On the right side of the code editor, there is a 'Format' dropdown menu currently set to 'PDF'.

Arguments

The `handleFinishedReport` method has the following arguments:

Argument Name	Data Type	Description
<code>reportName</code>	String	The name of the report for which this script should run.
<code>reportPath</code>	String	The path to the report in your project
<code>dataMap</code>	PyDictionary	The Python Dictionary containing the Parameters and Data Sources that were supplied to the report. This argument allows you to directly access Parameters and Data Sources in the report. Note that once <code>handleFinishedReport()</code> has been called, the report has already been generated, so changing the parameters from this function will not alter the resulting report. Instead, parameters should be altered from the Parameters tab.
<code>reportBytes</code>	byte[]	The report, presented in a byte array. The format of the report depends on the format specified in the Format dropdown list.

DataMap

The dataMap argument contains the raw data that was used to generate a report. Below is a demonstration using dataMap.

```
# The dataMap argument is simply a Python Dictionary with the name of each
Parameter and Data Source
# acting as a key.

# Assuming a Report Parameter named 'shift', the value of 'shift' may be accessed
with the following
dataMap['shift']

# Similar syntax may be used to extract the value from a Data Source.
data = dataMap['myDataSource']

# Rows objects, while similar in nature to a dictionary, are different objects.

# Individual rows in the Data Source may be accessed by index.
firstRow = data[0]

# getKeys() may be called on a row to list all of the column headers in the row.
firstHeader = firstRow.getKeys()[0]

# getKeyValue() may be used to access the value of a column in the row.
firstColumnInRow = firstRow.getKeyValue(firstHeader)
```

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

SQL Bridge (Transaction Groups)

What are Transaction Groups?

Simple data logging and much more! Transaction Groups give you the ability to **drag and drop** tags to start **synchronizing data** between **PLCs** and **databases**. You can use them to easily log from PLCs to the database, move data from the database back to PLCs, and even keep the two synchronized.

Originally conceived as an easy data storage method, Transaction Groups have long since been a core feature of Ignition. In their simplest form, they regularly read values from OPC addresses and store them into a SQL database. While data collection is still their primary use, they have grown in functionality over time.

The screenshot displays the Ignition SQL Bridge configuration window for a Transaction Group named "Tank Levels". The interface is divided into several sections:

- Control Buttons:** "Enabled" (green play icon), "Disabled" (red stop icon), and "Pause" (blue pause icon).
- Basic OPC/Group Items (7):** A table listing the data sources being synchronized.
- Run-Always Expression Items (ignore trigger) (0):** A table for items that run continuously.
- Triggered Expression Items (0):** A table for items triggered by specific events.
- Configuration Panel (Right):** Contains settings for execution scheduling, data source, table name, and storage options.

Item Name	Value	Target Name	Data Type	Properties
B3:0	78	B3_0	Int2	
B3:1	79	B3_1	Int2	
B3:2	77	B3_2	Int2	
B3:3	75	B3_3	Int2	
B3:4	78	B3_4	Int2	
B3:5	0	B3_5	Int2	
B3:6	90	B3_6	Int2	

Item Name	Value	Target Name	Data Type	Properties
-----------	-------	-------------	-----------	------------

Item Name	Value	Target Name	Data Type	Properties
-----------	-------	-------------	-----------	------------

Configuration Panel (Right):

- Execution Scheduling:** ☒ Timer ☐ Schedule. Value: 1 minute(s).
- Data source:** <Default>
- Table name:** tank_table
- ☒ Automatically create table
- ☐ Use custom index column: tank_table_ndx
- ☒ Store timestamp to: t_stamp
- ☐ Store quality code to: quality_code
- ☐ Delete records older than: 1 day(s)

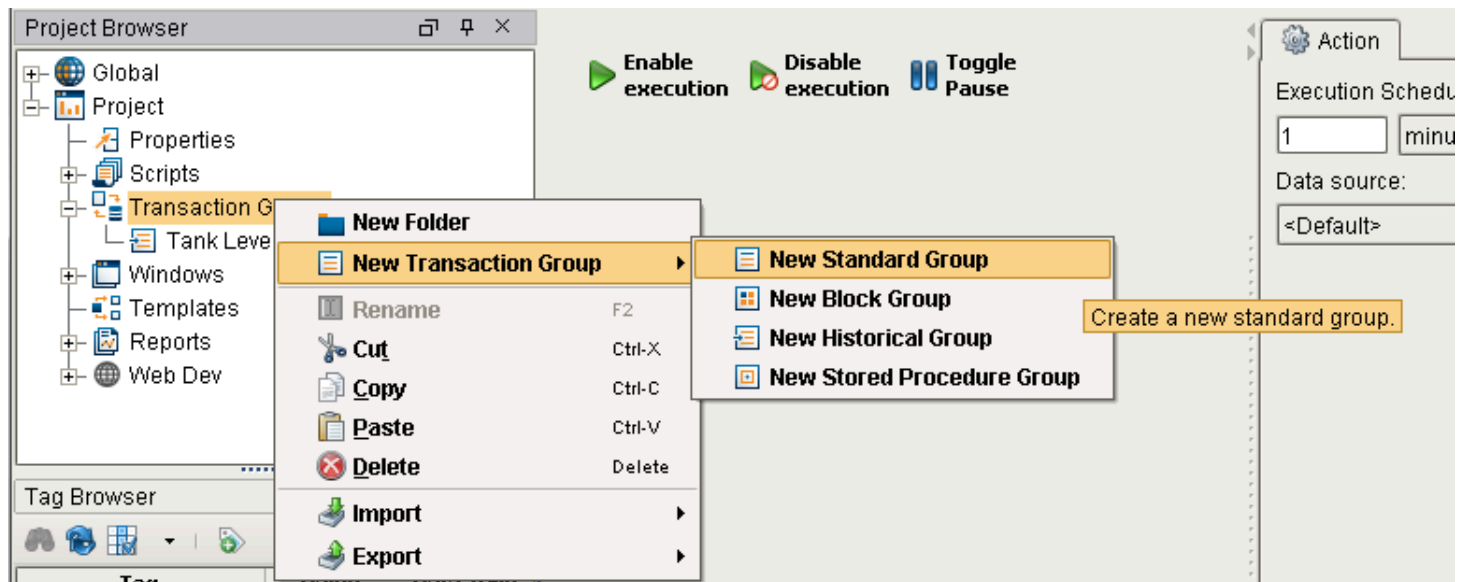
SQL Knowledge Not Required!

Ignition can automatically create and manage the database table for each group. Prior experience writing SQL queries or creating database tables are not required to log data.

B3_0	B3_1	B3_2	t_stamp
78	79	77	2016-06-29 11:33:26
78	79	77	2016-06-29 11:32:26
78	79	77	2016-06-29 11:31:26

What Can Transaction Groups Do?

There are four types of Transaction Groups, and they all handle data a little differently. Regardless of which group you use, they can execute at a set rate or on a schedule. A **trigger** can be used to determine when the group should record and you can even use Ignition's expression language in the trigger to allow complex logic to determine when logging occurs, making precision execution easy.



Historical Logging Made Easy

Historical Groups quickly and easily store data from the plant floor into any kind of SQL database! Items from any or all devices can be included in the same group, just drag a few tags over and start the group running. Ignition will log the data until you tell it to stop.

Ramp History

Running

Enabled
 Disabled
 Pause

Basic OPC/Group Items (5)

Item Name	Value	Target Name	Data Type	Properties
Ramp0	524.040	Ramp0	Float8	
Ramp1	43.030	Ramp1	Float8	
Ramp2	99.040	Ramp2	Float8	
Ramp3	74.545	Ramp3	Float8	
Ramp4	243.030	Ramp4	Float8	

Action Trigger Options

Execution Scheduling:

☒ Timer ☐ Schedule

1 second(s)

Data source:

<Default>

Table name:

ramp_historical

Database Query Browser

```
SELECT * FROM ramp_historical
```

☒ Limit SELECT to: 1000 rows

Resultset 1

ramp_historical...	Ramp0	Ramp1	Ramp2	Ramp3	Ramp4	t_stamp
163	149.333	62	124.333	253	362	2016-07-01 10:47:07
164	162.667	72	137.667	268	372	2016-07-01 10:47:08
165	176	82	151	283	382	2016-07-01 10:47:09
166	189.347	92.01	164.347	298.015	392.01	2016-07-01 10:47:10
167	202.68	2.01	-22.32	13.015	2.01	2016-07-01 10:47:11

Database and OPC Synchronization

Standard Groups are the most flexible group. They are capable of not only storing OPC values in the database, but can also write database values to OPC addresses or synchronize data changes between both the database and PLC. With this group you can create true realtime value tables in the database, and allow anything that can talk to the database to push values to a PLC. This is often used to create **Recipe** systems where the recipe values are stored in the database, and a user can select a recipe to write all your settings directly to Tags. Changing recipes is as easy as changing a tag value or selecting a name.

Line 1 Recipe

Running

Enabled
 Disabled
 Pause

Basic OPC/Group Items (4)

Item Name	Value	Mode	Target Name	Data Type	Properties
Product Code	Waiting	—	Read-only	Int2	
Conveyor Speed	Waiting	←	conveyor_1_speed	Float4	
Arm Speed	Waiting	←	arm_1_speed	Float4	
Feed Rate	Waiting	←	line_1_feed_rpm	Float4	

Rapidly Store Large Blocks of Data

Transfer large amounts of data very efficiently with the **Block Group**. This groups allows you to send whole arrays of data to and from the database. It works just like the Standard group, but on a much larger scale.

T4 ACC

Running

Enabled

Disabled

Pause

Item View

Block View

Block Items (4)

Item Name	Value	Mode	Target Name	Data Type	Prope...	Size
Item_T4_0		—	T4_0_ACC	String		2
ns=1;s=[SLC]_Meta:T4/T4:0/T4:0.ACC	0					
ns=1;s=[SLC]_Meta:T4/T4:0/T4:0.DN	false					
Item_T4_1		—	T4_3_ACC	String		2
Item_T4_2		—	T4_2_ACC	String		2
Item_T4_3		—	T4_1_ACC	String		2

Stored Procedures

Give your IT department total control of your databases. The **Stored Procedure Group** allows you to use PLC data as inputs and outputs for your existing Stored Procedures. This means IT can live happily knowing that they have control over how data is entered and returned from the database.

Calc Line Downtime

Running

Enabled

Disabled

Pause

Basic OPC/Group Items (2)

Item Name	Value	Target Name	Output	Data Ty...	Propert...
Line Number	3	linenum	None	Int2	
Minutes Down	82	Read-only	minutes	Int2	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Value	Target Name	Data Type	Properties
-----------	-------	-------------	-----------	------------

Action

Trigger

Options

Execution Scheduling:

Timer

Schedule

1

second(s)

Data source:

DB

Store timestamp to:

Store quality code to:

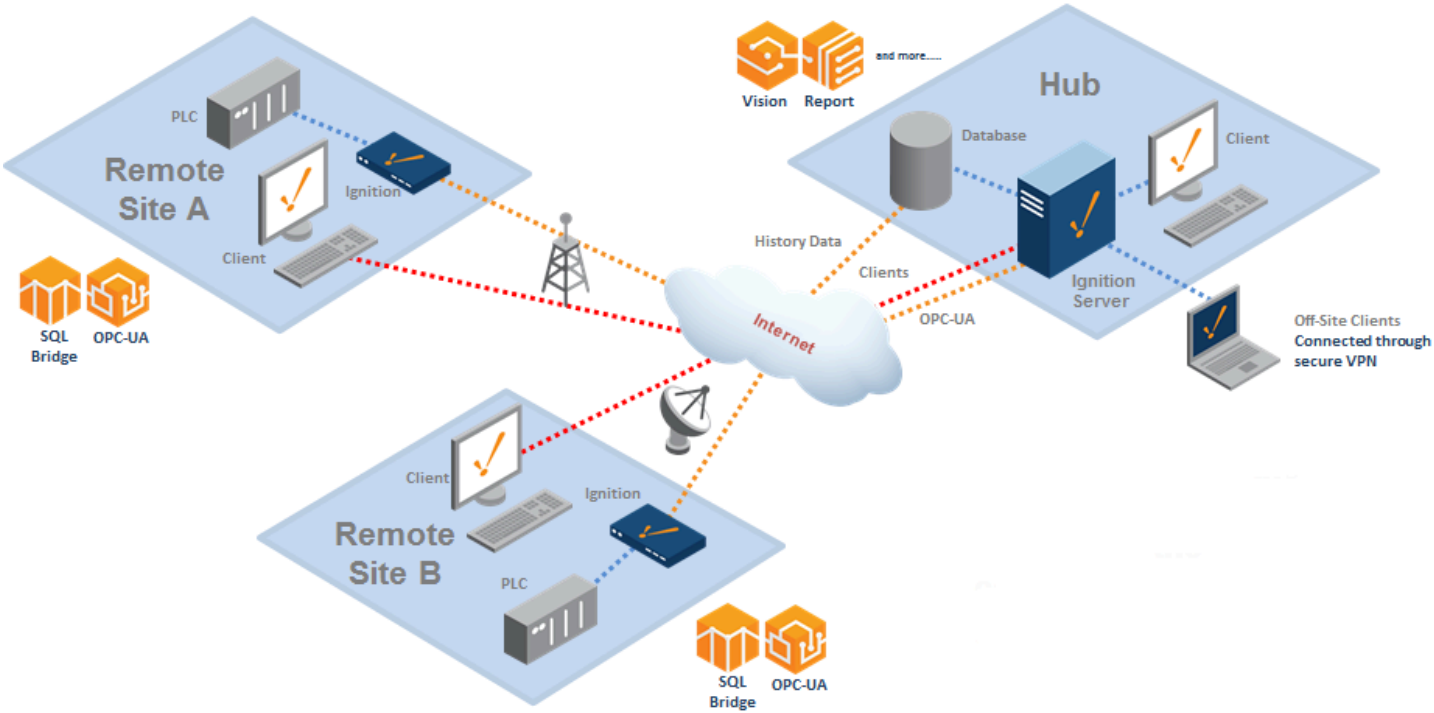
Procedure name:

downtime_minutes

Centralizing Data Collection

In Distributed systems, PLCs can be spread out over great distances. Collecting and centralizing data from each would normally be a massive burden. To combat this problem, Transaction Groups are used as the cornerstone of our **Hub and Spoke** architecture. Historical Groups can be applied locally to each PLC for a minimal cost, and forward all data into a single, central, database.

Remote Site with Central Hub
Remote Logging



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Transaction Group Examples

Transaction Groups

There are four basic types of Transaction Groups that can be used in Ignition:

- **Standard:** The heart of bi-directional data storage and management
- **Historical:** Simple historical trending
- **Block:** Efficient large scale data storage
- **Stored Procedure:** Interact with existing protected data systems

This Section has examples for each type of group and shows the different ways that you can use them. For a more complete understanding of how the parts of each group works, see [Understanding Transaction Groups](#).

Standard Group

The **Standard Group** is the most flexible group. It is commonly used as a bi-directional sync between your PLCs and databases. In addition to this, it can also be used to push data in either direction. This means the Standard Group can be used to store historical data, [add to/update existing tables](#), create [recipe management tools](#), and more!

Bi Directional

Running

Basic OPC/Group Items (8)

Item Name	Value	Mode	Target Name	Data Type	Properties
accumulatorLevel	0	—	accumulatorLevel	Int4	
ambientHum	22.008	—	ambientHum	Float4	
ambientTemp	24.287	—	ambientTemp	Float4	
dischargePressure	43.911	—	dischargePressure	Float4	
dischargeTemp	130	—	dischargeTemp	Int4	
receiverLevel	81	—	receiverLevel	Int4	
valveDischarge	0	—	valveDischarge	Int4	
valveKing	0	—	valveKing	Int4	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Value	Target Name	Data Type	Properties
-----------	-------	-------------	-----------	------------

Triggered Expression Items (1)

Item Name	Value	Target Name	Data Type	Properties
TempC	-4	Read-only	Int4	

Action
 Trigger
 Options

Execution Scheduling:

☒ Timer ☐ Schedule

1 second(s)

Update mode:

Bi-directional OPC wins

Data source:

<Default>

Table name:

status

☒ Automatically create table

☐ Use custom index column: status_ndx

☒ Store timestamp to: t_stamp

☐ Store quality code to: quality_code

☐ Delete records older than: 1 day(s)

Table action

☐ insert new row

☒ update/select

☒ first ☐ last ☐ custom ☐ key/value pairs

Historical Group

The **Historical Group** is the most straightforward and simplest to use. It will take OPC data and store it as history in a database.

History

Running

Basic OPC/Group Items (9)

Item Name	Value	Target Name	Data Type	Properties
Realistic1	11.406	Realistic1	Float8	
Realistic2	50.000	Realistic2	Float8	
Realistic3	-50.000	Realistic3	Float8	
Realistic4	-24.416	Realistic4	Float8	
Realistic5	-41.939	Realistic5	Float8	
Realistic6	-39.732	Realistic6	Float8	
Realistic7	36.584	Realistic7	Float8	
Realistic8	-38.485	Realistic8	Float8	
Realistic9	-43.396	Realistic9	Float8	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Value	Target Name	Data Type	Properties
-----------	-------	-------------	-----------	------------

Triggered Expression Items (0)

Item Name	Value	Target Name	Data Type	Properties
-----------	-------	-------------	-----------	------------

Action
 Trigger
 Options

Execution Scheduling:

☒ Timer ☐ Schedule

1 second(s)

Data source:

<Default>

Table name:

history

☒ Automatically create table

☐ Use custom index column: history_ndx

☒ Store timestamp to: t_stamp

☐ Store quality code to: quality_code

☐ Delete records older than: 1 day(s)

Block Group

The **Block Group** is used to efficiently store large amounts of data in "blocks" or chunks of similar data in the database. This is perfect if you have many devices with the same tags in them.

Block Group

Running

Enabled

Disabled

Pause

Item View

Block View

Block Items (3)

Item Name	Value	Mode	Target Name	Data Type	Properties	Size
Item_Realistic			Realistic	Float8		10
Item_Ramp			Ramp	Float8		10
Item_Sine			Sine	Float8		10

Basic OPC/Group Items (0)

Item Name	Value	Target Name	Data Type	Properties
-----------	-------	-------------	-----------	------------

Run-Always Expression Items (ignore trigger) (0)

Item Name	Value	Target Name	Data Type	Properties
-----------	-------	-------------	-----------	------------

Triggered Expression Items (0)

Item Name	Value	Target Name	Data Type	Properties
-----------	-------	-------------	-----------	------------

Action

Trigger

Options

Timer

Schedule

1

second(s)

Update mode:

OPC to DB

Data source:

<Default>

Table name:

block_group

☒ Automatically create table

☒ Automatically create rows

☐ Use custom index column:

block_group_ndx

☒ Store timestamp to:

t_stamp

☐ Store quality code to:

quality_code

☒ Store row id to:

row_id

☒ Store block id to:

block_id

☐ Delete records older than:

1

day(s)

Table action

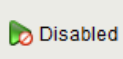
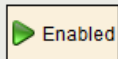
insert new block

Stored Procedure Group

The **Stored Procedure Group** is used in a similar way as the Historical Group except instead of selecting a target table, you select a target Stored Procedure and use the tags as inputs and outputs. The only major requirement that makes this group different is that a stored procedure must already exist in your database for this group to work.

Stored Procedure

Paused



Basic OPC/Group Items (6)

Item Name	Value	Target Name	Output	Data Type	Properties
WriteableDouble1	N/A	input1	None	Float8	
WriteableDouble2	N/A	Read-only	output1	Float8	
WriteableFloat1	N/A	input2	None	Float4	
WriteableFloat2	N/A	Read-only	output2	Float4	
WriteableInteger1	N/A	input3	None	Int4	
WriteableInteger2	N/A	Read-only	output3	Int4	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Value	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Value	Target Name	Data Type	Properties

Action

Trigger

Options

Execution Scheduling:

☒ Timer ☐ Schedule

1 second(s)

Data source:

<Default>

☐ Store timestamp to:

☐ Store quality code to:

Procedure name:

my_procedure

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Basic Historical Group



INDUCTIVE
UNIVERSITY

Basic Historical Group

[Watch the video](#)

A basic historical transaction group is any transaction group that logs data to a database.

To create a Historical Group

1. In the Project Browser, right-click on the **Transaction Groups**, and select **New Historical Group**.
2. Select tags from the OPC Browser and drag them into the **Basic OPC/Group Items** area of the transaction group.
The target databases are represented in the **Target Name** column and when the group executes, the data will be stored in the columns of these names.
3. Go to the **Action** tab and select **Timer** to specify how often the group executes.

There are other options that help configure the data as it enters the database. For example, having an additional column for the time stamp.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Scheduled Group



INDUCTIVE
UNIVERSITY

Scheduled Group

[Watch the video](#)

You can log data to a database at specific times using the execution scheduling. You create the **Scheduled** group the same way as you create a basic **Historical** group. But instead of executing on a **Timer**, you select the **Schedule** radio button, and enter the times in the text box and the rate.

The following table shows examples of how you can enter numbers in the text box:

Times	Rate	Result
8, 11, 5pm	n/a	Group will execute 3 times at 8 in the morning, 11 in the morning, and finally at 5pm.
8	n/a	Group will execute once at 8 in the morning.
8-10	1 hour	Group will execute at 8, 9, and 10 in the morning.
8-10, 17	1 hour	Group will execute at 8, 9, and 10 in the morning and once at 5pm.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Realtime Group



INDUCTIVE
UNIVERSITY

Realtime Group

[Watch the video](#)

Using transaction groups to update specific rows in a database is an easy way to expose OPC data to systems that cannot access OPC server but can query a SQL database.

To update the first row in a database

1. Create a new Standard group and drag your OPC tags into the **Basic OPC/Group Items** area.
2. On the **Action** tab, select the appropriate Timer, Data source, Table name, and other various settings.
3. In the **Table action** area of the **Action** tab, select **update/select**, and **first** which will continue to update the first row of the database according to the timer.
To update a specific row rather than the first row in the database, select the **custom** or the **key/value pairs** options.
4. Select the group, click **Enabled** and save the project to start the group.

The transaction group will only write the values that have changed to the database thus reducing the overhead.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Bi-directional Group



INDUCTIVE
UNIVERSITY

Bi-directional Group

[Watch the video](#)

A bidirectional transaction is any transaction that synchronizes data between sources. It is not a transaction that inserts values into a database like a historical group. Bidirectional transactions are useful in a number of cases. A bidirectional transaction can update a database table with the values from an OPC location thus making the values accessible to systems that have the ability to read from databases but not OPC locations. Likewise, bidirectional transactions allow external systems to update OPC locations by interacting with the database.

A transaction group designed for bi-direction synchronization evaluates and handles the updates at the group's rate, for example every second. In the event that both the database and the OPC locations experience a change between the executions of the transaction group the group decides to synchronize the database and the OPC location in accordance with the selected update mode. The update modes can also be individually tailored for each tag and database column by choosing the correct update mode in the Mode column of the Basic OPC/Group Items section.

To create an bidirectional transaction group

1. Create a **New Standard Group** and drag tags from the OPC server or the Tag Browser into the group's **Basic OPC/Group Items**.
2. Go to the **Action** tab, and set the following:
 - **Update mode:** either **Bi-directional OPC wins** or **Bi-directional DB wins**
 - **Data source:** the appropriate source
 - **Table name:** the appropriate name
 - **Automatically create table:** select the check box so a table is automatically created if the table name does not already exist **Table Action** area: select **update/select** and **first**. You can also synchronize the OPC and database locations based on a condition by choosing **custom** or the **key/value pairs** options.

3. Click **Enabled** and save the project to start the group.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Stored Procedure Group



INDUCTIVE
UNIVERSITY

Stored Procedure Group

[Watch the video](#)

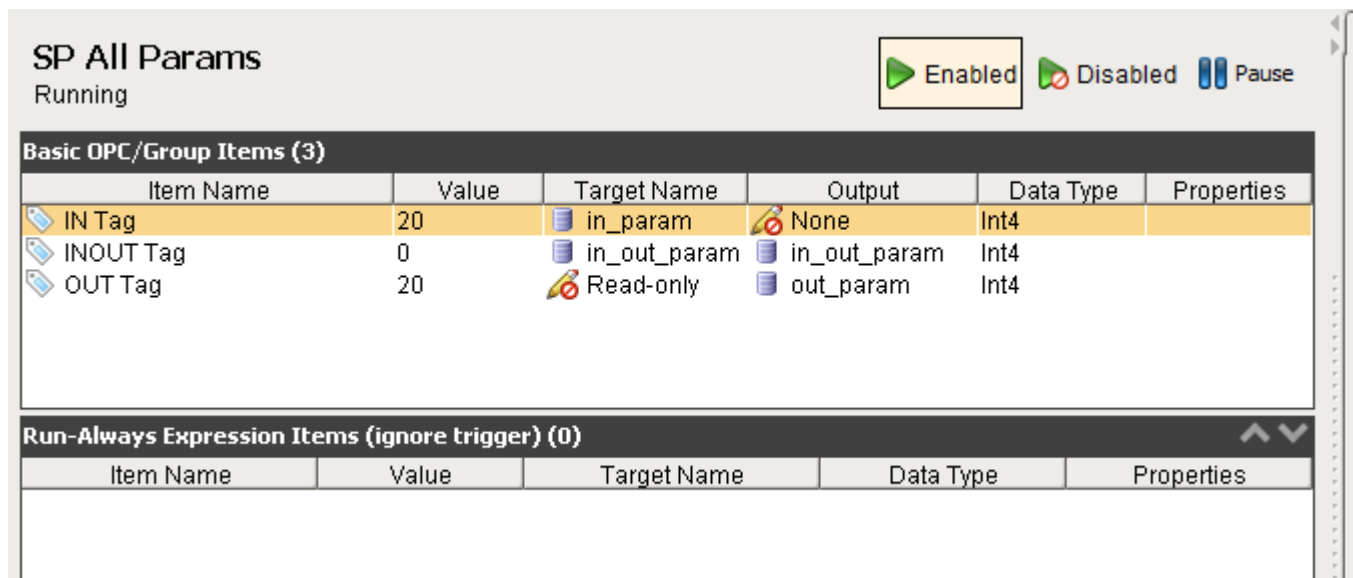
A stored procedure group will interact with a database stored procedure. A stored procedure in a database is like a function. The procedure has a name and it may or may not take parameters. Once the procedure is called and the parameters passed into the procedure, the query executes on the database server and a the resultset is returned. The actual text of the query is stored and managed on the database server. Stored procedure provide an increased level of security and maintenance that make them ideal for certain scenarios.

Parameters in the Stored Procedure Group

When using a Stored Procedure Group, parameters may be configured to each item based on the type of the parameter:

- The **Target Name** column is used for writing, so specifying an IN or INOUT parameters under this column will have the item try to write its value to the parameter
- The **Output** column is used to move the value of an OUT or INOUT parameter into an item in the group. Note that if an item in a group is configured to reference an OUT parameters, its **Target Name** value should be set to **Read-Only**

A general reference of how items should be configured based on the type of the associated parameter can be seen below.



This is a very simple scenario. Often stored procedures contain very long or complicated queries. Lets say you want to use a transaction group to call this stored procedure and then to handle the result by writing to a tag location.

To use a transaction group to call a stored procedure

1. Create a new **Stored Procedure** transaction group.
2. Drag two tags into the **Basic OPC/Group Items** section of the group. The two tags must be of the same datatype as the stored procedure's parameters. In this case the two tags will be a string and an integer type.
3. Go to the **Action** tab of the transaction group, select the appropriate **Data source**. Select the procedure name from the procedure name drop down menu.
4. In the **Basic OPC/Group Items** section, from the **Target Name** column dropdown of the string tag, select the input parameter, and select None for the output.
5. In the **Basic OPC/Group Items** section, from the **Target Name** column drop down of the integer tag, select Read-only, and select the output parameter for the output.
6. Click **Enabled** and save the project to start the transaction group.

NOTE

When running a Stored Procedure from an Oracle database, you cannot use named parameters. Instead you must use the index (number) arguments. IE: 1 is the first argument, 2 is the second, etc.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Block Group



INDUCTIVE
UNIVERSITY

Block Group

[Watch the video](#)

The Block group is a type of transaction group that stores data vertically. Whereas, a Standard group stores the information horizontally in a single row. Block groups share many of the same features as the Standard group. They can be bidirectional, insert into a database, or simply update the database. All the rows in a Block group are associated with a single database transaction therefore the process of writing to the database is very efficient.

To create a Block group

1. Create a new **Block** transaction group.
2. Drag a tag folder into the **Block Items** section of the new transaction group.
3. Select the item in the Block group, right-click and select **Edit**.
4. Change the **Name**, enter the **Target Name** to anything appropriate, and configure the remainder of the group settings under the **Action** tab.
5. Select the group, click **Enabled** and save the project to start the group.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Recipe Group



INDUCTIVE
UNIVERSITY

Recipe Group

[Watch the video](#)

You can use Standard or Block Transaction Groups to create recipe management system with bi-directional communication to pass database information to the PLC when requested. The main advantage of using transaction groups to design the recipe management group is that the group will write to all the OPC locations.

To use transaction groups to create recipes

1. Create a new **Standard** transaction group.
2. Drag a set of tags into the group's **Basic OPC/Group Items**.
One of these tags should be a recipe index tag and needs to be read-only.
3. Change the recipe index tag to read-only by selecting **Read-only** from the drop-down in the **Target Name** column.
4. Go to the **Action** tab, and set the following:
 - **Update mode**: choose **OPC to DB**.
 - **Table name**: select the table name that contains the recipes. This table should have an index.
 - **Use custom index column**: select the check box and choose the column that is the index from the table.
 - **Store timestamp to a column**: deselect the check box.
5. In the **Table action** area of the **Action** tab, click **update/select** and **custom** to create a custom clause that references the read-only tag. The read-only tag and the name of the index column of the database must be the same. Do not include the word "where" because it is assumed to be part of the enclosed statement.

6. Click **Enable** to enable the group and save the project to start the transaction group.
7. Test the recipe group by entering a new value into the recipe number (for example, the tag that is mapped to the index of the recipe database). The recipe values will download to the appropriate tags.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Update or Insert Group



INDUCTIVE
UNIVERSITY

Update or Insert Group

[Watch the video](#)

You can update a row or insert a new row into the database when a key pair combination does not exist. This eliminates the need to have a database that has every possible option considered in its original design. Because of the **insert row when not present** setting, the group will insert a new record whenever the designated ID doesn't exist. Afterwards, it will update the rows in the table that are associated with the key/value references as shown in this example.

To update or insert a new row into the database

1. Create a **New Standard Group**.
2. Drag a group of tags into the groups **Basic OPC/Group Items** section.
3. Change one of the tags to be read-only by selecting **Read Only** from the tag's **Target Name** column.
4. Select the appropriate settings under the group's **Action** tab.
5. In the group's **Action** tab, in the **Table action** area, select the **update/select** radio button.
6. In addition, select the **key/value pairs** radio button.
 - For the **Column** select the database table ID column.
 - For the **Value** column, select the read-only tag.
 - Select **Insert row when not present** check box at the bottom of the **Table action** area.
7. Click **Enable** and save the project to start the group.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Understanding Transaction Groups

Transaction Groups



INDUCTIVE
UNIVERSITY

About Transaction Groups

[Watch the video](#)

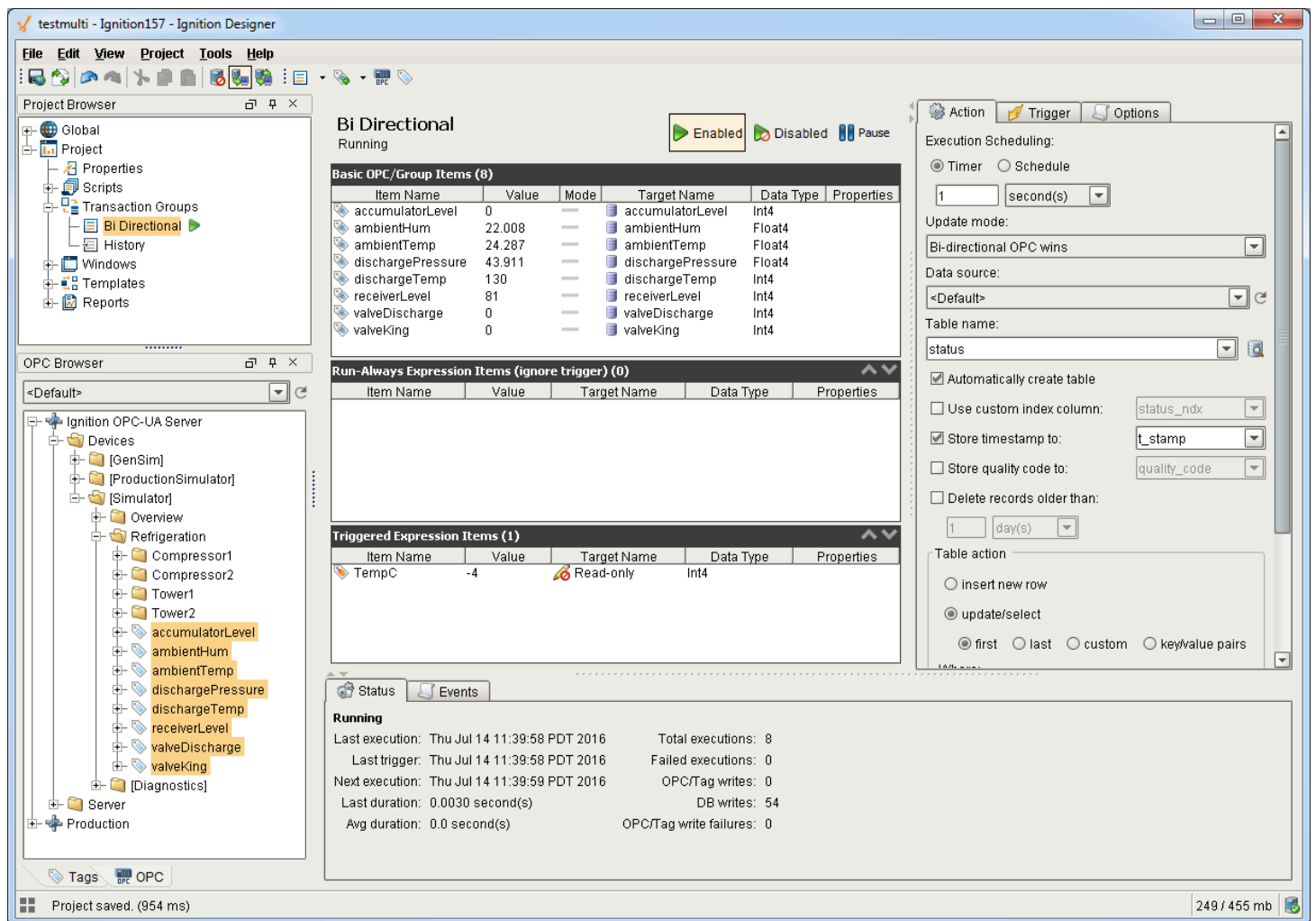
Transaction Groups are the heart of the SQL Bridge module. They are units of execution that perform a variety of actions, such as storing data historically, synchronizing database values to OPC, or loading recipe values. A variety of group types, items types, and options means that Transaction Groups can be configured to accomplish almost any task.

Transaction groups are configured in the Ignition Designer. Each Transaction Group is associated with a table in a database Ignition is connected to, and is made up of one or more Items. The group will then execute at a specific interval of time, or on a user-defined schedule. Generally each execution will create a new row in the database table with a separate column for each Item in the group. However, it is possible for some types of Transaction Groups to take values from the database and write to specific Tags.

Additionally, the Transaction Group can be configured to conditionally synchronize values by using a trigger. The trigger is evaluated every execution, and if the trigger condition is met, then synchronization will occur. If the trigger condition has not been met, then the group will wait until the next execution to re-evaluate the trigger.

There are several types of Transaction Groups, (Standard, Block, Historical, and Stored Procedure) and each offers different functionality. For example, the Historical Group allows you to quickly configure a group that reads OPC data and push it to the database. While the additional flexibility of a Standard Group allows you take values from the database and write them to a PLC.

Transaction Groups allow you to perform tasks such as: database to OPC synchronization, recipe management, and historical data logging.



The Transaction Group Workspace

Transaction groups are edited through the Ignition Designer. When a group is selected, you are presented with the transaction group workspace.

The workspace is broken into several parts:

1. Title bar - Shows the name of the currently selected group, as well as options to set it as Enabled or Disabled, and to Pause, if it's currently executing.
2. Item configuration - Shows all of the items configured in the selected group. Many settings can be modified directly through the display, the rest by double-clicking the item, or selecting "edit" in the context menu.
3. Action / Trigger / Options tabs - Define how and when a group executes. Holds most of the options that apply to the group in general, such as the update rate, and which data connection it uses.
4. Status / Events tabs - Provide information about the executing group, including the most recent messages that have been generated.

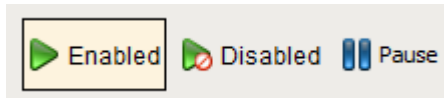
Items (Tags)

Each Item in the Transaction Group consists of several properties, but the key properties are the Value and Target Name.

- **Value:** The Value can be a reference to an OPC value such as a Tag in the Tag Browser. However, you can instead use a direct OPC Item Path without having to create a Tag in Ignition first. Additionally, the Value can be derived using Ignition's Expression Language.
- **Target Name:** In most cases, the Target Name is a column on the database table the Transaction Group is associated with. However, it is possible to have the Target Name 'Read-only'. When set to 'Read-only' the value of the item will not be tied to any columns in the database, but is still visible from the Transaction Group and can be used as a trigger.

Enabling Group Execution

In order for groups to be evaluated, they must first be enabled. This is done by selecting "Enabled" in the group title bar, and then saving the project. The group executing can be stopped by reversing the procedure and selecting "Disabled" before saving. If you want to quickly and temporarily stop the group's evaluation, toggle the "Pause" button. This will prevent execution until the group is unpaused,



or until the system is restarted.

i NOTE

Transaction Groups exist in a project, but they execute in the global Gateway space. This means that once your groups are enabled, they will run even without a client open.

Editing Group Settings

Group settings may be modified at any time, regardless of whether or not the group is executing. Modifications will be applied when the project is saved, and the group will be started or stopped as required. Some changes such as modifying items may cause features like live values to appear to be incorrect. It is therefore important to note the modified icon that appears next to the group, and to save often. If you would prefer to stop the group before making edits you can simply pause the group. Execution will begin again after the project is saved.

How Groups Execute

Generally speaking, groups work on a timer. They are set to run at a certain rate, and at that rate, then check the rest of the settings. If the trigger conditions pass, the group is executed fully. The following

section provides a fuller outline of the execution cycle.

Execution Cycle

All of the groups follow a similar execution cycle. The core evaluation may differ, but the general cycle is the same.

1. Timer executes, group enters execution
2. Is the group paused? Break execution.
3. Is the Gateway part of a redundant pair? If so, is it active? If not active, break execution. Groups only execute on the active node.
4. Evaluate "run-always" items: OPC items, Tag references, and Expression items set to ignore the trigger (or placed in the "run always" section of the configuration window).
5. Is trigger set/active? If there is a trigger defined, but it is not active, break execution.
6. Evaluate "triggered" expression items.
7. If applicable, read values from the database.
8. Execute a comparison between items and their targets.
9. Execute any writes to other Tags or the Database that result from execution.
10. Report alerts (alarms).
11. Acknowledge the trigger, if applicable.
12. Write handshake value, if applicable. If an error occurs at any stage besides the last stage, execution will break and the failure handshake will be written if configured. The group will attempt execution again after the next update rate period.

DANGER

If the group errors are due to a bad database connection, they will need to be manually restarted once the database connection is brought back.

Action Settings

The **action settings** of a group define how often the group will be evaluated, as well as important settings that apply to the group as a whole.

They are found on the **Action** tab, the first of the tabs on the right side of the Transaction Group workspace.

 Action

 Trigger

 Options

Execution Scheduling:

☒ Timer ☐ Schedule

Update mode:

Data source:

Table name:

☒ Automatically create table

☐ Use custom index column:

☒ Store timestamp to:

☐ Store quality code to:

☐ Delete records older than:

Table action

☐ insert new row

☒ update/select

☒ first ☐ last ☐ custom ☐ key/value pairs

Where:

Common Settings

The settings vary for the different types of groups, but a few settings are common to most of them:

Setting	Description
Execution scheduling	How often the group is evaluated. For a number of reasons, the group may not execute during the evaluation. The most common reason is the trigger, but see Execution Cycle above for more possible reasons why evaluation will exit.
Update mode	For groups that support it, sets the default for how items are compared to their targets.
Data source	The data connection to use for the group. Can be Default, which will use the default connection for the project.
Store timestamp	Stores a timestamp along with the data any time the group executes.
Store quality code	Stores an aggregate quality for the group along with the regular data. The aggregate quality is a bit-wise AND of the qualities of the items in the group.

Execution Scheduling

There are two ways to specify when the group should execute: timer and schedule mode.

Timer Mode

In timer mode, the group is evaluated regularly at the provided rate. As mentioned in the previous sections, due to trigger settings, full execution may not occur, but the trigger will at least be evaluated at this rate.

Schedule Mode

With schedule mode, you are providing a list of time (or time ranges) that the group should run at. If the pattern specified includes a time range, at rate must be provided, and the group will execute as in timer mode during that period.

The Schedule Pattern

The schedule is specified as a comma separated list of times or time ranges. You may use the following formats: 24-hour times. Ie. "8:00, 15:00, 21:00", for execution at 8am, 3pm, and 9pm. 12-hour with am/pm (if not specified, "12" is considered noon): "8am, 3pm, 9pm" Ranges, "8am-11am, 3pm-5pm"

Notes

It is allowed for time ranges to span over midnight, such as "9pm - 8am"

When using ranges, the execution times will be aligned to the start time. For example, if you specify a schedule of "9am - 5pm" with a rate of "30 minutes", the group will execute at 9, 9:30, 10, etc., regardless of when it was started. This is a useful difference compared to the Timer Mode, which runs based on when the group was started. For example, if you wanted a group that ran every hour, on the hour, you could specify a 1 hour rate with a range of "0-24".

Trigger and Handshake Settings

The trigger settings determine when a group will actually execute. They are examined each time the group evaluates (according to the update rate of the group). If they pass, the group will run and perform its action against the database.

The trigger settings are the same for all group types. They are found on the second tab (labeled "Trigger"), in the right side of the Transaction Group workspace.

Action
 Trigger
 Options

☒ Only evaluate when values have changed.

Tags to watch for change:

All tags

Select tags

☐ Execute this group on a trigger

Trigger on item:

☐ Only execute once while trigger is active
 ☐ Reset trigger after execution
 ☐ Prevent trigger caused by group start

Trigger conditions:

☐ is != 0 (or true)
 ☒ is = 0 (or false)
 ☐ is active:

>

0

 non-active:

<=

1

☐ Active on value change

☐ Write handshake on success
 Set:

 To value:

0

☐ Write handshake on failure
 Set:

 To value:

0

Only Execute When Values Have Changed (asynchronous trigger)

These settings are evaluated first. If set, the group will examine whether the values in the specified Tags have changed, and if not, will exit evaluation.

It is possible to monitor all Run-Always Tags in the group, or only specific ones.

Execute this Group on a Trigger

Enables trigger on a specific item in the group. The trigger item can be any Run-Always item, such as an OPC item, Tag reference, or an Expression item set to "Run-Always" mode.

In addition to the numeric settings that define the trigger, there are several other options:

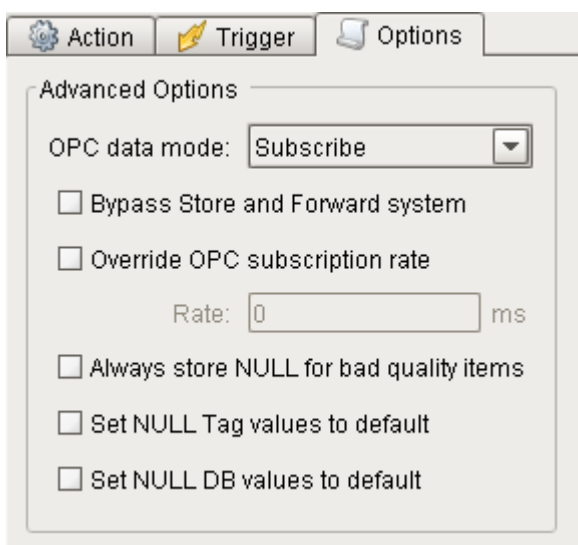
- **Only execute once while trigger is active** - The group will only execute once when the trigger goes into an active state, and will not execute again until the trigger goes inactive first. If unselected, the group will execute each time the trigger conditions evaluate to true.
- **Reset trigger after execution** - If using the "!=0" or "=0" trigger modes, the trigger can be set to write an opposite value after the group has executed successfully. This is useful for relaying the execution back to the PLC.
- **Prevent trigger caused by group start** - If selected, the group will not execute if the trigger is active on the first evaluation of the group. In the course of designing a group, it is common to stop and start it many times, and sometimes it is not desirable to have the group execute as a result of this. Selecting this option will prevent these executions, as well as executions caused by system restarts.

Handshake Settings

Group handshakes are also defined on the trigger tab. It is possible to specify both a success and failure handshake. The success handshake will write the specified value to the given item when the group has finished all other triggered execution without error.

The failure handshake, on the other hand, will be written when the group execution is cut short due to an error, such as an error writing to the database or an item.

Advanced Settings



The screenshot shows a software interface with three tabs: 'Action', 'Trigger', and 'Options'. The 'Options' tab is selected. Below the tabs is a section titled 'Advanced Options'. Inside this section, there is a dropdown menu for 'OPC data mode' set to 'Subscribe'. Below this are several checkboxes and a text field:

- ☐ Bypass Store and Forward system
- ☐ Override OPC subscription rate
Rate: ms
- ☐ Always store NULL for bad quality items
- ☐ Set NULL Tag values to default
- ☐ Set NULL DB values to default

Transaction groups offer several advanced settings that affect how execution occurs. These settings can be found under the Options tab for a group.

Setting	Description
OPC Data Mode	Modifies how the group receives data from OPC. Options are: • Subscribe: Data points are registered with the OPC server, and data is received by the group on-change. This is the default setting and generally offers the best performance, as it reduces unnecessary data flow and allows the OPC server to optimize reads. Note that data is received by the group asynchronously, meaning that it can arrive at any time. When the group executes, it "snapshots" the last values received and uses those during evaluation. If some values arrive after execution begins, they will not be used until the following execution cycle. • Read: Each time the group executes it will first read the values of OPC items from the server. This operation takes more time and involves more overhead than subscribed evaluation, but ensures that all values are updated together with the latest values. It is therefore commonly used with batching situations, where all of the data depends on each other and must be updated together. It's worth noting that when using an OPC item as the trigger, the item will be subscribed, and the rest of the values read when the trigger condition occurs. Note: This option was previously referred to as "polled reads" in earlier versions of the software.
Bypass Store and Forward System	Only applicable to groups that insert rows into the database. Causes groups to target the database directly instead of going through the store-and-forward system. If the connection becomes unavailable, the group will report errors instead of logging data to the cache.
Override OPC subscription rate	Specifies the rate at which OPC items in the group will be subscribed. These items are normally subscribed at the rate of the group, but by modifying this setting it is possible to request updates at a faster or slower rate.
Always store NULL for bad quality items	With this option set to True, it will force the group to store a NULL value when the item has a bad quality, instead of writing the bad quality value.
Set NULL DB Values to Default	New in 7.9.7 If a NULL is read from the database, it will instead use a default value to write to the Tag, depending on the type. This can prevent

Setting	Description
	errors for OPC Tags that do not accept NULL values. Not available in a Historical Group. A list of default values can be found in the Set NULL DB Values table below. New in 7.9.18 As of 7.9.18, enabling the Set DB Values to Default setting on Block Groups will clear the latched value, setting the item to a default if the corresponding database value is Null.
Set NULL Tag Values to Default	New in 7.9.7 If a NULL is read from the Tag, it will instead use a default value to write to the database, depending on the type. This can prevent errors for database columns that do not accept NULL values. The default values are the same as the table above.

Set NULL DB Values

Type	Default Value
Byte	0
Short	0
Integer	0
Long	0
Float	0.0
Double	0.0
Boolean	FALSE
String	" (Empty Sting)
Date/Time	Current Date/Time
Dataset	[0x0] (Empty Dataset)
Array	[] (Empty Array)

Creating a Transaction Group

Transaction groups are used to store history, log events, synchronize databases tables with PLC, perform calculations, and many more data-centric tasks.

To Create a Transaction Group

Let's create a basic History Group and start logging some PLC values to your database.

1. Click on the **Transaction Groups** in the **Project Browser** panel to switch the Designer's workspace to the Transaction Group workspace.
2. Click on **File > New > New Standard Group** to make a new Historical Group. Your new group, named **Group** will be selected. The **OPC Browser** panel is now docked to the lower-left of the screen.
3. Browse your OPC device and drag some OPC tags to the **Basic OPC/Group Items** section. Your group starts out disabled by default.
4. Click the **Enabled** button above the item tables to enable logging.
5. Go to the **Action** tab and change the **Table Name** in your table to something interesting. Right now your group only exists in the Designer.
6. **Save** the project to start the group. Your group is now running and logging data to your database connection. To see the data, you can use the Ignition Designer's built-in database query browser. The easiest way to do this is to click on the Query Browser () button to the right of your group's Table Name field. The Query Browser is a convenient way to directly query your database connection without leaving the Ignition Designer. Of course - you can also use any query browser tools that came with your database.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Types of Groups



INDUCTIVE
UNIVERSITY

Types of Groups

[Watch the video](#)

The SQL Bridge Module provides four different types of transaction groups that you can use in your projects. Each of these different types of groups vary in their operation and use for data logging and database to PLC synchronization.

NOTE

The limited version of SQL Bridge module only has access to the Historical Group

Standard Group

The standard group is called such because it's a flexible, general use group that can be adapted to a variety of situations. The data model is row based, with items mapping to columns and the data corresponding to a specific row of a table.

This group contains items, which may be mapped to the database, or used internally for features such as triggering or handshakes. Items that are mapped to the database target a specific column of a single specific row, chosen according to the group settings. Items can be mapped in a one-way fashion, or bi-directionally, in which the value of the database and the item will be synchronized.

The group may also insert new rows instead of updating a specific row. In this manner, data can be inserted for historical purposes based on a timer, with an optional trigger.

Group Settings

The standard group uses a timer-based execution model shared by all groups, and the normal trigger settings. Additionally, there are several settings specific to the group type.

Property Name	Description
Automatically create table	If the target table does not exist, or does not have all of the required columns, it will be created/modified on group startup. If not selected and the table doesn't match, an error will be generated on startup.
Use custom index column	If selected, you may enter any column name to hold the index. If unselected, the table index will be named <code><table name>_ndx</code> .
Store timestamp to	Specifies whether or not to store a timestamp with the record, and the target column. The timestamp will be generated by the group during execution. For groups that update a row, the timestamp will only be written if any of the values in the group are also written.
Store quality code to	If selected, stores an aggregate quality for the group to the specified column. The aggregate quality is the combined quality of all of the items that write to the table. For more information about quality values, see Data Quality.
Delete records older than	If selected, records in the target table will be deleted after they reach the specified age. This setting is useful for preventing tables from growing in an unbounded manner, which can cause disk space and performance problems over time.

Table Action

This section details how the group interacts with the table on each execution, and is **not** available for the Historical Group type. This means when the Timer or Schedule is active, and the Trigger condition are met. The group can insert a new row, or update the first, last or a custom record.

Property Name	Description
Insert New Row	This option will make the group insert a new record into the database every time the group executes. This is the forced behavior of the Historical group.
Update / Select	This option will either update or select from matching rows based on the option selected below it. The Update Mode property above determines whether an update (OPC to DB), select (DB to OPC), or both (Bi-directional) are used when the group executes. • First: Use the first row in the table. It is not recommended to use this option unless the order of the data in the table is guaranteed. • Last: Use the last row in the table. This is commonly used when another group (or another program) is inserting new rows for us, and we always want to update the most recent record. • Custom: A custom update clause is essentially the WHERE clause of the SQL query that will be generated to read and write the group data. This usually contains a reference to a tag in the group. For example, <code>column_name = {[~]item_name}</code>. • Key/Value Pairs: Used to inject dynamic values in order to create a WHERE clause for you. The table below this option will allow you to enter column names and link them to values (usually tags in the group). This option also has the ability to Insert a new row with the current key/value pair if it was not found.

Typical Uses

Standard groups can be used any time you want to work with a single row of data. This can include:

- Historical logging - set the group to insert new records, and log data historically either on a timer, or as the result of a trigger. Flexible trigger settings and handshakes make it possible to create robust transactions.
- Maintain status tables - Keep a row in the database updated with the current status values. Once in the database, your process data is now available for use by any application that can access a database, dramatically opening up possibilities.
- Manage recipes - Store recipe settings in the database, where you have a virtually unlimited amount of memory. Then, load them into the PLC by mapping DB-to-OPC using a custom where clause with an item binding in order to dynamically select the desired recipe.

- Sync PLCs - Items in the group can be set to target other items, both for one-way and bidirectional syncing. By adding items from multiple PLCs to the group, you can set the items of one PLC to sync with the others. By creating expression items that map from one PLC item to the other, you can manipulate the value before passing it on.

Block Group

The block group is so named because it writes "blocks" of data to a database table, consisting of multiple rows and columns.

A block group contains one or more block items. Each block item maps to a column in the group's table, and then defines any number of values (OPC or SQLTag items) that will be written vertically as rows under that column. The values may be defined in the block item in two modes. The first, List mode, lets a list of value-defining items to be entered. These value items may either be OPC items, SQLTag items, or static values. The second mode, Pattern mode, can be useful when OPC item paths or tag paths contain an incrementing number. You may provide a pattern for the item's path, using the wildcard marker `{?}` to indicate where the number should be inserted.

Block groups are very efficient, and can be used to store massive amounts of data to the database (for example, 100 columns each with 100 rows- 10,000 data points- will often take only a few hundred milliseconds to write, depending on the database). They are also particularly useful for mirroring array values in the database, as each element will appear under a single column, and share the same data type.

Like the standard group, the block group can insert a new block, or update the first, last or a custom block. Additionally, the group can be set to only insert rows that have changed in the block.

In addition to block items, the group can have other OPC items, tag references, and Expression items. These items can be used for triggers, handshakes, etc. They may also target a column to be written, and will write their single value to all rows in the block.

Group Settings

Beyond the differences in the data, namely that the block group works with multiple rows instead of just 1. This group type shares many settings with the [Standard Group](#).

Property Name	Description
Automatically create table	If the target table does not exist, or does not have all of the required columns, it will be created/modified on group startup. If not selected and the table doesn't match, an error will be generated on startup.
Automatically create rows	If the target rows do not exist, they will be created on group execution. If not selected and the rows don't match, no records will be updated.
Use custom index column	If selected, you may enter any column name to hold the index. If unselected, the table index will be named <code><table name>_ndx</code> .
Store timestamp to	Specifies whether or not to store a timestamp with the record, and the target column. The timestamp will be generated by the group during execution. For groups that update a row(s), the timestamp will only be written if any of the values in the group are also written.
Store quality code to	If selected, stores an aggregate quality for the row to the specified column. The aggregate quality is the combined quality of all of the items that write to that row. For more information about quality values, see Data Quality.
Store row id	Each row will be assigned a numeric id, starting at 0. If selected, this id will also be stored with the data.
Store block id	If selected, an incremental block id will be stored along with the data. This number will be 1 greater than the previous block id in the table.
Delete records older than	If selected, records in the target table will be deleted after they reach the specified age. This setting is useful for preventing tables from growing in an unbounded manner, which can cause disk space and performance problems over time.

Table Action

This section details how the group interacts with the table on each execution, and is **not** available for the Historical Group type. This means when the Timer or Schedule is active, and the Trigger condition are met. The group can insert a new row, or update the first, last or a custom record.

Property Name	Description
Insert New Block	If selected, each row of the block will be inserted when the group executes, even if the data has not changed.
Insert changed rows	- This option will only insert the rows that have new data when the group executes. This is particularly useful for recording history for many data points on an "on change" basis, provided there is a unique id column defined. The "store row id" feature is useful for this, as well as the ability to reference the item path in an item's value property.
Update / Select	This option will either update or select from matching rows based on the option selected below it. The Update Mode property above determines whether an update (OPC to DB), select (DB to OPC), or both (Bi-directional) are used when the group executes. • First: Use the first row in the table. It is not recommended to use this option unless the order of the data in the table is guaranteed. • Last: Use the last row in the table. This is commonly used when another group (or another program) is inserting new rows for us, and we always want to update the most recent record. • Custom: Like standard groups, this setting allows you to target a specific section of the table, using SQL WHERE clause syntax, with the ability to bind to dynamic item values. Unlike standard groups, however, the WHERE clause specified should result in enough rows to cover the block. Excess rows will not be written to, but fewer rows will result in a group warning indicating that some data could not be written.

Typical Uses

Block groups are useful in a number of situation where you need to deal with a lot of data efficiently. Mirroring/Synchronizing array values to DB - Arrays are often best stored vertically, which makes them perfect for block groups. Pattern mode makes configuration a breeze by allowing to you specify the array as a pattern, and set the bounds.

- Recipe management - Like standard groups, but when set points are better stored vertically than horizontally.
- Vertical history tables - Group data points by data type (int, float, string), create a copy of the item that stores item path, and then use the insert changed rows option to create your own vertically

storing historical tables. Create additional copies of the block item that refer to quality and timestamp in order to get further information about the data point.

Historical Group

The historical group makes it easy to quickly log data historically to a SQL database. This group inserts records of data into a SQL database, mapping items to columns. Full support for triggering, expression items, hour & event meters and more means that you can also set up complex historical transactions. Unlike the standard group, the historical group cannot update rows, only insert. It also cannot write back to items (besides trigger resets and handshakes).

Group Settings

The settings of the historical group are identical to the settings in the Standard Group, but limited to inserting rows.

Typical Uses

- Basic historical logging - Recording data to a SQL database gives you incredible storage and querying capabilities, and makes your process data available to any application that has DB access. +Shift tracking - Use an expression item to track the current shift based on time, and then trigger off of it to record summary values from the PLC. Use a handshake to tell the PLC to reset the values.

Stored Procedure Group

The stored procedure group lets you quickly map values bi-directionally to the parameters of a stored procedure.

This group is similar to the others in terms of execution, triggering, and item configuration. The primary difference is that unlike the other group types, the target is not a database table, but instead a stored procedure.

Items in the group can be mapped to input (or inout) parameters of the procedure. They also can be bound to output parameters, in which case the value returned from the procedure will be written to the item. Items can be bound to both an input and output at the same time.

Parameters may be specified using either **parameter names** or **numerical index**. That is, in any location where you can specify a parameter, you can either use the name defined in the database, or a 0-indexed value specifying the parameter's place in the function call.

! IMPORTANT

You cannot mix names and indices. That is, you must consistently use one or the other.

If using parameter names, the names should not include any particular identifying character (for example, `?` or `@`, which are used by some databases to specify a parameter).

Group Settings

The stored procedure group's settings look and act the same as those of the Historical Group. The primary difference, of course, is that instead of specifying a table name and column names, you'll specify a Stored Procedure and its parameters.

Property Name	Description
Store timestamp to	Specifies whether or not to store a timestamp with the record, and the target column. The timestamp will be generated by the group during execution. For groups that update a row, the timestamp will only be written if any of the values in the group are also written.
Store quality code to	If selected, stores an aggregate quality for the group to the specified column. The aggregate quality is the combined quality of all of the items that write to the table. For more information about quality values, see Data Quality.
Procedure Name	The name of the Stored Procedure (SP) that you will be using. You must look into the SP definition to see what inputs and outputs are available.

Typical Uses

- Call stored procedures - The stored procedure group is the obvious choice when you want to bind values to a stored procedure. It can also be used to call procedures that take no parameters (though this can also be accomplished from Expression Items/SQLTags).
- Replace RSSQL - The stored procedure group is very popular among users switching from RSSQL, given that application's heavy use of stored procedures.

Known Issues

When using Oracle, you must use indexed parameters.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Group Update Rate



INDUCTIVE
UNIVERSITY

Group Update Rate

[Watch the video](#)

The **Execution Schedule** controls the rate at which the transaction group executes. On the **Action** tab of a group you have selected, under **Execution Scheduling**, there are two options: Timer and Schedule. Timer, executes the group at a certain rate. Schedule, executes the group at specific times. When the Schedule option spans across a period of time, you must specify the rate at which the group executes during that time.

Timer

The Timer acts as the heartbeat of the transaction group. It can run at millisecond, second, minute, hour, or day rates. The Timer specifies the OPC tag subscription rate for the OPC tags. When a Timer is running the transaction group first analyzes the tags inside the **Basic OPC/Group Items** section of the transaction group. Then it looks at the trigger configuration and evaluates for tag changes. Then it evaluates the specific trigger conditions and decides to execute on a trigger. If the triggered condition is true, the transaction group proceeds to the **Triggered Expression Items** section of the transaction group. Only after this flow is complete, will the transaction group interact with the database and for example, insert the tag values into the database.

Schedule

An important difference between the Timer and the Schedule options is that the schedule option will automatically align to the specified start time on the update rate. For example, if the Schedule option is selected and the times are entered as **8,12pm-3pm** and the update rate is **15 minutes**, the transaction group will execute at 8am and then at 12pm, 12:15pm, 12:30pm and so on.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Item Types



INDUCTIVE
UNIVERSITY

Item Types

[Watch the video](#)

Items are the core elements of a group. They are executed, and the values are then used by the group for logic purposes, by other items, and to write to the database. They can be written to from the database or from other items.

Type of Item	Description
OPC Item	Directly subscribed to an OPC server at the rate of the group. Executed by the group, so alarms are evaluated when the group is executed. These items are executed even when the trigger isn't active.
Run-Always Expression Item	Much like an expression tag, can be either a static value, an expression, or a database query. Run-Always expression items are evaluated at each group interval, before the trigger state is evaluated.
Triggered Expression Item	Same as Run-Always expression items, except that they are only executed after the trigger has been evaluated and is active.
Tag Reference	A reference to a Tag. Allows a Tag to be used in a group like any other item type, except that the tag is evaluated by its scan class instead of by the group. For more information, see Tag References vs. OPC Items .

Execution Order

Items generally aren't executed in a reliable order, with the exception of **Expression** items.

Expression items can be ordered using the up and down arrows located to the right of the list where the items are displayed. This can be crucial for performing complex operations that require a specific sequence.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Tag References vs. OPC Items



INDUCTIVE
UNIVERSITY

Tag References vs. OPC Items

[Watch the video](#)

It is easy to confuse the definition and purpose of Tag reference items (Scan Class) and direct OPC items (Timer) in transaction groups, though they have distinct benefits.

Tags may be referenced inside of groups, however it is critical to remember that they are executed by the Ignition Gateway, according to their scan classes, and independent of the group. Adding a tag into a group is like creating a shortcut to that tag. However, once in the group, the item can be used like any other item. That is, it may be mapped bi-directionally to the database, used as a trigger, be the target of another item, and so on. It is even possible to create an hour meter out of the item. Core properties of the tag such as alarming and scaling, however, are defined in the actual tag, not in the group.

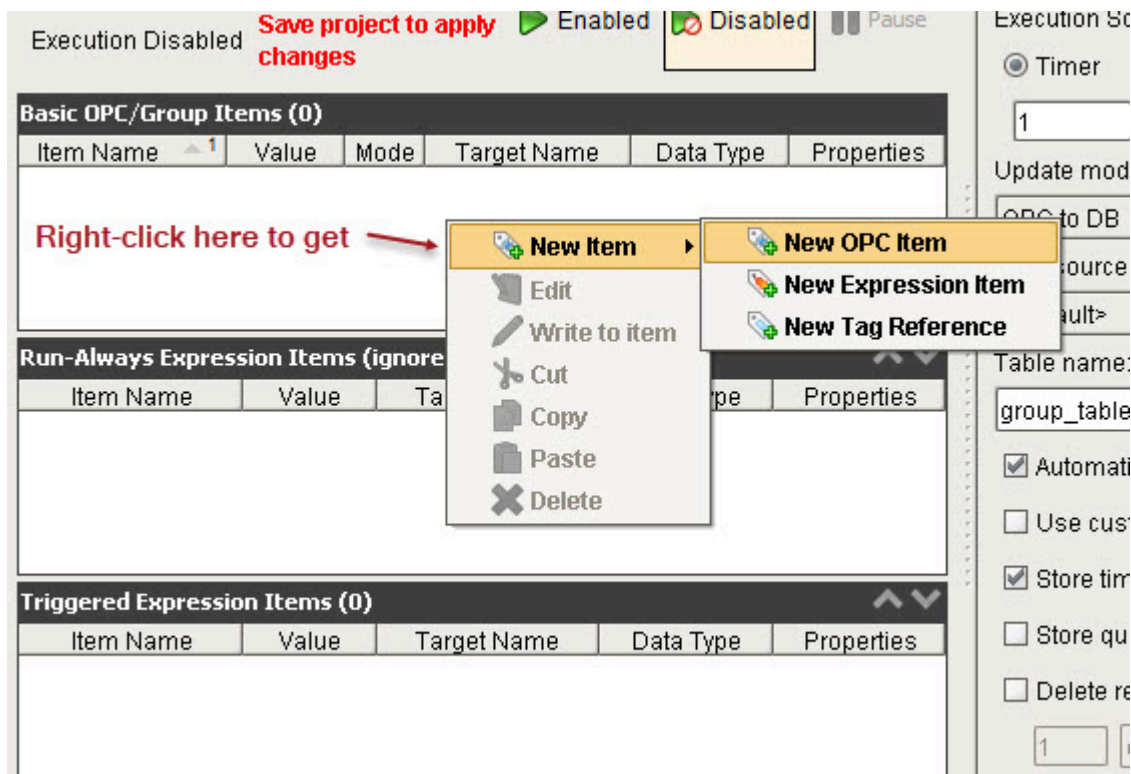
OPC Items in groups (as well as expression items in groups), however, are completely executed by the group. They do not exist outside of the group in which they are defined. They are subscribed and evaluated according to the rate of the group. Generally speaking, it is most common to create OPC items in groups, even if a particular point might already exist in tags. This leads to more understandable group execution, as all evaluation occurs in the group according to the timer and trigger settings. Tag references are useful when it is necessary to have a single value in multiple groups, for example, as a trigger in order to coordinate execution.

OPC Item

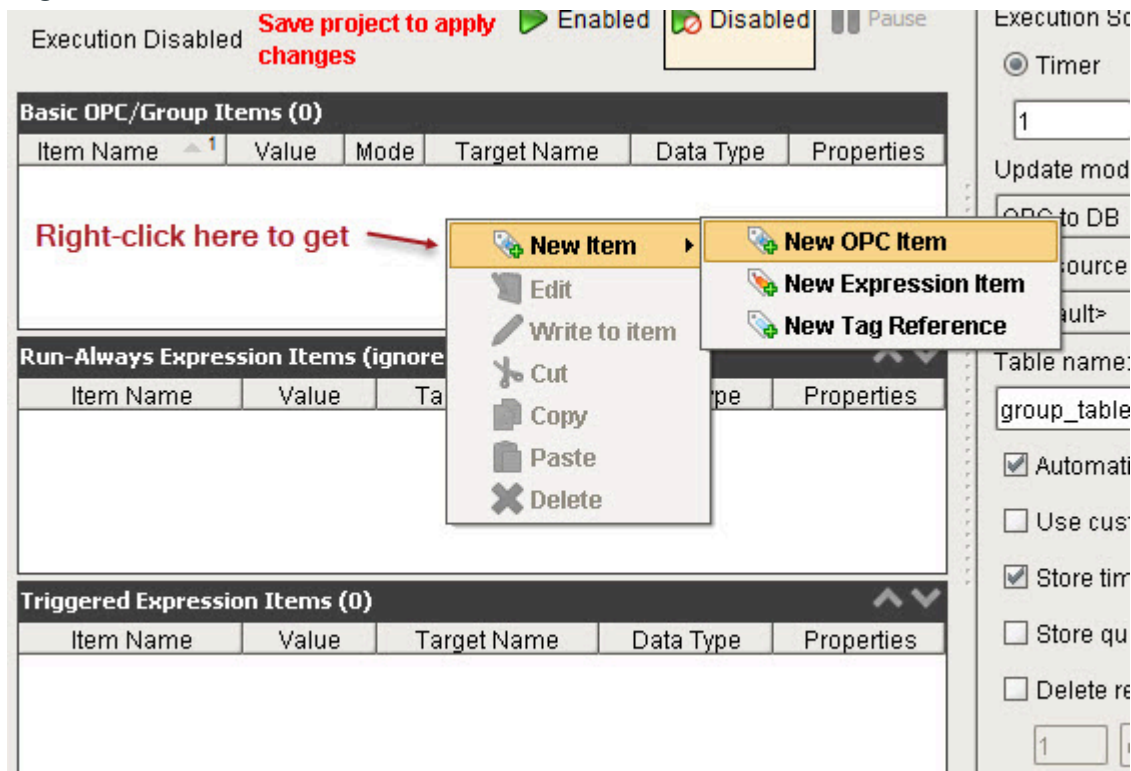
OPC Items are the backbone of a group. They get their values from PLCs and the values are then used by other items in the group and/or to write to the database. They are directly subscribed to an OPC server at the rate of the group and are executed by the group so their alarms are evaluated when the group is executed. These items are executed even when the trigger isn't active.

To Create a New OPC Item

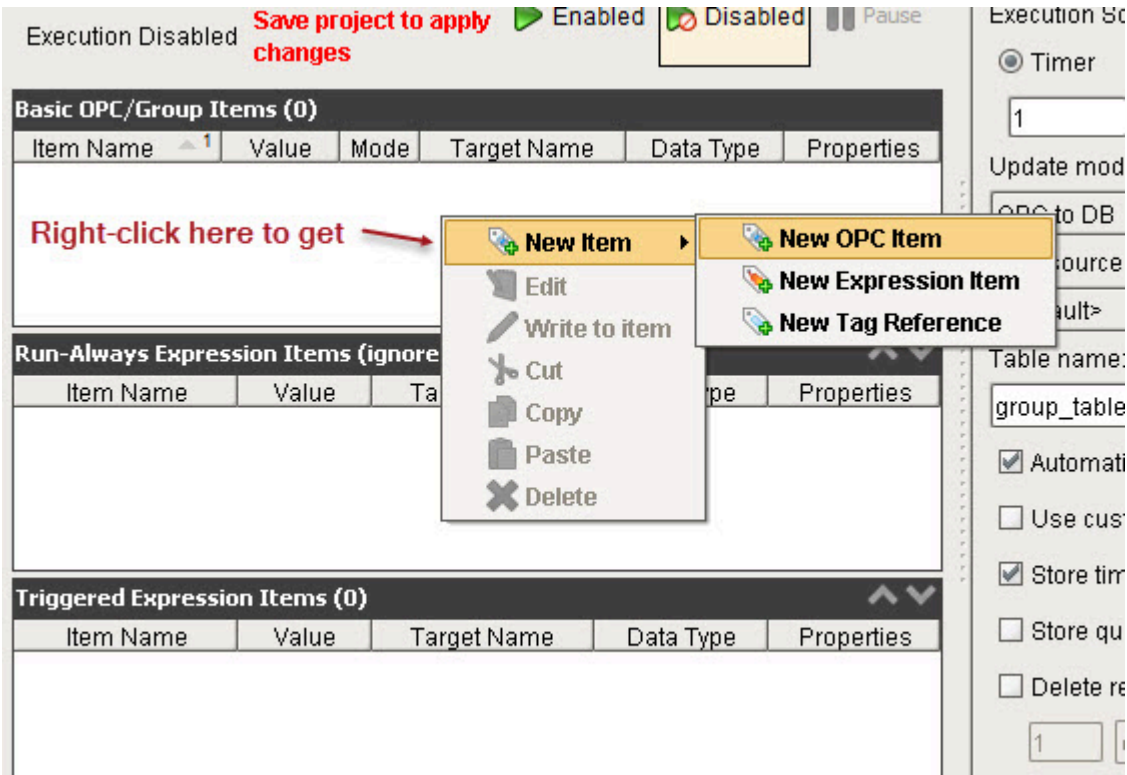
1. In the Designer, go to Project Browser, and click on **Transaction Groups**. The workspace now changes to the Transaction Group workspace.
2. Right-click on **Transaction Group** to create a New Transaction Group, or click on a group you have previously created. You will now see the workspace changes to look like:



3. Right-click in the white area, and choose **New Item > New OPC Item**.



The **Edit group tag** window is displayed as follows:



The following table describes all the properties on the **Edit group tag** window.

Property Name	Category	Description
Name	General	The name of the OPC item in the group. There cannot be duplicate names within a group.
Data Type	General	The datatype used to read values from the PLC.
OPC Server	OPC	The Selected OPC Server. This is a drop-down list showing all the OPC Servers added in the Ignition Gateway.
OPC Item Path	OPC	The OPC Item's address Including Device name or Channel if required.
Property	Value Mode	Which property of the OPC item you want to use. • Value - Item value • Quality - Quality code from OPC Server (192 = GOOD_DATA) • Timestamp - The last time the item value changed • Name - The SQLBridge Item Name property of this Item
Mode	Value Mode	Options for displaying values based on the Item value. • Direct Value - Item value • Hour Meter - Record the amount of time the Item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The datatype should be set to integer or float when using an Hour Meter regardless of the OPC Item type. ◦ On Zero - Use a zero value to accumulate time instead of a non-zero value ◦ Retentive - Retain the Hour Meter value when it is not accumulating. ◦ Units - The time units to display. • Event Meter - Record the number or times the Item value is non-zero. The datatype should be set to integer when using an Event Meter regardless of the OPC Item type. ◦ On Zero - Use a zero value to accumulate events instead of a non-zero value

Property Name	Category	Description
Mode	Write Target	Changes the items directional read/write option. • Use group's mode - Inherit the Update Mode from the Item's Group. • OPC to DB - Only read from the OPC server and write to the database. • DB to OPC - Only read from the database and write to the OPC Server. • Bi-directional OPC wins - Read and Write to both the database and OPC Server. On group start, write OPC Server values to the database. • Bi-directional DB wins - Read and Write to both the database and OPC Server. On group start, write database values to the OPC Server.
Target Type	Write Target	This is the selection for what the Item will write to when the group executes. • None, read-only item - Do not write this value to the database. • Database field - Write the Item value to the specified column in the database table. This list will populate with all the column names from the Group's target table after the first time the group is run.
Target Name	Write Target	The name of the column in the database that this Item will write to when the group executes. The Target Name list will populate with all the column names from the Group's target table if the Target Type is Database field.
Alarming	Alarming	For alarm settings on OPC items, please see the Alarming Properties page.

Tag Reference

Tag References are used just like OPC Items, adding the convenience of using a Tag that has already been set up with scaling and alarm data.

Property Name	Category	Description
Name	General	The name of the Tag Reference in the group. There cannot be duplicate names within a group.
Tag Path	General	The path to the tag being referenced. This value is not editable except by clicking the Insert Tag button. There cannot be duplicate names within a group.
Data Type	General	The datatype to write to into the database if this item is not read-only.
Property	Value Mode	Which property of the Tag you want to use. - Value - Item value - Quality - Quality code from OPC Server (192 = GOOD_DATA) - Timestamp - The last time the item value changed - Name - The SQLBridge Item Name property of this Item
Mode	Value Mode	Options for displaying values based on the Item value. - Direct Value - Item value - Hour Meter - Record the amount of time the Item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The datatype should be set to integer or float when using an Hour Meter regardless of the OPC Item type. - On Zero - Use a zero value to accumulate time instead of a non-zero value - Retentive - Retain the Hour Meter value when it is not accumulating. - Units - The time units to display. - Event Meter - Record the number or times the Item value is non-zero. The datatype should be set to integer when using an Event Meter regardless of the OPC Item type. - On Zero - Use a zero value to accumulate events instead of a non-zero value

Property Name	Category	Description
Mode	Write Target	Changes the items directional read/write option. This is only editable when the target Type is set to Database field. • Use group's mode - Inherit the Update Mode from the Item's Group. • OPC to DB - Only read from the OPC server and write to the database. • DB to OPC - Only read from the database and write to the OPC Server. • Bi-directional OPC wins - Read and Write to both the database and OPC Server. On group start, write OPC Server values to the database. • Bi-directional DB wins - Read and Write to both the database and OPC Server. On group start, write database values to the database.
Target Type	Write Target	This is the selection for what the Item will write to when the group executes. • None, read-only item - Do not write this value to the database. • Database field - Write the Item value to the specified column in the database table. This list will populate with all the column names from the Group's target table after the first time the group is run.
Target Name	Write Target	The name of the column in the database that this Item will write to when the group executes. The Target Name list will populate with all the column names from the Group's target table if the Target Type is Database field.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Expression Items



INDUCTIVE
UNIVERSITY

Expression Items

[Watch the video](#)

Expression Items are items not driven by a PLC. Instead they allow you to configure a static value, or use some other means to automatically set a value, such as a query. They are useful for executing comparisons, simple math, and looking up values from other database tables.

Much like OPC Items, Expression Items can have alarms configured, as well as numeric scaling applied directly to the item.

Scope

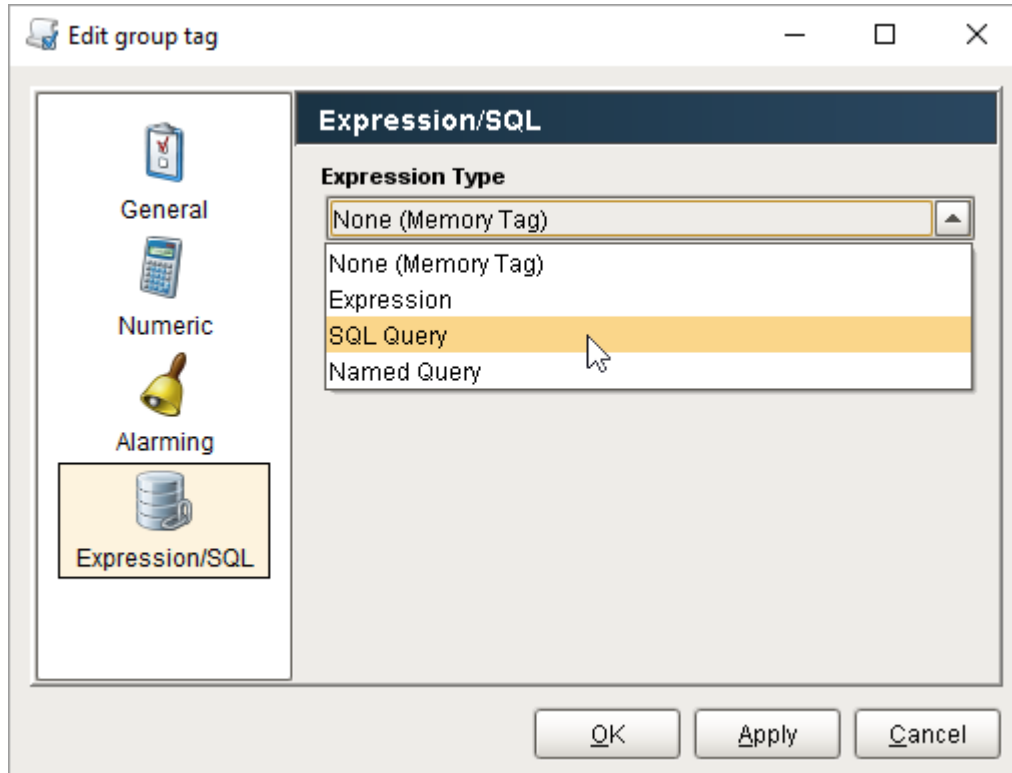
It is important to understand that an Expression Item only exists within its group, and can not be referenced by items in other transaction groups, Tags, and any components on a window. The only exception is to use an Expression Item to store/retrieve a value with `storeVariable()` and `getVariable()` functions. These functions store values in a space that is accessible by all Transaction Groups.

Execution Order

All Expression items will evaluate in order from top to bottom. This means referencing an Expression Item above will pull the new value, but referencing an Expression Item below will give you the value from the last group execution.

Expression Type

How an Expression Item determines its value depends heavily on its type.



- **None:** Behaves similar to a Memory Tag in that the value does not automatically change.
- **Expression:** Uses [Ignition's Expression Language](#) to determine the value on the Item. The expression can reference other items in the group, as well as Tags.
- **SQL Query:** Utilizes a SQL query to determine the item's value. Thus, a query can execute on the item and the results can be referenced by other items in the same group.
- **New in 7.9.4** **Named Query** : Named Queries can be called from an Expression item.

Run Always vs. Triggered Items



INDUCTIVE
UNIVERSITY

**Run Always vs. Triggered
Items**

[Watch the video](#)

Expression Item can be configured in two different evaluation states:

Triggered Expression Items executes only when the transaction group is triggered. However if the group is **not** configured to execute on a trigger, then the item will evaluate every time the group executes (similar to how the Run-Always state works). This is the default evaluation state new Expression Items use.

Run-Always Expression Items will run before the group trigger is checked, so it always executes at the group's rate. This allows your expression to always evaluate regardless of the trigger in the group. Additionally, this state allows you to use the Expression Item as the trigger for the group. We advise that you never have a Target for a Run-Always Expression item because it always runs.

Expression Items

Running

Enabled

Disabled

Pause

Basic OPC/Group Items (0)

Item Name	Value	Mode	Target Name	Data Type	Properties
-----------	-------	------	-------------	-----------	------------

Run-Always Expression Items (ignore trigger) (1)

Item Name	Value	Target Name	Data Type	Properties
 My Run-Always It...	100	 Read-only	Int4	

Triggered Expression Items (1)

Item Name	Value	Target Name	Data Type	Properties
 My Triggered Item	0	 Read-only	Int4	

Changing the Evaluation State

Toggling between the two modes can be accomplished by dragging and dropping the Expression Item to either the **Run-Always Expression Items** table or the **Triggered Expression Items** table. Alternatively, the evaluation state can be changed by editing the Expression Item and toggling the **Run-always (ignore trigger)** checkbox.

Edit group tag

General Properties

General Properties

Name
item

Value
0

Data Type
Integer

Value Mode

Property
Value

Mode

☒ Direct value

☐ Hour meter ☐ Event meter

☐ On Zero ☐ Retentive Units second(s)

☐ Reset on condition: > 0

Evaluation Options

☐ Run-always (ignore trigger)

Write target

Target Type
None, read-only item

Target Name

OK Apply Cancel

SQL Queries and Expressions



INDUCTIVE
UNIVERSITY

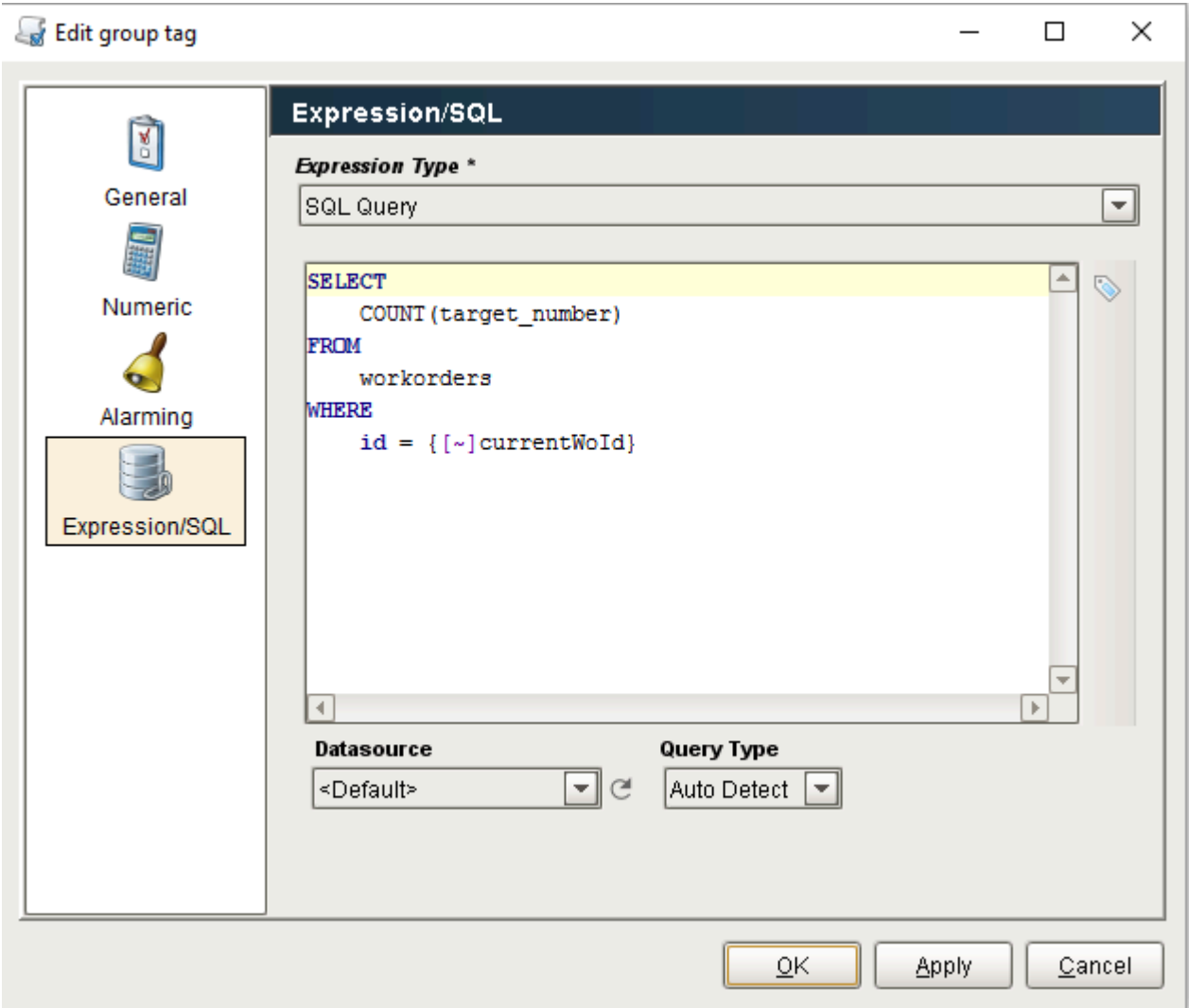
SQL Query Expression Items

[Watch the video](#)

Expression items can use **SQL statements** and **Ignition's Expression language** to automatically determine the value of an Expression Item. This is useful in scenarios where you want to use a value from the database as the trigger for the transaction group, or aggregate several other items in the group into a single value.

Expressions and queries on an Expression Item can reference the value of other items in the group or Tags in the system by clicking on the Tag icon .

There are several Expression functions available that exist only for Transaction groups. You can find them in the **Store and Forward** and **Variables** sections of the **f(x)** function list.



Expression Item Properties

The following properties are available when editing an Expression Item

Property Name	Description
Name	The name of the OPC item in the group. There cannot be duplicate names within a group.
Value	The static value of this Expression item. This will be overwritten by an Expression/SQL binding.
Datatype	The datatype values are stored as.
Property	Which property of the OPC item you want to use. - Value - Item value - Quality - Quality code of the expression/SQL Query (192 = GOOD_DATA) - Timestamp - The last time the item value changed. - Name - The SQLBridge Item Name property of this Item.
Mode	Options for displaying values based on the item value. - Direct Value - Item value - Hour Meter - Record the amount of time the Item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The datatype should be set to integer or float when using an Hour Meter regardless of the OPC Item type. - On Zero - Use a zero value to accumulate time instead of a non-zero value - Retentive - Retain the Hour Meter value when it is not accumulating. - Units - The time units to display. - Event Meter - Record the number or times the Item value is non-zero. The datatype should be set to integer when using an Event Meter regardless of the OPC Item type. - On Zero - Use a zero value to accumulate events instead of a non-zero value
Evaluation Mode	Run-always (ignore Trigger) - When selected, this causes the group to evaluate at each group interval, before the trigger state is evaluated.

Property Name	Description
Write Target	Target Type - This is the selection for what the Item will write to when the group executes. • None, read-only item - Do not write this value to the database. • Database field - Write the Item value to the specified column in the database table. • Other Tag - Write the Expression Item's value back to an OPC item or Tag Reference.

In addition to properties above, expression items share some common properties with tags.

- Numeric properties can be found on the [Tag Scaling Properties page](#).
- Alarming properties can be found under the Alarming Properties section of the [Tag Properties page](#).
- Expression/SQL Properties can be found under the Expression/SQL Properties section of the [Tag Properties page](#).

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

SQL Query Expression Items



INDUCTIVE
UNIVERSITY

SQL Query Expression Items

[Watch the video](#)

Expression items can be SQL statements that handle select, update, insert, or delete statements from a database. More complicated queries can require you to set the query type manually. If the query is simple, Ignition automatically detects the query type. SQL query expression items can serve as triggers for the transaction group or they can only execute when the transaction group is triggered.

To trigger a transaction group with a SQL query Expression item

1. In the Transaction Group workspace, create a [New Expression item](#).
2. On the **Edit group tag**, go to the **Expression/SQL** tab, and select **SQL Query** from the dropdown list.
3. Enter a SQL query into the Expression item, and click **OK**. The SQL query must return a numerical value.
4. Go to the **Trigger** tab, from the **Trigger on item** dropdown list, select the Expression item that has the SQL query.
5. Properly configure the trigger conditions and start the transaction group.

SQL STATEMENTS IN TRANSACTION GROUPS

It is important to be mindful of expression items that perform SQL statements. A transaction group that executes repeatedly can also be executing multiple expression items. If these expression items are executing a SQL statement, the query count on a database can increase dramatically.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Run Always vs. Triggered Items



INDUCTIVE
UNIVERSITY

Run Always vs. Triggered Items

[Watch the video](#)

In the Transaction Group workspace, you can set the Expression items in a group to execute in two different ways:

- **Run-Always** in which the Expression items will execute at the groups rate (based on the **Execution Scheduling** settings on the **Action** tab).
- **Triggered** in which the Expression items run when the group is triggered (based on the settings on the **Trigger** tab).

If on the **Trigger** tab, the option **Execute this group on a trigger** is not selected, the items in the **Run-Always Expression Item** area and **Triggered Expression Items** area will all execute at the same time. The advantage selecting the **Execute this group on a trigger** option is that it gives you flexibility on when the expression items are executed which may result in easing the load on a database.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Hour and Event Meters



INDUCTIVE
UNIVERSITY

Hour and Event Meters

[Watch the video](#)

Hour meter and **Event meter** refer to the **Value Mode** option settings on tags in Transaction Groups. The Value mode drives values that are used to create values that determine how long a value was true. While the selected Value Mode for most transactions is **Direct**, however, the **Hour meter** mode accumulates value for the duration of a condition, and the **Event meter** accumulates count in response to the condition.

Hour Meter

It is common to write to a tag during the time when a tag's value is true. An hour meter simplifies this effort. Hour meters can be meters that accumulate the millisecond, second, minute, hour, or day.

To count the duration of a tag being true

1. From OPC Browser or Tag Browser, drag a boolean tag into the **Basic OPC/Groups Items** area of a **Standard** transaction group.
2. From OPC Browser or Tag Browser, drag a memory tag or OPC tag (must be a numeric data type) into the **Basic OPC/Groups Items** portion of the **Standard** transaction group.
3. Right-click on the boolean tag in the transaction group and select **Edit** to edit it. The **Edit group tag** window is displayed.
4. In the **Edit group tag** window, for **Value Mode**, select **Hour meter**.
5. In the **Edit group tag** window, for **Target Type** select **Other tag** from the dropdown menu, and in **Target Name** enter the name of the memory tag as the target, and click **OK**.
6. In the **Basic OPC/Groups Items** area where the tags are located, go to the **Target Name** column, left-click on each tag to get the dropdown menu, and set the following:
 - i. For the boolean tag, select the memory tag from the dropdown which is set previously as the target tag to write the hour meter to.

- ii. For the memory tag, select Read-only from the dropdown.
7. Click **Enabled** at the top of the page, and do a **File > Save** to start the group.
8. Make the boolean tag true to start the Hour meter.

Event Meter

Another common scenario is to count the number of times an event occurred. For example, where there is boolean tag and you want to count the number of cycles the boolean tag has experienced.

To count in response to a tag being true

1. From OPC Browser or Tag Browser, drag a boolean tag into the **Basic OPC/Groups Items** area of a **Standard** transaction group.
2. From OPC Browser or Tag Browser, drag a memory tag or OPC tag (must be a numeric data type) into the **Basic OPC/Groups Items** portion of the **Standard** transaction group.
3. Right-click on the boolean tag in the transaction group and select Edit to edit it. The **Edit group tag** window is displayed.
4. In the **Edit group tag** window, for **Value Mode**, select **Event meter**.
5. In the **Edit group tag** window, for **Target Type** select **Other tag** from the dropdown menu, and in **Target Name** enter the name of the memory tag as the target, and click **OK**.
6. In the **Basic OPC/Groups Items** area where the tags are located, go to the **Target Name** column, left-click on each tag to get the dropdown menu, and set the following:
 - i. For the boolean tag, select the memory tag from the dropdown which is set previously as the target tag to write the hour meter to.
 - ii. For the memory tag, select Read-only from the dropdown.
7. Click **Enabled** at the top of the page, and do a **File > Save** to start the group.
8. Make the boolean tag true to start the Event meter.

Reset an Hour or Event Meter Based on a Condition



INDUCTIVE
UNIVERSITY

Resetting Hour and Event
Meters

[Watch the video](#)

You can set the hour or event meter based on a condition.

1. In the **Basic OPC/Group Items** section, right-click and **Edit** a tag that is serving as the Hour or Event meter. The **edit group tag** window is displayed.
2. In the **Value Mode** area, select the **Reset on condition** check box, click the **tag** icon to display the **Choose Tag** window, and select a **Group Tag** from the pop-up window.
3. Click **OK**.
4. Next to the **tag** icon, choose the **operator** sign, and enter a number.
5. Click **OK**. The target tag will now reset in response to the condition.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Trigger Options

Triggering a Group

It is often useful to execute a group only when a certain condition is met or as a bit turns on or off. Triggers allow Groups to run based on values changing in various ways.

Execute on Value Change



INDUCTIVE
UNIVERSITY

Trigger - On Value Change

[Watch the video](#)

A group can execute when the group's tags have changed, or when a particular tag within the group has changed. In either case, the transaction group will execute every time the value or values change.

1. In the **Trigger** tab, select at the very top the **Only evaluate when values have changed** checkbox. Now the group will execute if any of the tags change.
2. To execute when only one tag changes, from the **Tags to watch for change** dropdown, select **Custom**, click on **Select tags**, select the tag from the pop-up window, and click **OK**. You can select more than one tag at a time in order to monitor more than one tag for value changes.
3. From the **Trigger on item** dropdown, select the appropriate tag to execute the tag on a trigger.
4. Select the **Active on value change** radio button.

Execute While Condition is True



INDUCTIVE
UNIVERSITY

Trigger - While Condition is True

[Watch the video](#)

Groups can execute while a condition is true resulting in the transaction group continuing to execute for the duration of this condition.

1. Create a transaction group, and drag a numeric or boolean tag into the **Basic OPC/Group Items** section.
2. From the **Target Name** column dropdown, select **Read Only**.
3. Go to the **Trigger** tab, and select the **Execute this group on a trigger** checkbox.
4. In the **Trigger conditions** area, set the trigger conditions which will determine under what condition the group executes.

Execute on a Rising Edge



INDUCTIVE
UNIVERSITY

Trigger - Rising Edge (Once)

[Watch the video](#)

Groups can execute when the trigger becomes True. This is known as a **rising edge trigger** and it will only execute once and will not execute again until the trigger repeats the same cycle.

1. Create a standard transaction group with any number of tags as long as one of them is a boolean tag that will serve at the trigger for the group.
2. Set the **Write Target** for the boolean tag to **Read-only** by selecting read-only from its drop down in the Target Name column.
3. Go to the **Trigger** tab and select the check box to **Execute this group on a trigger**. Select the boolean tag from the drop down menu and select to have the group only execute once while the trigger is active.
4. Save the transaction group to start the group.

Reset Trigger



INDUCTIVE
UNIVERSITY

Trigger - Reset Trigger

[Watch the video](#)

Resetting a trigger after execution of a triggered transaction group will result in the transaction group writing once to the targets followed by writing back to the trigger to reset it.

To reset the trigger after execution

1. Create a transaction group with a boolean tag. The write target for this tag should be read only.
2. Select the **Trigger** tab and select the **Execute this group on a trigger** check box.
3. Select the **Reset trigger after execution** check box.
4. Press save to start the transaction group.

Handshakes



INDUCTIVE
UNIVERSITY

Trigger - Handshakes

[Watch the video](#)

When a group executes, it either completes successfully or an error prevents its execution. The outcome of an execution can be handled in the handshake section of the trigger section of the transaction group. When a group executes successfully or fails to execute, the handshake can write a value back to a tag to alert the user that the group executed successfully or unsuccessfully.

To set handshake values for alerting the user

1. Create a transaction group with a boolean and a numeric tag.
2. Set the boolean and the numeric tag to read only.
3. Go to the **Trigger** tab and choose to **Execute this group on a trigger**.
4. Select the boolean tag as the trigger In the **Trigger on item** drop down, and select the appropriate execution conditions.
5. In the bottom section, select **Write handshake on success**, select the numeric tag to write to, and choose a number that signifies success.
6. Likewise, in the bottom section, select **Write handshake on failure**, select the numeric tag to write to, and choose a number that signifies failure.