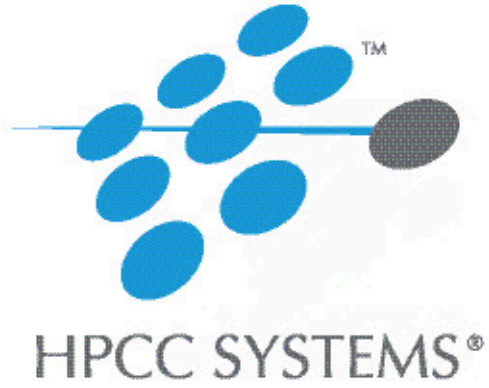
Visualizing ECL Results

Boca Raton Documentation Team



Visualizing ECL Results

Boca Raton Documentation Team

We welcome your comments and feedback about this document via email to <docfeedback@hpccsystems.com>

Please include **Documentation Feedback** in the subject line and reference the document name, page numbers, and current Version Number in the text of the message.

2022 Version 8.4.46-1

Visualization Introduction

The Visualization bundle is an open-source add-on to the HPCC Systems® platform to allow you to create visualizations from the results of queries written in ECL.

Visualizations are an important means of conveying information from massive data. A good visual representation can help a human produce actionable analysis. A visually comprehensive representation of the information can help make the obscure more obvious.

Pie Charts, Line graphs, Maps, and other visual graphs help us understand the answers found in data queries. Crunching big data is only part of a solution; we must be able to make sense of the data, too. Data visualizations simplify the complex.

The Visualizer bundle extends the HPCC Systems platform's functionality by allowing you to plot your data onto charts, graphs, and maps to add a visual representation that can be easily understood.

In addition, the underlying visualization framework supports advanced features to allow you to combine graphs to make interactive dashboards.

Installation

To install the Visualization bundle, use the ecl command line interface.

```
ecl bundle install https://github.com/hpcc-systems/Visualizer.git
```

To use the "ecl bundle install <git url>" command, you must have git installed and configured on your system. Git must be accessible to the user (in the path).

Using the Visualization library

Once installed, you merely IMPORT the library, then call any method that is appropriate for your data shape.

Creating a basic visualization typically requires the following steps:

1. Creation of a suitable dataset.

2. Output the dataset with a suitable name, so that visualization can locate the data.

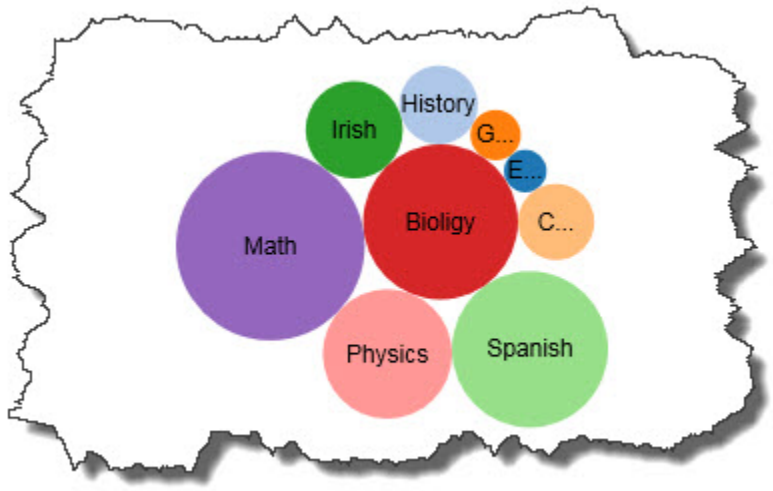
3. Create (and output) a the visualization, referencing the named output from step 2.

To change the appearance of the visualization, dermatology properties can be provided while creating the visualization or can be applied at runtime via the live dermatology property editor.

For Example:

```
IMPORT Visualizer;
ds := DATASET([ {'English', 5},
                {'History', 17},
                {'Geography', 7},
                {'Chemistry', 16},
                {'Irish', 26},
                {'Spanish', 67},
                {'Bioligy', 66},
                {'Physics', 46},
                {'Math', 98}],
                {STRING subject, INTEGER4 year});
data_example := OUTPUT(ds, NAMED('Chart2D__test'));
data_example;
viz_bubble := Visualizer.TwoD.Bubble('bubble',, 'Chart2D__test');
viz_bubble;
```

This code creates a Bubble chart as shown below:

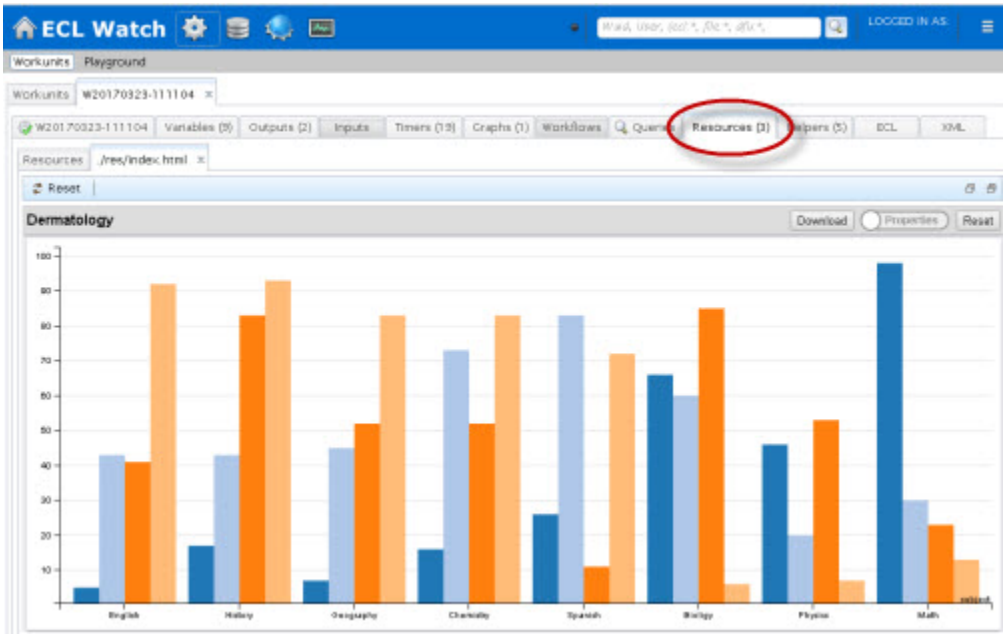


See Methods for details about the categories of visualization methods and all available methods.

Viewing the Visualization

After running a query with a visualization included, you can see the visualization in ECL Watch. Visualizations are appended to the workunit as additional *resources* and can be viewed from the **Resources** tab on the Workunit Details page.

Open the workunit, then select the **Resources** tab.



Dermatology Properties

The dermatology layer provides a means of setting properties for the manner in which a visualization appears. For example, you can change the palletteID to a new color scheme. After running a query with a visualization included, it can be viewed within ECL Watch.

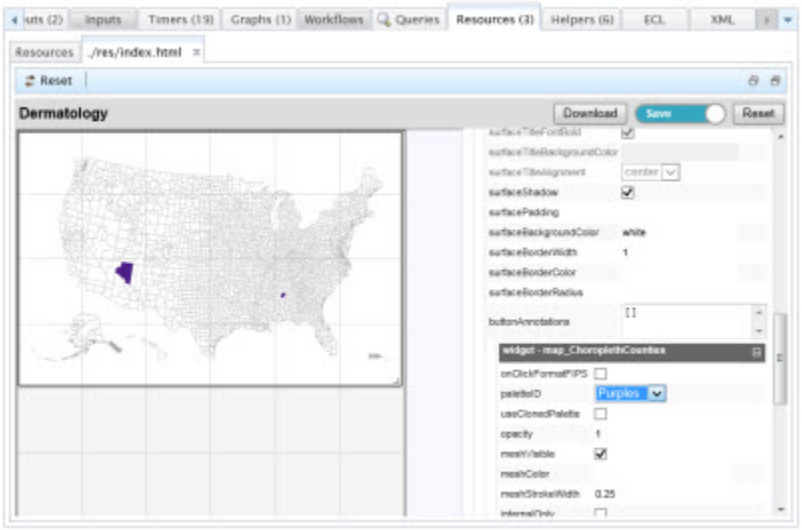
The **Dermatology Editor** lets you alter the appearance of the visualization after it has been executed. The same properties can also be specified in the ECL when creating the visualization.

You can set properties in your ECL code or afterwards in ECL Watch:

1. Open the workunit, then select the **Resources** tab.

2. Press the **Properties** button.

3. Edit Properties as desired.



4. Press the **Save** button.

Methods

Methods are categorized by the shape of the data they support. The categories are:

Two-Dimensional

Multi-Dimensional

Geospatial

General

Two-Dimensional Visualization Methods

This MODULE contains a selection of visualizations for two-dimensional data.

Bubble

Pie

Summary

WordCloud

Multi-Dimensional Visualization Methods

This MODULE contains a selection of visualizations for multi-dimensional data.

Area

Bar

Column

HexBin

Line

Scatter

Step

Geospatial Visualization Methods

USStates

USCounties

Euro

EuroIE

EuroGB

Note: The EuroIE and EuroGB methods are provided as examples. To create other country-specific methods, merely use the Euro method and provide the two-character country code in the _region parameter.

General Visualization Methods

Grid

Handson Grid

Two-Dimensional Methods

This section covers methods that create two-dimensional visualizations contained in the TwoD module. These methods provide ways to depict data in a two-dimensional space.

Bubble

Visualizer.TwoD.Bubble((*id* , **[***dataSource***], [***outputName***], [***filteredBy***], [***mappings***], [***properties***]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie query), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps human labels <--> field Ids
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **Bubble** visualization method creates a bubble chart from two-dimensional data. A bubble chart is a variation of a pie chart where data points are shown as bubbles and the size of the bubble is represented by the second dimension of the data.

Example:

```
IMPORT Visualizer;
ds := DATASET([ {'English', 5},
                {'History', 17},
                {'Geography', 7},
                {'Chemistry', 16},
                {'Irish', 26},
                {'Spanish', 67},
                {'Biology', 66},
                {'Physics', 46},
                {'Math', 98}],
                {STRING subject, INTEGER4 year});
data_example := OUTPUT(ds, NAMED('Chart2D__test'));
data_example;
viz_bubble := Visualizer.TwoD.Bubble('bubble',, 'Chart2D__test');
viz_bubble;
```

Pie

Visualizer.TwoD.Pie((*id* , **[***dataSource***], [***outputName***], [***filteredBy***], [***mappings***], [***properties***]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie query), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps human labels <--> field Ids
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **Pie** visualization method creates a pie chart from two-dimensional data. A pie chart is a graph in which a circle is divided into secions representing a portion of the whole.

Example:

```
IMPORT Visualizer;
ds := DATASET([ {'English', 5},
                {'History', 17},
                {'Geography', 7},
                {'Chemistry', 16},
                {'Irish', 26},
                {'Spanish', 67},
                {'Bioligy', 66},
                {'Physics', 46},
                {'Math', 98}],
                {STRING subject, INTEGER4 year});
data_example := OUTPUT(ds, NAMED('Chart2D__test'));
data_example;
viz_pie := Visualizer.TwoD.Pie('pie',, 'Chart2D__test');
viz_pie;
```

Summary

Visualizer.TwoD.Summary((*id* , **[***dataSource***], [***outputName***], [***filteredBy***], [***mappings***], [***properties***]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie query), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps human labels <--> field Ids
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **Summary** visualization method creates a summary chart from two-dimensional data. The summary graph defaults to graphic showing scrolling data values.

Example:

```
IMPORT Visualizer;
ds := DATASET([ {'English', 5},
                {'History', 17},
                {'Geography', 7},
                {'Chemistry', 16},
                {'Irish', 26},
                {'Spanish', 67},
                {'Bioligy', 66},
                {'Physics', 46},
                {'Math', 98}],
                {STRING subject, INTEGER4 year});
data_example := OUTPUT(ds, NAMED('Chart2D__test'));
data_example;
viz_Summary := Visualizer.TwoD.Summary('Summary',, 'Chart2D__test');
viz_Summary;
```

WordCloud

Visualizer.TwoD.WordCloud((*id* , **[***dataSource***], [***outputName***], [***filteredBy***], [***mappings***], [***properties***]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie query), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps human labels <--> field Ids
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **WordCloud** visualization method creates a word cloud chart from two-dimensional data. A WordCloud is weighted word list in visual design. It is a visual representation for text data, typically used to depict the weight or importance of words by the font size or color.

Example:

```
IMPORT Visualizer;
ds := DATASET([ {'English', 5},
                {'History', 17},
                {'Geography', 7},
                {'Chemistry', 16},
                {'Irish', 26},
                {'Spanish', 67},
                {'Biology', 66},
                {'Physics', 46},
                {'Math', 98}],
                {STRING subject, INTEGER4 year});
data_example := OUTPUT(ds, NAMED('Chart2D__test'));
data_example;
viz_WordCloud := Visualizer.TwoD.WordCloud('WordCloud',, 'Chart2D__test');
viz_WordCloud;
```

Multi-Dimensional Methods

This section covers methods that create multi-dimensional visualizations contained in the MultiD module. These methods provide ways to depict data in a multi-dimensional space.

Area

Visualizer.MuiltD.Area((*id* **,** **[***.dataSource***],** **[***outputName***],** **[***mappings***],** **[***properties***]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **Area** visualization method displays graphically quantitative data. The area between the axis and each line is shown and can be emphasized using shading or colors. This is commonly used to compare two or more quantities.

Example:

```
IMPORT Visualizer;
  ds := DATASET([ {'English', 5, 43, 41, 92},
                  {'History', 17, 43, 83, 93},
                  {'Geography', 7, 45, 52, 83},
                  {'Chemistry', 16, 73, 52, 83},
                  {'Spanish', 26, 83, 11, 72},
                  {'Bioligy', 66, 60, 85, 6},
                  {'Physics', 46, 20, 53, 7},
                  {'Math', 98, 30, 23, 13}],
                  {STRING subject, INTEGER4 year1,
                   INTEGER4 year2, INTEGER4 year3, INTEGER4 year4});
  data_exams := OUTPUT(ds, NAMED('MultiD__test'));
  data_exams;

  viz_area := Visualizer.MultiD.Area('area',, 'MultiD__test');
  Viz_area;
```

Bar

Visualizer.MuiltD.Bar((*id* , **[**.*dataSource*]**, [***outputName*]**, [***mappings*]**, [***properties*]**]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **Bar** visualization method displays a graph consisting of horizontal rectangles (bars) where each length is proportional to data value.

Example:

```
IMPORT Visualizer;
  ds := DATASET([ {'English', 5, 43, 41, 92},
                  {'History', 17, 43, 83, 93},
                  {'Geography', 7, 45, 52, 83},
                  {'Chemistry', 16, 73, 52, 83},
                  {'Spanish', 26, 83, 11, 72},
                  {'Bioligy', 66, 60, 85, 6},
                  {'Physics', 46, 20, 53, 7},
                  {'Math', 98, 30, 23, 13}],
                  {STRING subject, INTEGER4 year1,
                   INTEGER4 year2, INTEGER4 year3, INTEGER4 year4});
  data_exams := OUTPUT(ds, NAMED('MultiD__test'));
  data_exams;

  viz_bar := Visualizer.MultiD.Bar('bar',, 'MultiD__test');
  viz_bar;
```

Column

Visualizer.MuiltD.Column((*id* , **[***.dataSource***], [***outputName***], [***mappings***], [***properties***]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **Column** visualization method displays a graph consisting of vertical rectangles (bars) where each length is proportional to data value.

Example:

```
IMPORT Visualizer;
  ds := DATASET([ {'English', 5, 43, 41, 92},
                  {'History', 17, 43, 83, 93},
                  {'Geography', 7, 45, 52, 83},
                  {'Chemistry', 16, 73, 52, 83},
                  {'Spanish', 26, 83, 11, 72},
                  {'Bioligy', 66, 60, 85, 6},
                  {'Physics', 46, 20, 53, 7},
                  {'Math', 98, 30, 23, 13}],
                  {STRING subject, INTEGER4 year1,
                   INTEGER4 year2, INTEGER4 year3, INTEGER4 year4});
  data_exams := OUTPUT(ds, NAMED('MultiD__test'));
  data_exams;

  viz_Column := Visualizer.MultiD.Column('column',, 'MultiD__test');
  viz_Column;
```

HexBin

Visualizer.MuiltD.HexBin((*id* **,** [*.dataSource*]**,** [*outputName*]**,** [*mappings*]**,** [*properties*]**);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **HexBin** visualization method displays a Hexagonal Bin plot which plots two or more continuous variables against each other. Hexagonal binning is useful for aggregating data values into a coarser display. For example, rather than showing thousands of points on a scatter plot, you can combine points into a few hexagons to show the distribution.

Example:

```
IMPORT Visualizer;
  ds := DATASET([ {'English', 5, 43, 41, 92},
                  {'History', 17, 43, 83, 93},
                  {'Geography', 7, 45, 52, 83},
                  {'Chemistry', 16, 73, 52, 83},
                  {'Spanish', 26, 83, 11, 72},
                  {'Bioligy', 66, 60, 85, 6},
                  {'Physics', 46, 20, 53, 7},
                  {'Math', 98, 30, 23, 13}],
                  {STRING subject, INTEGER4 year1,
                   INTEGER4 year2, INTEGER4 year3, INTEGER4 year4});
  data_exams := OUTPUT(ds, NAMED('MultiD__test'));
  data_exams;

  viz_hexBin := Visualizer.MultiD.HexBin('hexBin',, 'MultiD__test');
  viz_hexBin;
```

Line

Visualizer.MuiltD.Line((*id* , **[***.dataSource***], [***outputName***], [***mappings***], [***properties***]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **Line** visualization method displays a line graph which uses points connected by lines to show how values change.

Example:

```
IMPORT Visualizer;
  ds := DATASET([ {'English', 5, 43, 41, 92},
                  {'History', 17, 43, 83, 93},
                  {'Geography', 7, 45, 52, 83},
                  {'Chemistry', 16, 73, 52, 83},
                  {'Spanish', 26, 83, 11, 72},
                  {'Bioligy', 66, 60, 85, 6},
                  {'Physics', 46, 20, 53, 7},
                  {'Math', 98, 30, 23, 13}],
                  {STRING subject, INTEGER4 year1,
                   INTEGER4 year2, INTEGER4 year3, INTEGER4 year4});
  data_exams := OUTPUT(ds, NAMED('MultiD__test'));
  data_exams;

  viz_line := Visualizer.MultiD.Line('line',, 'MultiD__test');
  viz_line;
```

Scatter

Visualizer.MuiltD.Scatter((*id* , **[***.dataSource***], [***outputName***], [***mappings***], [***properties***]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **Scatter** visualization method displays a scatter plot using horizontal and vertical axes to plot data points.

Example:

```
IMPORT Visualizer;
  ds := DATASET([ {'English', 5, 43, 41, 92},
                  {'History', 17, 43, 83, 93},
                  {'Geography', 7, 45, 52, 83},
                  {'Chemistry', 16, 73, 52, 83},
                  {'Spanish', 26, 83, 11, 72},
                  {'Bioligy', 66, 60, 85, 6},
                  {'Physics', 46, 20, 53, 7},
                  {'Math', 98, 30, 23, 13}],
                  {STRING subject, INTEGER4 year1,
                   INTEGER4 year2, INTEGER4 year3, INTEGER4 year4});
  data_exams := OUTPUT(ds, NAMED('MultiD__test'));
  data_exams;
  viz_scatter := Visualizer.MultiD.Scatter('scatter',, 'MultiD__test');
  viz_scatter;
```

Step

Visualizer.MuiltD.Step((*id* , **[***.dataSource***], [***outputName***], [***mappings***], [***properties***]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **Step** visualization method creates a step graph made of lines in horizontal intervals or 'steps'.

Example:

```
IMPORT Visualizer;
  ds := DATASET([ {'English', 5, 43, 41, 92},
                  {'History', 17, 43, 83, 93},
                  {'Geography', 7, 45, 52, 83},
                  {'Chemistry', 16, 73, 52, 83},
                  {'Spanish', 26, 83, 11, 72},
                  {'Bioligy', 66, 60, 85, 6},
                  {'Physics', 46, 20, 53, 7},
                  {'Math', 98, 30, 23, 13}],
                  {STRING subject, INTEGER4 year1,
                   INTEGER4 year2, INTEGER4 year3, INTEGER4 year4});
  data_exams := OUTPUT(ds, NAMED('MultiD__test'));
  data_exams;
  viz_step := Visualizer.MultiD.Step('step',, 'MultiD__test');
  viz_step;
```

Geospatial Methods

This section covers the Geospatial methods that create geographical map visualizations contained in the Choropleth module. These methods project data onto maps using shading to depict values.

USStates

Visualizer.Choropleth.USStates((*id* , **[***.dataSource***], [***outputName***], [***mappings***], [***properties***]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **USStates** visualization method depicts data on a map of U.S. States.

Example:

```
IMPORT Visualizer;
  _usStates := DATASET([  {'AL', 4779736},
                          {'AK', 710231},
                          {'AZ', 6392017},
                          {'AR', 2915918}],
                          {STRING State, INTEGER4 weight});
  data_usStates := OUTPUT(_usStates, NAMED('choro_usStates'));
  data_usStates;
  viz_usstates := Visualizer.Choropleth.USStates('usStates',, 'choro_usStates');
  viz_usstates;
```

USCounties

Visualizer.Choropleth.USCounties((*id* , [*.dataSource*], [*outputName*], [*mappings*], [*properties*]);

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **USCounties** visualization method depicts data on a map of U.S. counties.

Example:

```
IMPORT Visualizer;

  _usCounties := DATASET([    {1073,29.946185501741},
                              {1097,0.79566003616637},
                              {1117,1.5223596574691},
                              {4005,27.311773623042}],
                              {STRING FIPS, INTEGER4 weight});

  data_usCounties := OUTPUT(_usCounties, NAMED('choro_usCounties'));
  data_usCounties;
  viz_uscounties := Visualizer.Choropleth.USCounties('usCounties',, 'choro_usCounties');
  viz_uscounties;
```

Euro

Visualizer.Choropleth.Euro((*id* ,[*region*] [.*dataSource*], [*outputName*], [*mappings*], [*properties*]);

id	Unique identifier for the visualization
region	2 Letter European Code (GB, IE, etc.)
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **Euro** visualization method depicts data on a map of Europe. The **EuroIE** and **EuroGB** methods are provided as examples. To create other country-specific methods, merely use the Euro method and provide the two-character country code in the _region parameter.

Example:

```
IMPORT Visualizer;
_euroIE := DATASET([    {'Carlow', '27431', '27181', '54612'},
                                  {'Dublin City', '257303', '270309', '527612'},
                                  {'Kilkenny', '47788', '47631', '95419'},
                                  {'Cork', '198658', '201144', '399802'}],
                                  {STRING region, INTEGER4 males, INTEGER4 females,
                                   INTEGER4 total});
  data_euroIE := OUTPUT(_euroIE, NAMED('choro_euroIE'));
  data_euroIE;

  viz_euroIE := Visualizer.Choropleth.EuroIE('euroIE',, 'choro_euroIE',
                DATASET([{'County', 'region'}, {'Population', 'total'}], Visualizer.KeyValueDef),,
                DATASET([{'paletteID', 'Greens'}], Visualizer.KeyValueDef));
  viz_euroIE;
```

General Visualization Methods

This section covers methods that create generic visualizations contained in the Any module. These visualizations are handy for testing and examining your data source.

Grid

Visualizer.Any.Grid((*id* , **[**.*dataSource*], **[***outputName*], **[***mappings*], **[***properties*]**);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **Grid** visualization method creates a simple table or grid from any data.

Example:

```
IMPORT Visualizer;
ds := DATASET([ {'English', 5},
                {'History', 17},
                {'Geography', 7},
                {'Chemistry', 16},
                {'Irish', 26},
                {'Spanish', 67},
                {'Bioligy', 66},
                {'Physics', 46},
                {'Math', 98}],
                {STRING subject, INTEGER4 year});
data_example := OUTPUT(ds, NAMED('Chart2D__test'));
data_example;
viz_Grid := Visualizer.Any.Grid('Grid',, 'Chart2D__test');
viz_Grid;
```

HandsonGrid

Visualizer.Any.HandsonGrid((*id* , **[**.*dataSource*]**, [***outputName***], [***mappings***], [***properties***]);**

id	Unique identifier for the visualization
dataSource	Location of the result set (WUID, Logical File, Roxie Query result), defaults to the current WU
outputName	Result set name (ignored for Logical Files)
mappings	Maps Column Name <--> field ID
filteredBy	Filter condition (also useful for widget interactions)
properties	Dermatology properties. See Dermatology Properties
Return:	A "meta" output describing the visualization.

The **HandsonGrid** visualization method creates a hands on table or grid from any data.

Example:

```
IMPORT Visualizer;
ds := DATASET([ {'English', 5},
                {'History', 17},
                {'Geography', 7},
                {'Chemistry', 16},
                {'Irish', 26},
                {'Spanish', 67},
                {'Bioligy', 66},
                {'Physics', 46},
                {'Math', 98}],
                {STRING subject, INTEGER4 year});
data_example := OUTPUT(ds, NAMED('Chart2D__test'));
data_example;
viz_HandsonGrid := Visualizer.Any.HandsonGrid('HandsonGrid',, 'Chart2D__test');
viz_HandsonGrid;
```