Advanced Topics

This section contains advanced Prime Analytics topics, including

Customizing the Chart Data Set Interpretation

Dashboards include many visual elements. Of these, charts are of the most important because they present your data visually and allow your data to be interpreted quickly and easily. Charts have two Boolean parameters that might be difficult to understand at first: Crosstab mode and Series in rows. Both parameters are set to false by default.

If Crosstab mode is set to true, the chart expects a matrix dataset where series and categories create the matrix rows and columns and the data populates the matrix. The first data column is used for either series or category labels. If Series in rows is set to true, a series is a matrix row and the categories are set in columns. The following tables show how the data is interpreted using the Crosstab mode and Series in rows parameters. In the tables:

- S = series
- C = categories (X axis label for a vertical graph)
- ... = direction the table can be extended
- Shaded cells = column headers

In this example:

- Crosstab mode = true
- Series in rows = true

	C1	C2	...
S1	D	D	...
S2	D	D	...
S3	D	D	...
...	...	...	...

In this example:

- Crosstab mode = true
- Series in rows = false

	S1	S2	...
C1	D	D	...
C2	D	D	...
C3	D	D	...
...	...	...	...

Notice that the column headers will show in your chart when Crosstab mode is true. If Crosstab mode is false, the chart expects a three column dataset where each row corresponds to exactly one data point. In this case, the Series in rows parameter determines the column order. If Series in rows is false, the expected order is Series, Category, Data. If Series in rows is true, the expected order is Category, Series, Data. The following tables show how the data is interpreted when Crosstab mode is false.

Note In the CCC charts, use text strings or floating points for category labels. If you use integers for category labels, the charts might become corrupted.

Note In general, MDX queries and SQL queries with a GROUP BY clause are used to generate data sets in cross tab mode. Other SQL queries are more likely to generate data sets that aren't in cross tab mode.

Note Internally the CCC library uses the values 0,1, … n to refer to the categories along the X axis. If you also use these numbers as a category label, the data point will be drawn in the wrong category. If you need numerical values along the category axis, use floating-point formatted numbers.

If crosstab mode is false, the column headers (names) are not part of the dataset and therefore are not visible in the chart. In this example:

- Crosstab mode = false
- Series in rows = false

S	C	D
S	C	D
S	C	D
...	...	...

In this example:

- Crosstab mode = false
- Series in rows = true

C	S	D
C	S	D
C	S	D
...	...	...

Customizing CCC Charts Using Extension Points

The Pentaho Community Chart Components provide an extensive list of options under Advanced Properties. These options include extension points that allow you to set the font family, font size, and text orientation for different chart text components of the chart. When you select an extension point, you can create a list of name value pairs. The name has the format A_B where A is the item you want to customize and B is the property you want to customizes. For example, if you want to customize the font of title A = titleLabel and the property B = font, the parameter to set is titleLabel_font.

Common item types and the properties you can set as an indented list include:

- Text items, for example, titleLabel, xAxisLabel, yAxisLabel, legend, barLabel, pieLabel:
  - font—For example, 13 px serif.
  - fillStyle—For example, yellow.
  - textAngle—An angle in radians (you can to turn the horizontal axis labels at an angle to prevent them from overlapping with one another).

    Note If you change the textAngle of the xAxisLabel, you probably also need to set the textAlignment to Left to prevent the text from crossing the xAxis.

  - textAlign—For example, left, center, right.
  - textBaseline—For example, top, bottom.
  - textStyle—For example, black.
  - text—A function used to modify or format the text, for example, function(d) {return d.substr(5)} to limit the length of the text to the first five characters.
- dot (the marker used in a dot chart):
  - fillStyle—For example, white.
  - shape—For example, square.
  - shapeRadius—For example, 4
- line-items (line, xAxis, yAxis, dot, xAxisRule, yAxisRule):
  - lineWidth—For example, 0.5.
  - fillStyle—For example, green or #00ff00.
  - strokeStyle—For example, blue.
- bar (the marker used in a bar chart)
- pie (for the pie-charts):
  - innerRadius—For example, 10.
  - strokeStyle—For example, white.

Adding Interactivity with JavaScript functions

You can insert JavaScript functions in many chart locations. If you want to insert a function use the syntax:

```
function() { <body> return result; }
```

Do not put a = in front of the function and do not put a ';' after the closing bracket, otherwise the function will be interpreted as a text string instead of a function. If you want to insert a complex function in the dashboard, put the function in a separate resource file (.js). The JavaScript that you enter will be included in your dashboard file without thorough validation. If the JavaScript has a scoping error, for example, too many or too few closing brackets, your dashboard file might become corrupted and lead to unpredictable results or errors. If you put the JavaScript in a separate .js file, the file will be discarded if it contains a scoping error, and other components of your dashboard will not be affected.

Detailed descriptions of most of these components and their parameters can be found in the Pentaho solution repository. In the user console, go to the folder: BI Developer Examples/CDF/Documentation, or from the File menu, choose Open and navigate to the BI Developer Examples directory.

Note If the BI developer Examples does not show in your repository, edit the index.xml file in this folder and set visibility to true (or ask your administrator to do so).

Visibility of Updates in Continuous Queries

A continuous query (CQ) typically runs for a long and indefinite amount of time. An important question that arises for such long-running CQs is how and when they see updates to tables. For example, consider a CQ that involves a join between stream S and table T. Suppose that table T is updated from time to time on no particular schedule. The question is when each update is visible to the CQ; essentially how often should the system take a fresh snapshot of the committed table set. In theory, two extremes are possible:

- Never—Each CQ does not see any updates to participating tables during its lifetime.
- Immediately—Each CQ sees all updates instantly.

The first option is not desirable because a CQ can live forever and the updates can represent important information. However, the second option has two problems. First, the CQ can involve a join with a subquery over the table being updated. Recomputing every subquery for every incoming stream tuple impacts performance. Recomputation (refreshing a cache) is not a problem for most common situations where the CQ simply performs an index lookup. Updating the snapshot visibility is easy to do. However, a second and more important problem with immediate snapshots is that they can cause non-deterministic and non-repeatable results where a given chunk of data in a window can see multiple versions of the same record.

The solution is to make updates to tables visible in a streaming query only when windows advance and are not seen between window advances. For most queries, the application developer does not need to do anything to enable the update visibility. For some queries, however, where a complex subquery exists that caches computations over a table, the cache is only recomputed when the special built-in cq_cache_inval() function is invoked. It is up to the application developer to ensure this, and the typical pattern is to deploy on a table that can be modified, a trigger that calls the cq_cache_inval() function. On update of the table, the trigger is fired, and that results in the system recomputing the cache on the next window boundary.