Tips & Tricks for Writing and Debugging Chart Features

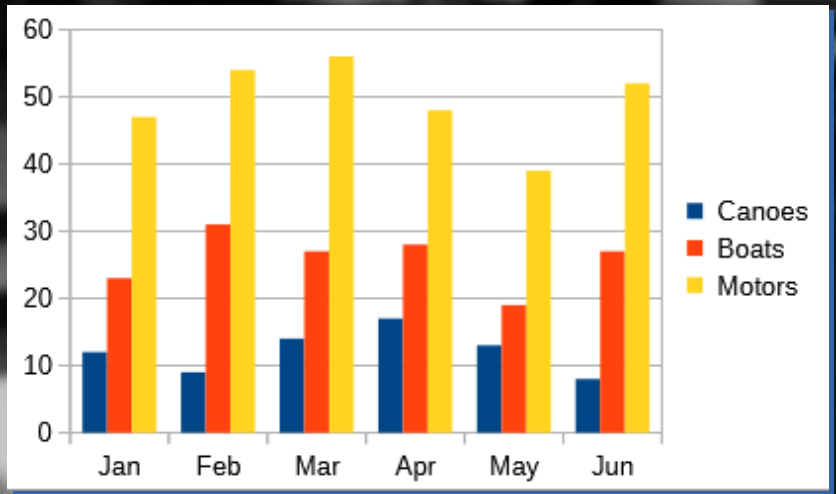
Balázs Varga

balazs.varga.extern@allotropia.de

Overview of Chart Wizard

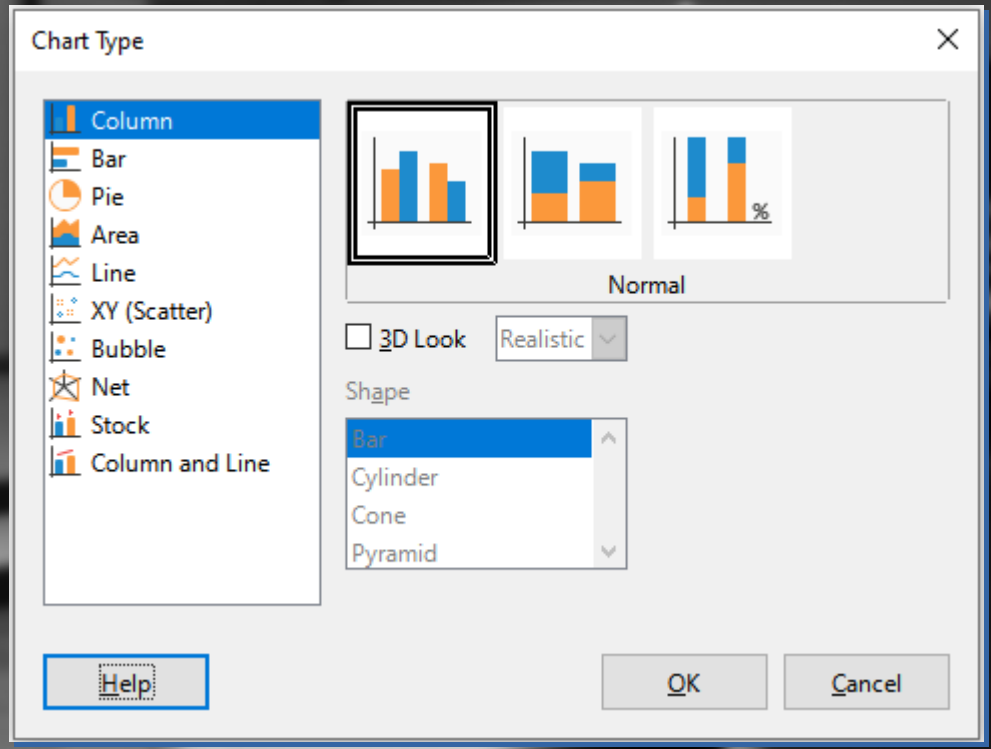
- Select the cells containing all of the data—including names, categories, and labels—to be included in the chart. The selection can be a single block, individual cells, or groups of cells (columns or rows). In this example, it may be best to select the cell range A2:D8, which will intentionally omit the overall title “Equipment Rentals” from the chart.
- When the data is in one place, the Chart Wizard can guess the range and create an initial chart even if all of the data is not selected. Before opening the Chart Wizard, just place the cursor or select a cell anywhere in the area of the data.
- Go to Insert > Chart on the Menu bar then click Finish to save the selections and close the Chart Wizard

	A	B	C	D
1	**Equipment Rentals**			
2		Canoes	Boats	Motors
3	Jan	12	23	47
4	Feb	9	31	54
5	Mar	14	27	56
6	Apr	17	28	48
7	May	13	19	39
8	Jun	8	27	52



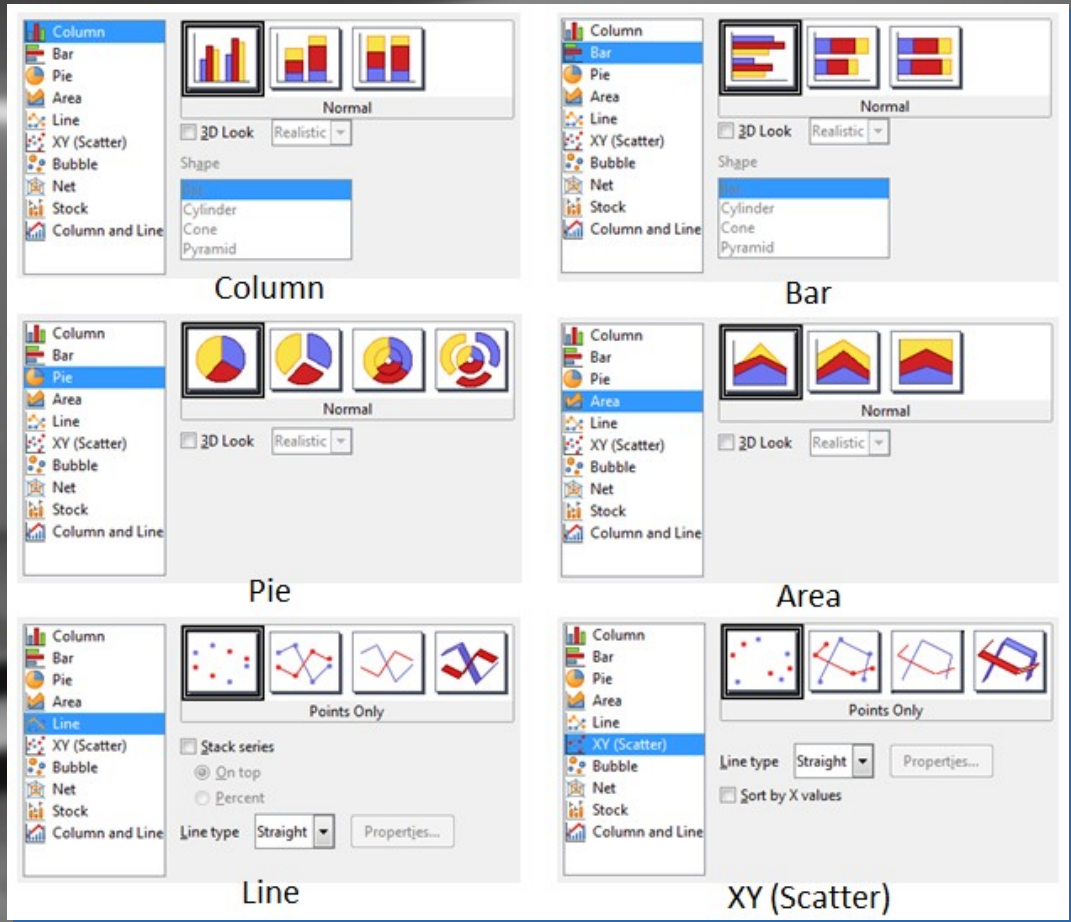
The available chart types

- Calc offers a choice of ten basic chart types
- The 2D variants are:
  - Normal
  - Stacked
  - Percent stacked
- 3D Look
  - Realistic – tries to give the best 3D look.
  - Simple – tries to mimic the chart view of other products.
- Shape
  - Gives options for the shape of the columns in 3D charts. The choices are: Bar, Cylinder, Cone, and Pyramid.



The available chart types

- Calc offers a choice of ten basic chart types
- The 2D variants are:
  - Normal
  - Stacked
  - Percent stacked
- 3D Look
  - Realistic – tries to give the best 3D look.
  - Simple – tries to mimic the chart view of other products.
- Shape
  - Gives options for the shape of the columns in 3D charts. The choices are: Bar, Cylinder, Cone, and Pyramid.



The chart elements for 2D and 3D charts

- The chart wall contains the graphic displaying the data.
- The chart area is the background of the entire chart.
- The chart title and subtitle, chart legend, axes labels, and axes names are in the chart area.
- The chart floor is only available for 3D charts.

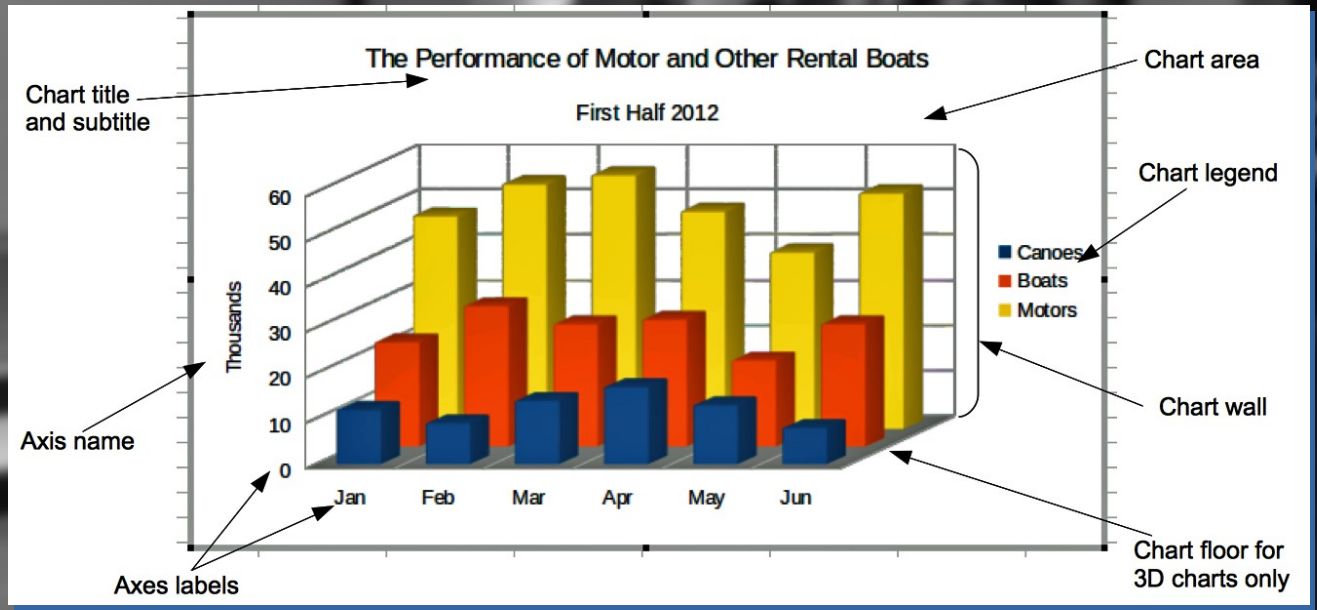
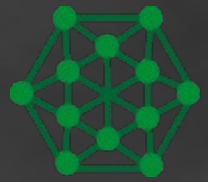


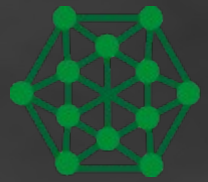
Chart2 API Overview



allotropia

- The chart2 module is quite complex. It simplifies four kinds of operation:
  - The creation of a new chart in a spreadsheet document, based on a template name.
  - The accessing and modification of elements inside a chart, such as the title, legend, axes, and colors.
  - The addition of extra data to a chart, such as error bars or a second graph.
  - The embedding of a chart in a document other than a spreadsheet, namely in a text document or slide presentation.

Chart Creation

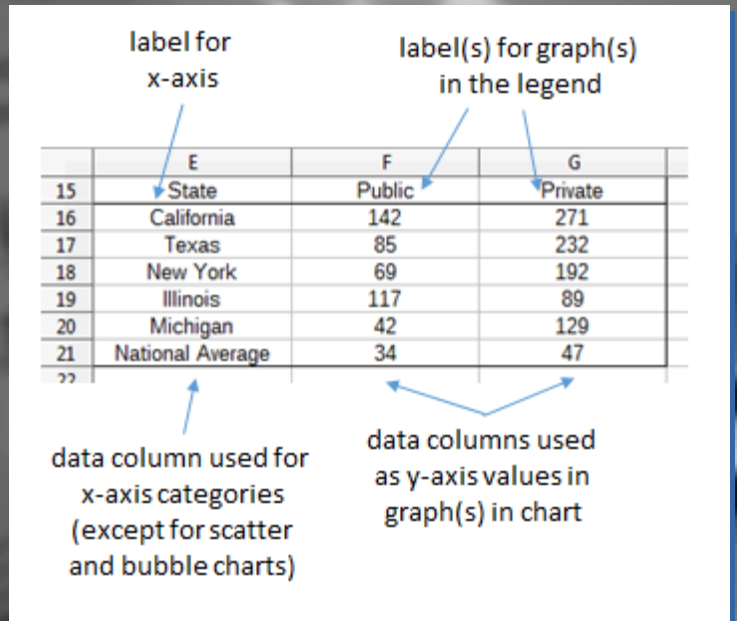


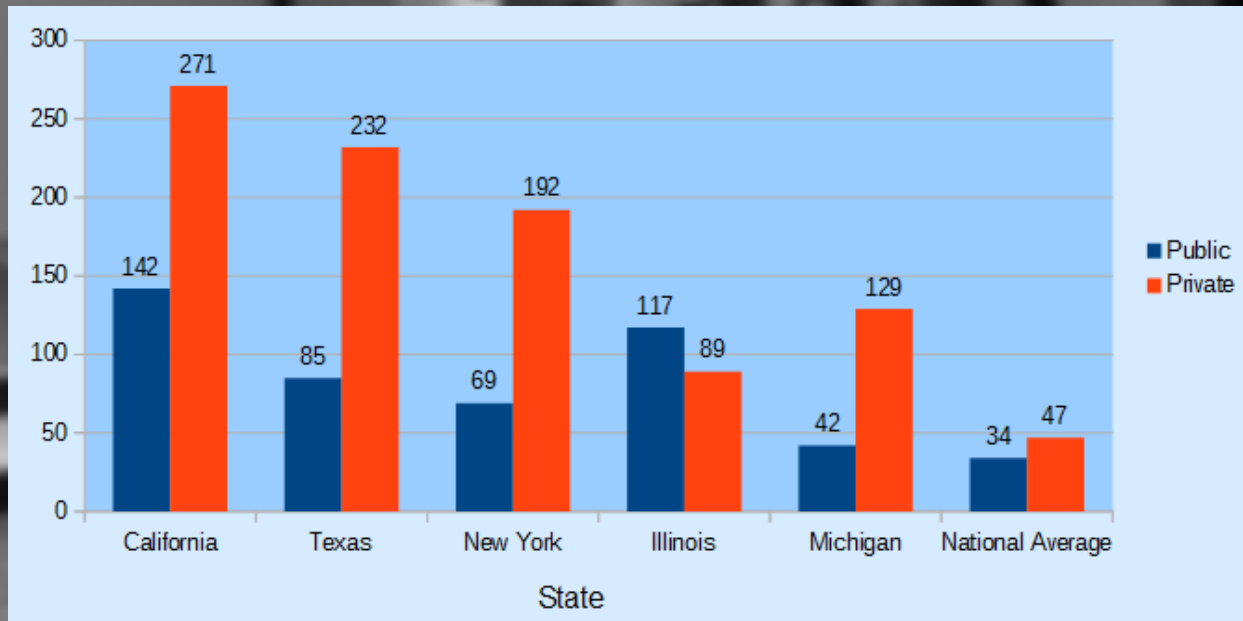
allotropia

- Chart creation can be divided into three steps:
  - A TableChart service is created inside the spreadsheet.
  - The ChartDocument service is accessed inside the TableChart.
  - The ChartDocument is initialized by linking together a chart template, diagram, and data source.

Creating a Table Chart

- The data is organized into columns, the first for the x-axis categories, and the others for the y-axis data displayed as graphs. The first row of the data range contains labels for the x-axis and the graphs.





Creating a Table Chart

- XTableCharts.addNewByName() adds a new TableChart to the TableCharts collection in a spreadsheet. This is shown graphically below, and is implemented by Chart2.addTableChart().

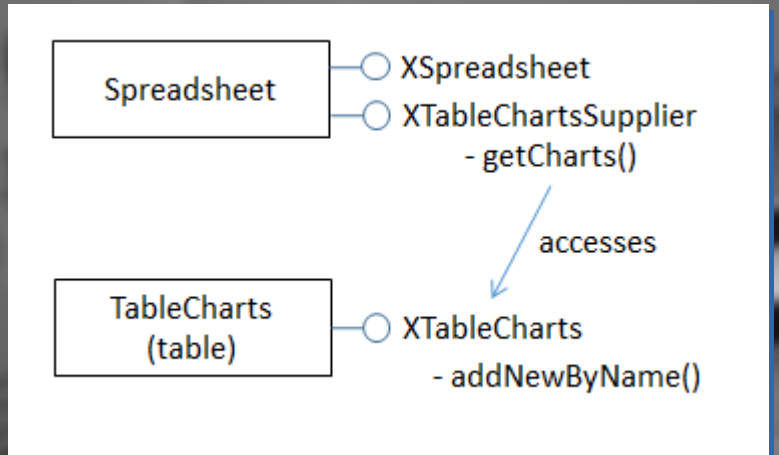


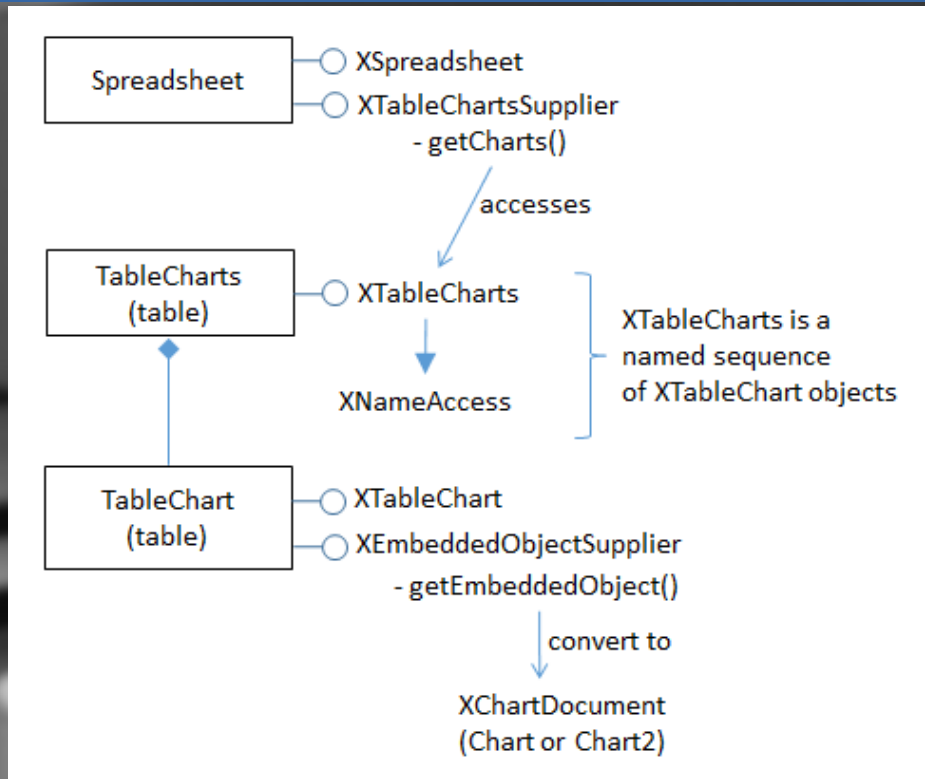
Chart2.addTableChart() is defined as:

```
// in the Chart2 class
public static void addTableChart(XSpreadsheet sheet,
                String chartName, CellRangeAddress cellsRange,
                String cellName, int width, int height)
// create table chart at cell name and size width x height
{
  XTableChartsSupplier chartsSupplier =
                    Lo.qi(XTableChartsSupplier.class, sheet);
  XTableCharts tableCharts = chartsSupplier.getCharts();

  com.sun.star.awt.Point pos = Calc.getCellPos(sheet, cellName);
  Rectangle rect = new Rectangle(pos.X, pos.Y,
                                 width*1000, height*1000);

  CellRangeAddress[] addrs = new CellRangeAddress[]{ cellsRange };
  tableCharts.addNewByName(chartName, rect, addrs, true, true);
}  // end of addTableChart()
```

Accessing the Chart Document

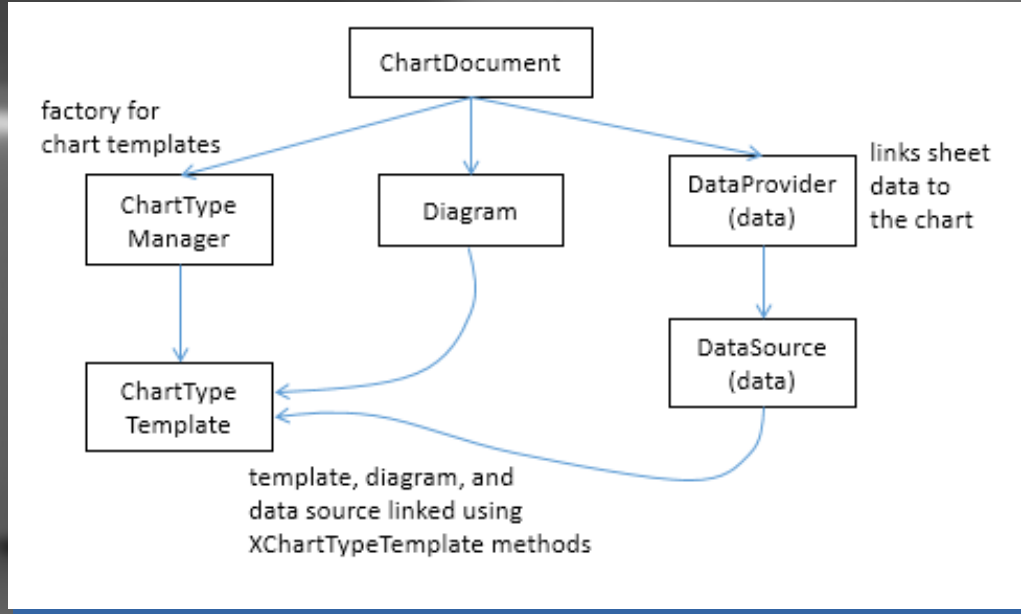
```
// in the Chart2 class
public static XChartDocument getChartDoc(XSpreadsheet sheet,
                                         String chartName)
// return the chart doc from the sheet
{ // get the named table chart
  XTableChart tableChart = getTableChart(sheet, chartName);
  if (tableChart == null)
    return null;

  // chart doc is embedded inside table chart
  XEmbeddedObjectSupplier eos =
              Lo.qi(XEmbeddedObjectSupplier.class, tableChart);
  return Lo.qi(XChartDocument.class, eos.getEmbeddedObject());
}  // end of getChartDoc()



public static XTableChart getTableChart(XSpreadsheet sheet,
                                        String chartName)
// return the named table chart from the sheet
{ // get the supplier for the table charts
  XTableChartsSupplier chartsSupplier =
                      Lo.qi(XTableChartsSupplier.class, sheet);
  XTableCharts tableCharts = chartsSupplier.getCharts();
  XNameAccess tcAccess = Lo.qi(XNameAccess.class, tableCharts);

  // try to access the chart with the specified name
  XTableChart tableChart = null;
  try {
    tableChart = Lo.qi(XTableChart.class,
                       tcAccess.getByName(chartName));
  }
  catch(Exception ex)
  {  System.out.println("Could not access " + chartName); }
  return tableChart;
}  // end of getTableChart()
```

```
Chart2.addTableChart(sheet, chartName, cellsRange, cellName,
                                                   width, height);
XChartDocument chartDoc = Chart2.getChartDoc(sheet, chartName);
```



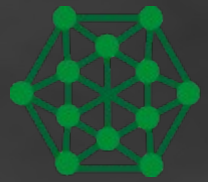
Initializing the Chart Document

- The chart document is initialized by linking three components: the chart template, the chart's diagram, and a data source.
- The first line converts "E15:G21" into a data range, which is passed to Chart2.insertChart(). The "A22" string and the 20x11 mm dimensions specify the position and size of the chart, and the last argument ("Column") is the desired chart template.




```
CellRangeAddress rangeAddr = Calc.getAddress(sheet, "E15:G21");
XChartDocument chartDoc =
      Chart2.insertChart(sheet, rangeAddr, "A22", 20,11, "Column");
```

Initializing the Chart Document



allotropia

```
// in the Chart2 class
// globals
private static final String CHART_NAME = "chart$$_";


public static XChartDocument insertChart(XSpreadsheet sheet,
                    CellRangeAddress cellsRange, String cellName,
                    int width, int height, String diagramName)
{
  String chartName = CHART_NAME + (int)(Math.random()*10000);
                     // generate a random name

  addTableChart(sheet, chartName, cellsRange, cellName,
                                          width, height);
  // get newly created (empty) chart
  XChartDocument chartDoc = getChartDoc(sheet, chartName);

  // assign chart template to the chart's diagram
  System.out.println("Using chart template: " + diagramName);
  XDiagram diagram = chartDoc.getFirstDiagram();
  XChartTypeTemplate ctTemplate =
              setTemplate(chartDoc, diagram, diagramName);
  if (ctTemplate == null)
    return null;

  boolean hasCats = hasCategories(diagramName);

  // initialize data source
  XDataProvider dp = chartDoc.getDataProvider();
  PropertyValue[] aProps = Props.makeProps(
        new String[] { "CellRangeRepresentation", "DataRowSource",
                       "FirstCellAsLabel" , "HasCategories" },
        new Object[] { Calc.getRangeStr(cellsRange, sheet),
                       ChartDataRowSource.COLUMNS, true, hasCats });
  XDataSource ds = dp.createDataSource(aProps);

  // add data source to chart template
  PropertyValue[] args = Props.makeProps("HasCategories", hasCats);
  ctTemplate.changeDiagramData(diagram, ds, args);

  // apply style settings to chart doc
  setBackgroundColors(chartDoc, Calc.PALE_BLUE, Calc.LIGHT_BLUE);
                              // background and wall colors

  if (hasCats)  // charts using x-axis categories
    setDataPointLabels(chartDoc, Chart2.DP_NUMBER);
                         // show y-axis values

  printChartTypes(chartDoc);
  return chartDoc;
}  // end of insertChart()
```

- insertChart() creates a new chart document by calling addTableChart() and getChartDoc(), and then proceeds to link the chart template, diagram, and data source.
- The chart diagram is the easiest to obtain, since it's directly accessible via the XChartDocument reference:

```
// part of Chart2.insertChart()...
XDiagram diagram = chartDoc.getFirstDiagram();
```

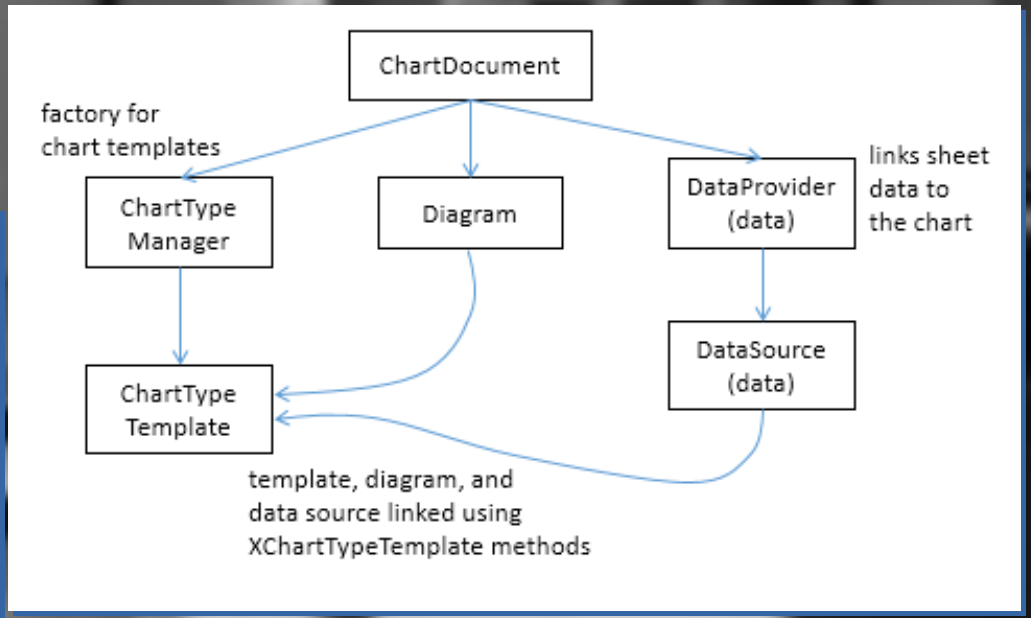
Get the Data Source

- Back in Chart2.insertChart(), the right-most branch involves the creation of an XDataProvider instance:

```
// part of Chart2.insertChart()...
XDataProvider dp = chartDoc.getDataProvider();
```

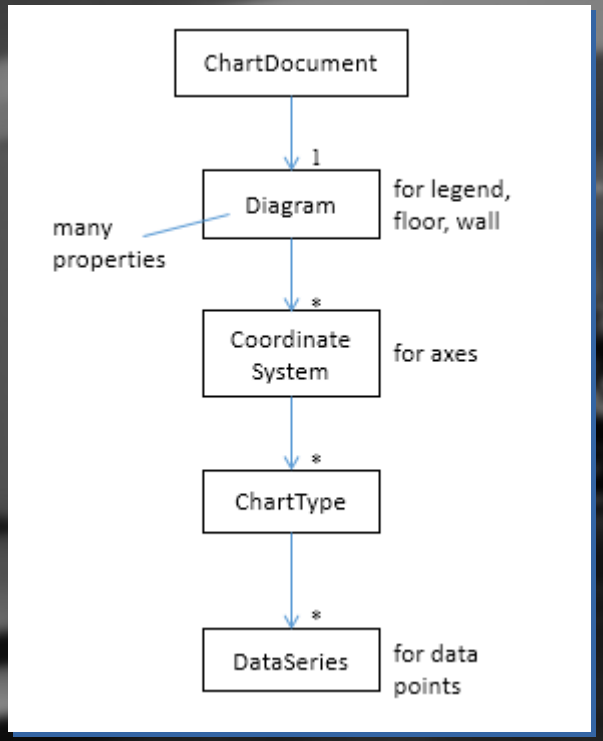
- This data provider converts the chart's data range into an XDataSource:

```
// part of Chart2.insertChart()...

boolean hasCats = hasCategories(diagramName);

PropertyValue[] aProps = Props.makeProps(
      new String[] { "CellRangeRepresentation", "DataRowSource",
                     "FirstCellAsLabel" , "HasCategories" },
      new Object[] { Calc.getRangeStr(cellsRange, sheet),
                     ChartDataRowSource.COLUMNS, true, hasCats });

XDataSource ds = dp.createDataSource(aProps);
```



Accessing and Modifying Chart Elements

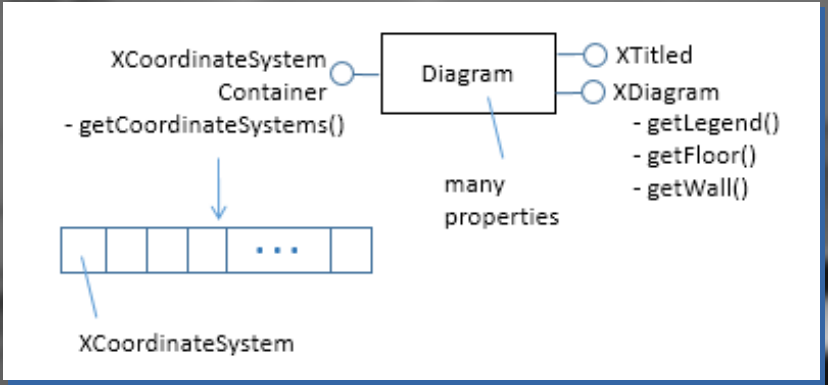
- Almost every aspect of a chart can be adjusted, including such things as its color scheme, the fonts, the scaling of the axes, the positioning of the legend, axis labels, and titles. It's also possible to augment charts with regression line details, error bars, and additional graphs. These elements are located in a number of different places in the hierarchy of services accessible through the ChartDocument service. A simplified version of this hierarchy is shown.



Accessing the Diagram

- A chart's Diagram service is easily reached by calling ChartDocument.getFirstDiagram(), which returns a reference to the diagram's XDiagram interface:

```
XDiagram diagram = chartDoc.getFirstDiagram();
```



- Chart2.setBackgroundColors() changes the background and wall colors of the chart through the ChartDocument and Diagram services:

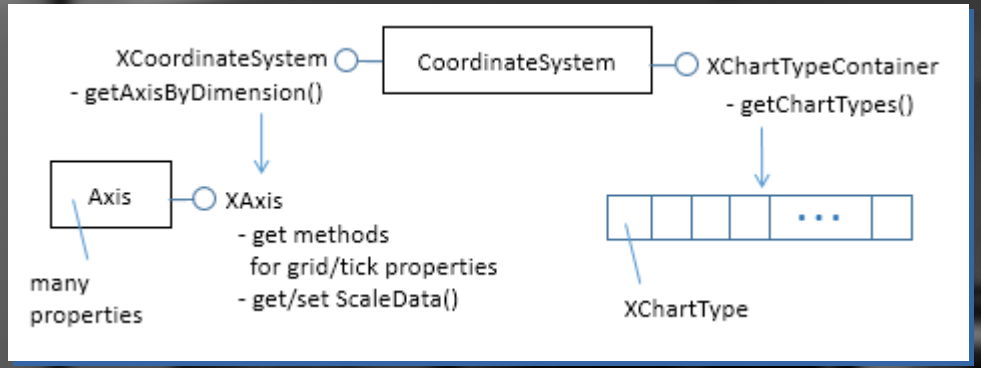
```
// in the Chart2 class
public static void setBackgroundColors(XChartDocument chartDoc,
                                       int bgColor, int wallColor)
{ if (bgColor > 0) {
    XPropertySet bgProps = chartDoc.getPageBackground();
    // Props.showProps("Background", bgProps);
    Props.setProperty(bgProps, "FillBackground", true);
    Props.setProperty(bgProps, "FillStyle", FillStyle.SOLID);
    Props.setProperty(bgProps, "FillColor", bgColor);
  }

  if (wallColor > 0) {
    XDiagram diagram = chartDoc.getFirstDiagram();
    XPropertySet wallProps = diagram.getWall();
    // Props.showProps("Wall", wallProps);
    Props.setProperty(wallProps, "FillBackground", true);
    Props.setProperty(wallProps, "FillStyle", FillStyle.SOLID);
    Props.setProperty(wallProps, "FillColor", wallColor);
  }
}  // end of setBackgroundColors()
```

Accessing the Coordinate System

- The diagram's coordinate systems are reached through XCoordinateSystemContainer.getCoordinateSystems(). Chart2.getCoordSystem() assumes that the programmer only wants the first coordinate system:

```
// in the Chart2 class
public static XCoordinateSystem getCoordSystem(
                                XChartDocument chartDoc)
{ XDiagram diagram = chartDoc.getFirstDiagram();
  XCoordinateSystemContainer coordSysCon =
            Lo.qi(XCoordinateSystemContainer.class, diagram);
  XCoordinateSystem[] coordSys =
                      coordSysCon.getCoordinateSystems();
  if (coordSys.length > 1)
    System.out.println("No of coord systems: " + coordSys.length +
                                    "; using first");
  return coordSys[0];  // return first
}  // end of getCoordSystem()
```

- The CoordinateSystem service is employed to access the chart's axes and its chart type (or types).

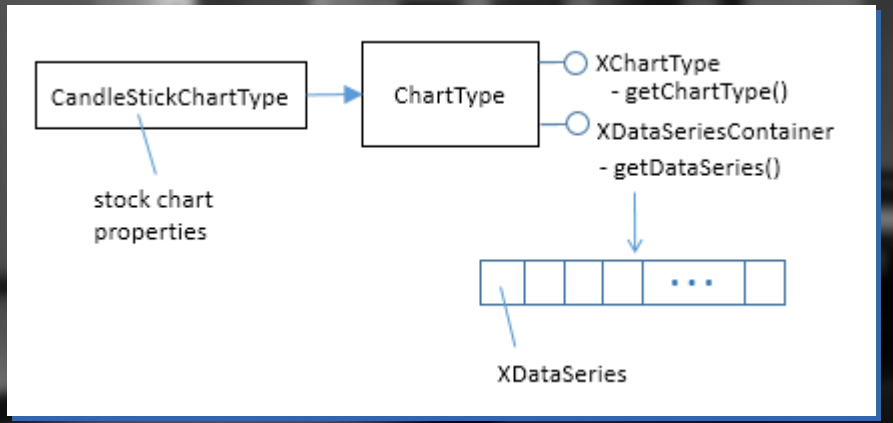


Accessing the Chart Type

- Chart types in a coordinate system are reached through XChartTypeContainer.getChartTypes(). Chart2.getChartType() assumes the programmer only wants the first chart type in the array:

```
// in the Chart2 class
public static XChartType getChartType(XChartDocument chartDoc)
{
  XChartType[] chartTypes = getChartTypes(chartDoc);
  return chartTypes[0];  // get first
}


public static XChartType[] getChartTypes(XChartDocument chartDoc)
{
  XCoordinateSystem coordSys = getCoordSystem(chartDoc);
  XChartTypeContainer ctCon =
      Lo.qi(XChartTypeContainer.class, coordSys);
  return ctCon.getChartTypes();
}  // end of getChartTypes()
```

- Somewhat surprisingly, the ChartType service isn't the home for chart type related properties; instead XChartType contains methods for examining chart type "roles". One useful features of XChartType is getChartType() which returns the type as a string.
- The CandleStickChartType service inherits ChartType, and contains properties related to stock charts.



Accessing the Data Series

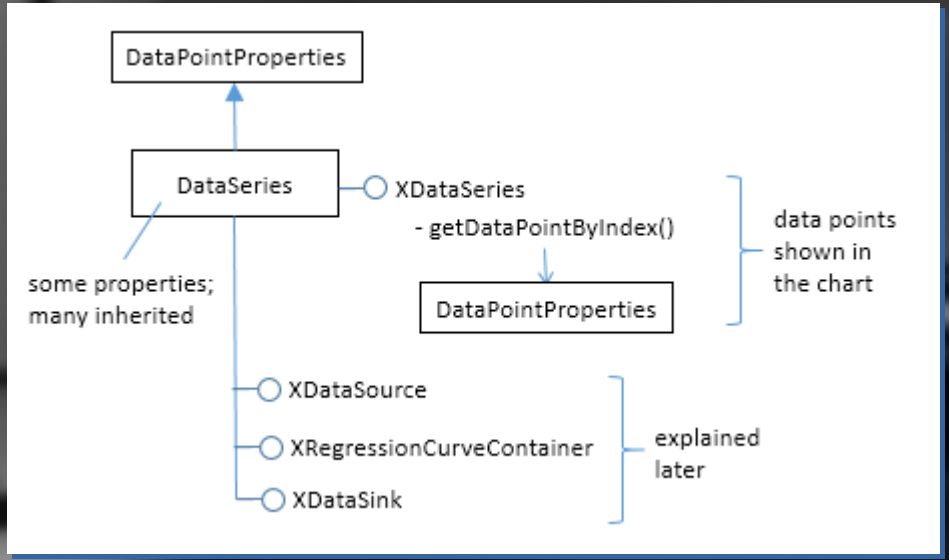
- Data series for a chart type is accessed via XDataSeriesContainer.getDataSeries(). This is implemented by Chart2.getDataSeries():

```
// in the Chart2 class
public static XDataSeries[] getDataSeries(XChartDocument chartDoc)
{
  XChartType xChartType = getChartType(chartDoc);
  XDataSeriesContainer dsCon =
                  Lo.qi(XDataSeriesContainer.class, xChartType);
  return dsCon.getDataSeries();
}  //end of getDataSeries()
```

- The second of the two chart changing methods called at the end of Chart2.insertChart(): Chart2.setDataPointLabels(), which switches on the displaying of the y-axis data points as numbers. The call is:

```
// part of Chart2.insertChart()...
setDataPointLabels(chartDoc, Chart2.DP_NUMBER);
```

- The DataSeries service is one of the more complex parts of the Chart2 module because of its support for several important interfaces.

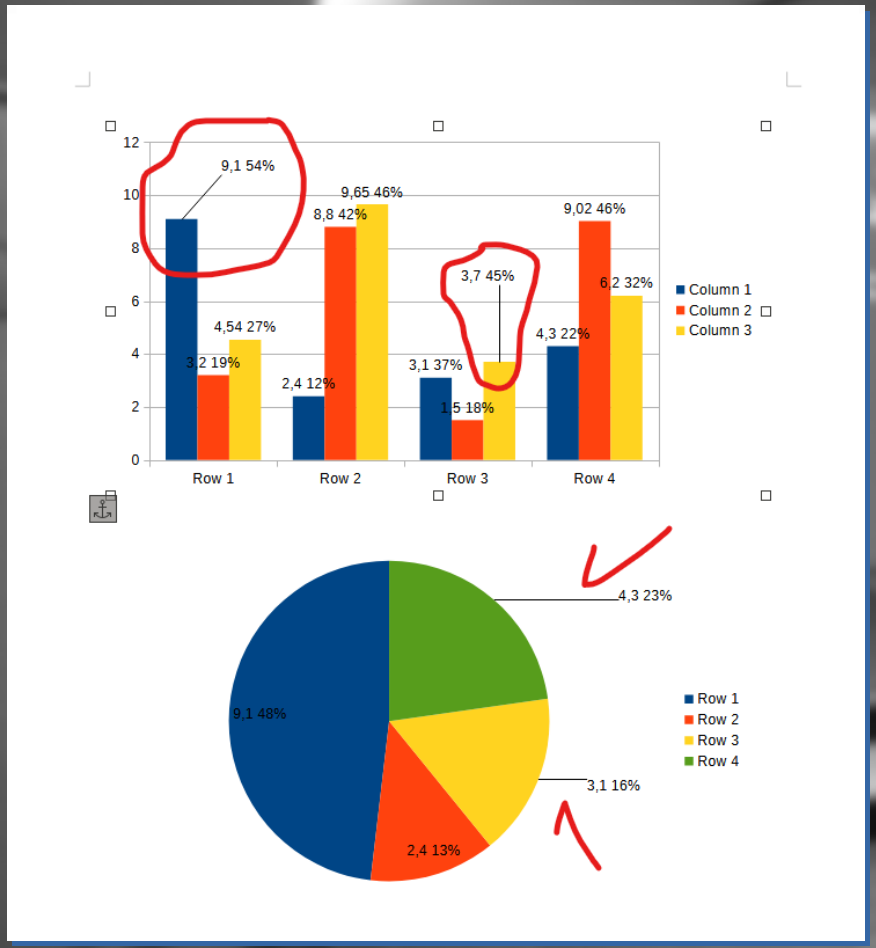


Accessing the Data Series


```
// in the Chart2 class
// data point label types
public static final int DP_NUMBER = 0;
public static final int DP_PERCENT = 1;
public static final int DP_CATEGORY = 2;
public static final int DP_SYMBOL = 3;
public static final int DP_NONE = 4;


public static void setDataPointLabels(XChartDocument chartDoc,
                                       int labelType)
// change label type for all data series
{
  XDataSeries[] dataSeriesArr = getDataSeries(chartDoc);
  for (XDataSeries dataSeries : dataSeriesArr) {
    // visit every data series
    DataPointLabel dpLabel =
         (DataPointLabel) Props.getProperty(dataSeries, "Label");
    dpLabel.ShowNumber = false;        // reset show types
    dpLabel.ShowCategoryName = false;
    dpLabel.ShowLegendSymbol = false;

    if (labelType == DP_NUMBER)
      dpLabel.ShowNumber = true;
    else if (labelType == DP_PERCENT) {
      dpLabel.ShowNumber = true;
      dpLabel.ShowNumberInPercent = true;
    }
    else if (labelType == DP_CATEGORY)
      dpLabel.ShowCategoryName = true;
    else if (labelType == DP_SYMBOL)
      dpLabel.ShowLegendSymbol = true;
    else if (labelType == DP_NONE) {}  // do nothing
    else
      System.out.println("Unrecognized label type");
    Props.setProperty(dataSeries, "Label", dataPointLabel);
  }
}  // end of setDataPointLabels()
```

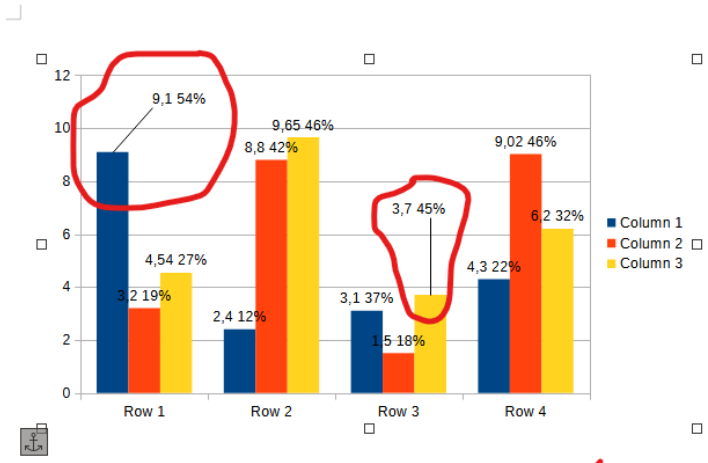
- Chart2.setDataPointLabels() uses Chart2.getDataSeries() described on the previous slide, which returns an array of all the data series used in the chart. setDataPointLabels() iterates through the array and manipulates the "Label" property for each series. In other words, it modifies each data series property without accessing each point.
- The "Label" DataSeries property is inherited from DataPointProperties. "Label" is of type DataPointLabel which maintains four 'show' booleans for displaying the number and other kinds of information next to the data point. Depending on the labelType value passed to Chart2.setDataPointLabels(), one or more of these booleans are set and the "Label" property updated.

Implement Custom Label positions with leader lines



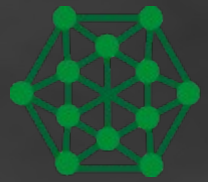
- “It would be better for the data labels to be freely placed, like text boxes with position and size x and y coordinates, instead. Really all elements should be freely customizable like text boxes, in terms of placement, size, and content. It may interfere and cause problems otherwise. The individual elements can still be moved as a group when moving a chart.”

Implement Custom Label positions with leader lines



- 86859: tdf#48436 Chart: add CustomLabelPosition UNO API property | Link
- 87759: tdf#130032 Chart OOXML Import: fix data label custom position | Link
- 88371: tdf#130590 Chart OOXML export: fix custom label position | Link
- 88531: tdf#90749 chart: add leader lines to custom data label positions | Link
- 89446: tdf#108110 ODF chart: import/export of custom position of data point labels | Link
- 98323: tdf#134563 Add UNO API for custom leader lines | Link
- 101442: tdf#134571 chart2, xmloff: add loext:custom-leader-lines | Link

More information



allotropia

- https://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star_1_1chart2.html
- https://wiki.documentfoundation.org/Documentation/DevGuide/Charts
- https://books.libreoffice.org/en/CG71/CG7103-ChartsAndGraphs.html#bkmRefHeading385931127550272


allotropia
Questions & Answers!
OSB Open Source Business ALLIANCE
OASIS MEMBER
LibreOffice The Document Foundation