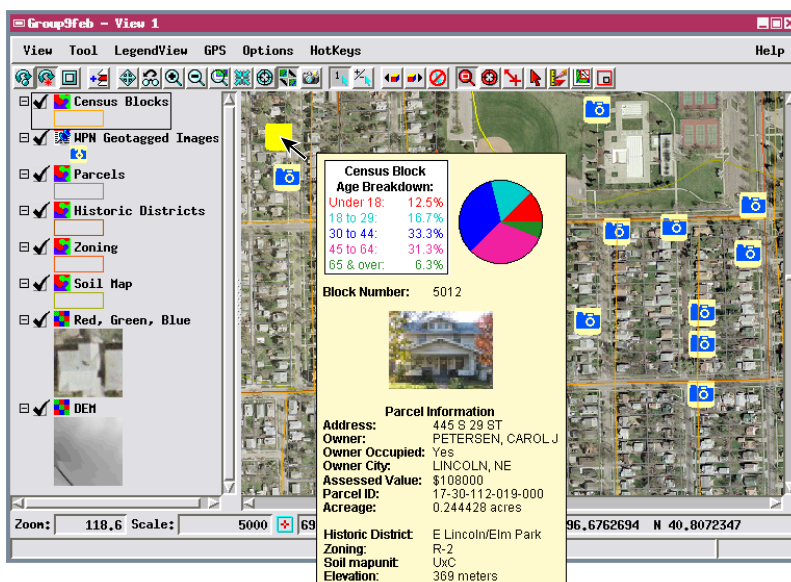


Spatial Display

Designing Complex DataTips

Complex DataTips can be used with a map layout, display layout, and in a TNTAtlas to interactively present detailed information about individual features. The DataTip can merge and present this information for specific features from each geodata layer in the view. A complex DataTip is not limited to showing the attributes of a feature as text; it can also combine attributes from one or multiple layers, present numerical results in graphical form, and include a digital photo of the feature.

The DataTip controls on the individual Layer Controls windows allow you to choose a field in a database table to provide DataTip information for that specific layer and to provide prefix and suffix text if desired. If you choose an image field in a geotagged photo database as the source field, then the relevant photo thumbnail is shown in the DataTip. You can embed formatting codes in the prefix and suffix strings to set font, font style, and alignment for DataTip text; text justification codes can also be used to set horizontal positioning for image thumbnails. More information about formatting codes can be found in the Technical Guide entitled *Adding Styling to DataTips*.



You can include multiple lines of information from the same layer in the DataTip by choosing a string expression field in the database table; the expression in this field can reference information in various fields in the table and format it to multiple text lines with prefix text and control of font styles and alignment. A sample multiline DataTip string expression is shown at the bottom of this page.

For more complex DataTip presentations you can use a display control script. A script gives you more extensive control over DataTip text, allows you to perform computations with attribute values, and can dynamically create graphics such as charts and graphs with the current attribute information. Script functions and class methods are provided to let you create graphics and use them alone in the DataTip or combine the graphics with text information set with the control script and with information from other layers in a complex DataTip. The display control script that creates the pie graphs shown in the DataTip illustrations on this plate is excerpted on the reverse side.

Census Block Age Breakdown:	
Under 18:	16.9%
18 to 29:	15.3%
30 to 44:	23.7%
45 to 64:	27.1%
65 & over:	16.9%

Block Number: 5019

Parcel Information
Address: 705 S 32 ST
Owner: SMITH, RANDALL B
Owner Occupied: Yes
Owner City: LINCOLN, NE
Assessed Value: \$104800
Parcel ID: 17-30-119-021-000
Acreage: 0.154917 acres

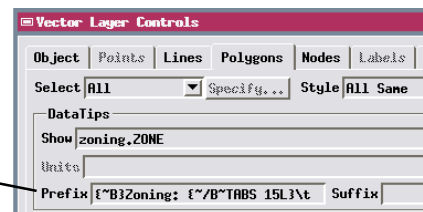
Historic District: Woods Park Bungalow
Zoning: R-2
Soil mapunit: UxC
Elevation: 368 meters

Pie graph with legend and text line for census block layer created by Display Control Script (script excerpts on reverse side of this plate).

Image thumbnail from pinmapped geotagged image database table, center-justified using formatting code embedded in prefix string.

Multiline DataTip for Parcel layer: uses a string expression field in table to create a formatted multiline listing of multiple attributes from same layer. String expression is shown in box below.

Simple DataTip information for several layers (one attribute from each) with formatting codes embedded in prefix string to set bolding and tabs.



String Expression for Sample Multiline DataTip from Single Layer (Parcels)

```
"\n" +
newline to create space above title
"{~CJ~FARIALBD.TTF}Parcel Information{~LJ~FARIAL.TTF}" + "\n" +
centered title text in bold
"{~FARIALBD.TTF}Address:{~FARIAL.TTF~TABS 15L}\t" + CA032904.SITUS_ADDR + "\n" +
"{~FARIALBD.TTF}Owner:{~FARIAL.TTF~TABS 15L}\t" + CA032904.OWNER + "\n" +
"{~FARIALBD.TTF}Owner Occupied:{~FARIAL.TTF~TABS 15L}\t" + CA032904.OwnerOccupied + "\n" +
"{~FARIALBD.TTF}Owner City:{~FARIAL.TTF~TABS 15L}\t" + CA032904.OWN_CITY + ", " + CA032904.OWN_STATE + "\n" +
"{~FARIALBD.TTF}Assessed Value:{~FARIAL.TTF~TABS 15L}\t" + sprintf("%f", CA032904.ASSESSED) + "\n" +
"{~FARIALBD.TTF}Parcel ID:{~FARIAL.TTF~TABS 15L}\t" + CA032904.PRECINCT + "-" + CA032904.SECTION + "-" +
CA032904.BLOCK + "-" + CA032904.PARCEL + "-" + CA032904.SUBPARCEL + "\n" +
"{~FARIALBD.TTF}Acreage:{~FARIAL.TTF~TABS 15L}\t" + NumToStr(POLYSTATS.Area/4046.873) + " acres" + "\n";
```

each additional line sets bold font for prefix, then resets to normal font and tabs over 15 characters before adding text from database field or fields.

Many sample scripts have been prepared to illustrate how you might use the features of the TNT products' scripting language for scripts and queries. These scripts can be downloaded from www.microimages.com/downloads/scripts.htm.

Excerpt from the Age Pie Chart Display Control Script (AgePieChartTip.sml)

procedure called when group or layout is initialized

```
proc OnInitialize () {
```

create RGBA graphics device w/ specified height, width to be used to draw graph and its legend

```
width = 220; height = 120;
dev.Create(height, width);
offset.x = 10; offset.y = 10;
```

set offset of datatip from cursor position (right and down)

procedure called when View is created for group

```
proc OnGroupCreateView (
class GRE_GROUP group,
class GRE_VIEW view
```

```
) {
get layer containing the census blocks by name from the group and get the vector from this layer
```

```
blkLayer = (class GRE_LAYER_VECTOR) group.GetLayerByName("Census Blocks");
DispGetVectorFromLayer(BlkVec, blkLayer);
vecGeoref = GetLastUsedGeorefObject(BlkVec);
}
```

procedure called when DataTip event is triggered; use for drawing pie chart

```
func OnViewDataTipShowRequest (
class GRE_VIEW view,
```

```
class POINT2D point,
class TOOLTIP datatip
){
```

```
datatip.MarginHeight = 5;
datatip.MarginWidth = 5;
```

set the size of the margin between DataTip contents and border in screen pixels

clear graphics device to 100% transparent before drawing

```
local class COLOR transparent(0, 0, 0, 0);
```

transparent color

```
dev.Clear(transparent);
trans = view.GetTransLayerToScreen(blkLayer, 1);
```

get transform from screen to layer

```
ptLayer = trans.ConvertPoint2DFwd(point);
```

cursor position in layer coordinates

get number of parcel polygon under cursor

```
polyNum = FindClosestPoly(BlkVec, ptLayer.x, ptLayer.y, vecGeoref, 0);
```

get age group population percentages from database table for current polygon and convert to degrees of arc out of 360

```
pctU18 = 3.6 * BlkVec.poly[polyNum].AgeGroups.PctUnder18;
pct18to29 = 3.6 * BlkVec.poly[polyNum].AgeGroups.PctFrm18to29;
pct30to44 = 3.6 * BlkVec.poly[polyNum].AgeGroups.PctFrm30to44;
pct45to64 = 3.6 * BlkVec.poly[polyNum].AgeGroups.PctFrm45to64;
pct65over = 3.6 * BlkVec.poly[polyNum].AgeGroups.PctOver64;
```

set variables for positioning drawing elements (in pixels from upper left corner)

```
rectwidth = 120;
hcentRect = rectwidth / 2;
xCent = rectwidth + (width - rectwidth) / 2;
yCent = height / 2;
r = 40;
```

width of rectangle for labels

location of pie chart center

radius of pie chart

```
boxtop = 1;
```

y-coord of top of rectangle for labels

```
fontsize = 12;
linespace = fontsize + 3;
```

```
gc = dev.CreateGC();
```

create graphics context for device for graph and text

fill white rectangle with black border for label background

```
gc.SetColorName("white");
gc.SetLineWidth(1, "pixels");
gc.FillRect(1, boxtop, rectwidth, 110);
```

fill rectangle uleft x, uleft y, width, height

```
gc.SetColorName("black");
gc.DrawRect(1, boxtop, rectwidth-2, 109);
```

draw black rectangle border

```
gc.DrawTextSetFont("ARIALBD.TTF");
gc.DrawTextSetHeightPixels(fontsize);
```

set text parameters

```
gc.TextStyle.RoundWidth = 1;
texty = boxtop + linespace;
```

set vertical position of text

```
color.Name = "black";
```

draw title for legend area

```
gc.SetColor(color);
gc.DrawTextSetColors(color);
title$ = "Census Block";
start = hcentRect - (gc.TextGetWidth(title$) / 2);
gc.DrawTextSimple(title$, start, texty);
```

increment vertical position of text

```
texty += linespace;
```

next line of title

```
title$ = "Age Breakdown:";
```

```
start = hcentRect - (gc.TextGetWidth(title$) / 2);
gc.DrawTextSimple(title$, start, texty);
gc.DrawTextSetHeightPixels(fontsize);
```

reset font parameters

```
gc.DrawTextSetFont("ARIAL.TTF");
```

draw pie wedge for under 18 group in red

```
color.Name = "red";
gc.SetColor(color);
texty += linespace;
```

increment vertical position of text

```
if (pctU18 > 0) {
```

draw wedge w/ centerX, centerY, radiusX, radiusY, startangle, sweepangle

```
gc.FillArcWedge(xCent, yCent, r, r, 0, pctU18);
sum += pctU18;
```

increment sum of wedge angles

```
gc.DrawTextSetColors(color);
gc.DrawTextSimple("Under 18:", 5, texty);
percent$ = sprintf("%.1f%", pctU18 / 3.6);
```

draw label and percentage in same color

compute start position for percent to right-align

```
start = rectwidth-4 - gc.TextGetWidth(percent$);
gc.DrawTextSimple(percent$, start, texty);
```

draw pie wedge for 18 to 29 group in cyan

```
color.Name = "cyan3";
gc.SetColor(color);
texty += linespace;
```

```
if (pct18to29 > 0) {
gc.FillArcWedge(xCent, yCent, r, r, sum, pct18to29);
sum += pct18to29;
}
gc.DrawTextSetColors(color);
gc.DrawTextSimple("18 to 29:", 5, texty);
percent$ = sprintf("%.1f%", pct18to29 / 3.6);
start = rectwidth-4 - gc.TextGetWidth(percent$);
gc.DrawTextSimple(percent$, start, texty);
```

[code for drawing additional three wedges of pie chart and their legend entries omitted here]

```
datatip.AppendImage(dev);
```

append graphic to DataTip

```
datatip.AppendText("\n");
```

add carriage return after graphic for spacing

```
return 0;
```

return 0 to render image in normal DataTip frame along with info from other layers