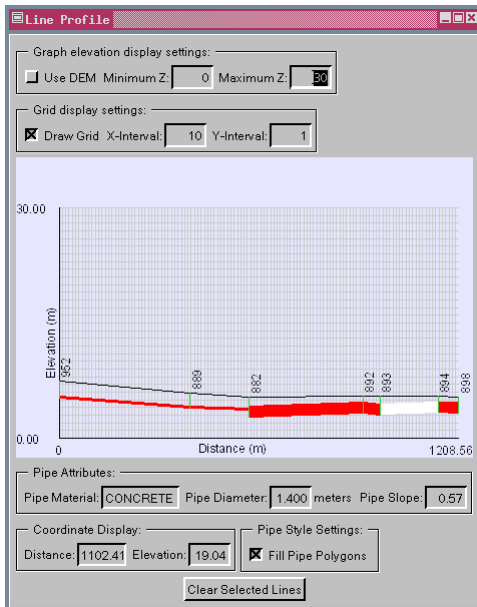
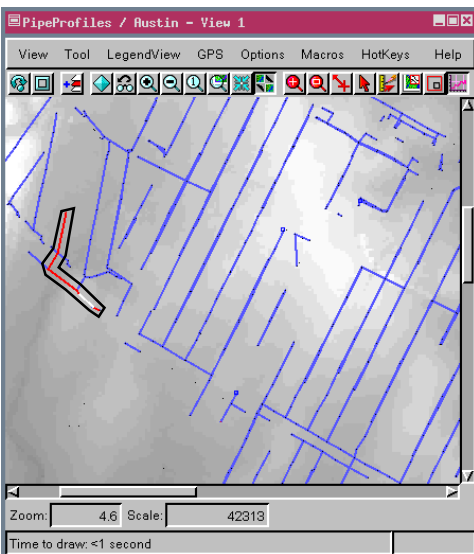


## Sample Tool Script

# Infrastructure Graphical Profile



Update the selected lines displayed in the Line Profile window by right clicking in either the Line Profile or View window.



- set elevation range for graphic
- pipe diameter and length determine pipe polygon dimensions at scale set by elevation range
- get elevation and distance for cursor position in profile graphic
- manhole location drawn as vertical line from pipe bottom to surface
- show surface approximation (connect manhole lines) or draw surface profile from DEM
- Coordinate Display panel provides information about position of cursor when over graphical display

There are many infrastructure applications (for example, water and sewer lines) in which a graphical profile can provide an invaluable visualization tool. DataTips can provide considerable information about the elements at a particular location, however, you may be interested in viewing information for multiple contiguous elements either alone or with additional information, such as elevation or start and end node information along a line. In addition to relative and absolute elevation, a variety of easily understood information can be presented in a sophisticated graphical profile. A Tool Script that incorporates a number of these features is provided as an example that can be readily adapted to your applications. This sample Tool Script lets you select and view connected lines in profile with additional information, such as node attributes for the start and end of the line, pipe material, diameter, length, and so on.

## Selection Features

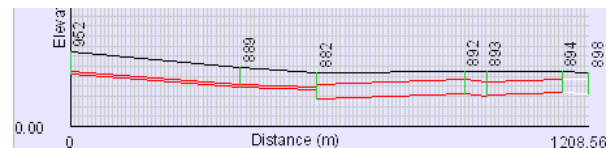
- only allows selection of a sequence of connected lines
- add to either end of connected lines
- selected and active line colors you set are used
- active element highlights in both View and Line Profile window on mouseover in either window
- left click to select lines, right click to update profile graphic
- clear selected lines to start a new set of connected lines

## Attributes

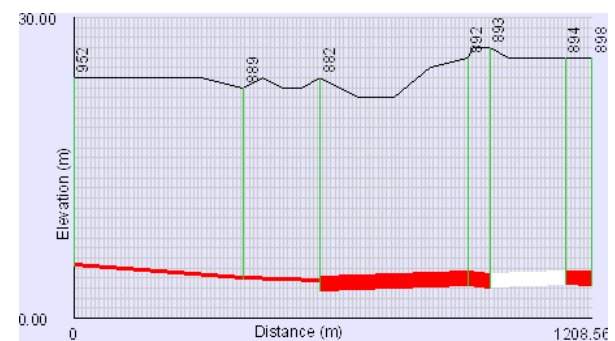
- labels line ends with manhole (node) ID
- presents pipe material, diameter, and slope for the active line
- displays total length of selected lines

## Graphics

- draw pipes as filled or unfilled polygons
- uses active and highlight colors you set from the View/Options menu
- show/hide grid with X and Y intervals you set



unfilled (above), filled (upper left and below)



Choosing to use the DEM alters the graphic from straight line connection from manhole to manhole to a line that represents the changes in surface elevation along the distance of the pipe. The bottom layer is presumed to contain the surface values.

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/freestuf/scripts.htm](http://www.microimages.com/freestuf/scripts.htm).

## Partial Script for Infrastructure Graphical Profiles (LineProfile.sml)

The following excerpt is only a portion of the part of the script that draws the graph section of the Line Profile window.

```
func class POLYLINE constructPipeFace(class POLYLINE bottom,
class POLYLINE top)
{
    local class POLYLINE ret;
    if (bottom.GetNumPoints() != top.GetNumPoints())
    {
        numeric fillPipes = 0;
        return ret;
    }
    local numeric i;
    for (i=0; i<bottom.GetNumPoints(); i=i+2)
    {
        ret.AppendVertex(bottom.GetVertex(i));
        ret.AppendVertex(bottom.GetVertex(i+1));
        ret.AppendVertex(top.GetVertex(i+1));
        ret.AppendVertex(top.GetVertex(i));
        ret.AppendVertex(bottom.GetVertex(i));
    }
    return ret;
}
func class POLYLINE constructPipeTop(class POLYLINE pipeBottom,
class POLYLINE origLine)
{
    local class POLYLINE newLine;
    local class POINT2D tmp;
    local numeric i;
    for (i=0; i<origLine.GetNumPoints(); i++)
    {
        local numeric vertex1, vertex2;
        if (i==0) vertex1 = i;
        else vertex1 = i-1;
        vertex2 = i;
        local class POINT2D p1 = origLine.GetVertex(vertex1);
        local class POINT2D p2 = origLine.GetVertex(vertex2);
        if (p1==p2)
        {
            vertex1++;
            vertex2++;
            if (i==0) vertex1 = i;
        }
        local numeric elemNum = findClosestLineElement(origLine,
vertex1, vertex2);

        local numeric elevation = readLineTableRecord(elemNum,
"PIPE_DIAM")/1000 + pipeBottom.GetVertex(i).y;
        if (!IsNull(elevation))
        {
            tmp.x = pipeBottom.GetVertex(i).x;
            tmp.y = elevation;
            newLine.AppendVertex(tmp);
        }
    }
    return newLine;
}
```

function called to construct the pipe face

function called to construct the pipe top

get elevation from end node

```
func class STRINGLIST constructManholeNames(class POLYLINE
origLine)
{
    local class STRINGLIST manholeNames;
    local class POINT2D prevPoint;

    local numeric i;
    for (i=0; i<origLine.GetNumPoints(); i++)
    {
        local class POINT2D point1 = origLine.GetVertex(i);
        local string manholeLabel = "";
        if (point1 != prevPoint)
        {
            local numeric elemNum = FindClosestNode(lineVector,
point1.x, point1.y);
            manholeLabel = readNodeTableRecordStr(elemNum,
"MH_ID");
        }
        manholeNames.AddToEnd(manholeLabel);
        prevPoint = point1;
    }
    return manholeNames;
}
func class POLYLINE constructManholeDepth(class POLYLINE
pipeBottom, class POLYLINE origLine)
{
    local class POLYLINE newLine;
    local class POINT2D tmp;

    local numeric i;
    for (i=0; i<origLine.GetNumPoints(); i++)
    {
        local class POINT2D point1 = origLine.GetVertex(i);
        local numeric elemNum = FindClosestNode(lineVector, point1.x,
point1.y);
        local numeric elevation = readNodeTableRecord(elemNum,
"MH_INVERT");
        tmp.x = pipeBottom.GetVertex(i).x;
        tmp.y = elevation;
        newLine.AppendVertex(tmp);
    }
    return newLine;
}
func class POLYLINE constructPipeBottom(class POLYLINE
origLine)
{
    local string field = "PIPE_I_SN";
    local class POLYLINE newLine;
    if (origLine.GetNumPoints()<1) return newLine;
    local class POINT2D tmp;
    local numeric i;
    local numeric distance=0;
    local numeric elemNum = findClosestLineElement(origLine, 0, 1);
    if (isLineReversed(elemNum))
    {
        field = "CAN_I_DS";
    }
}
```

function called to construct manhole names

draw the manhole names at the manhole tops

function called to construct manhole depth

get elevation from end node

function called to construct the pipe bottom

check to see if line is reversed