Dynamic Legending

Robert Pincus

Abstract

Riverside County is one of the largest counties in California and County planners have developed over 190 unique land use zones. ArcIMS "out-of-the-box" legending displays all of the unique zones for a given layer. With so many zones in the land use layer, the legend is cumbersome and impractical for the user. Riverside County GIS staff have developed a methodology that creates a legend displaying only those features visible in the map window. This paper will discuss the methodology of this dynamic legend as well the pros and cons of implementating this feature.

Introduction

The process of creating a dynamic legend was developed as part of a pilot project to determine if it was possible to create a legend that would show only features for visible layers in the map extent. The pilot project website also served as the first customized ArcIMS project that Riverside County GIS application staff was to do. Since the development of this technique the process and coding has been refined, but the process outlined in this paper is still applicable.

During the last quarter of 2002, staff were assigned the task of creating an ArcIMS viewer that would enable county constituents to find which county supervisorial district they lived in. Program requirements dictated that the website had to be simple and easy to use and be an HTML Viewer. It was also decided to add some functionality to the website that would later be used in the future ArcIMS websites. One the these websites is the replacement for the County's AML based flagship program Geo_Info.

A part of Geo_Info's core functionality is to produce maps that only show those features of layers that are in the map extent. This is an important function for the county since the zoning layer itself contains more than 190 different zoning types. This functionality would have to be replicated on the ArcIMS websites because the default legending and table of contents would make the websites difficult to view. Figure 1 shows an example of this. In the map frame, five zones are shown from the county's zoning layer. However, the ArcIms' default legending displays all the zoning types. The user would have to scroll through the legend to find the item that best matches the displayed zoning types. Also with so many zoning types, it would be hard to find a particular shading/pattern from the long list.

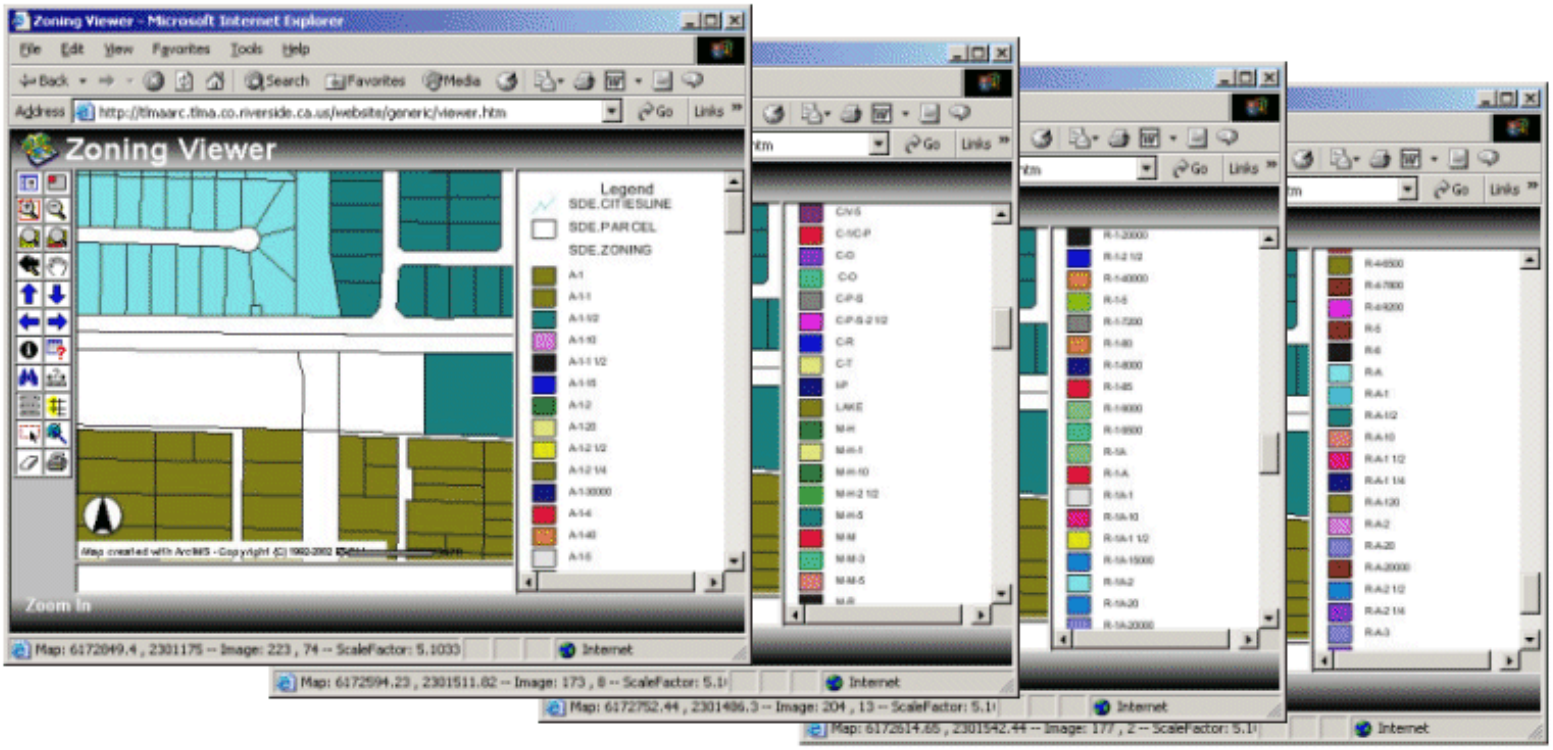

Figure 1. Example of a scrolling list in the legend.

Staff agreed that the generic website from the ArcIMS designer needed to be modified to make it more user friendly. Figure 2 shows that from the six frames in the generic html viewer, staff settled on a viewer with three frames, the ToolFrame, MapFrame, and TOCFrame.

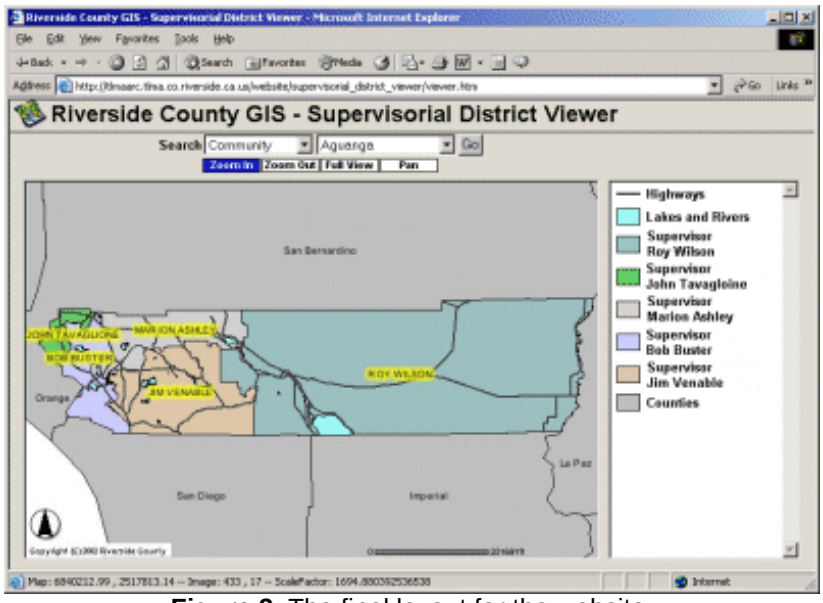

Figure 2. The final layout for the website.

The Steps Used to Create Dynamic Legending

- Create legend swatches for each feature in each layer.
- Create a new HTML document inside TOCFrame after the map has been reloaded.
- Step through the visible layers and create a query to find out how many features are in the envelope.
- If there are features, then step through the server's reply to find the feature and get to corresponding swatch. Use JavaScript to add to the legend document.
- When all the visible layers have been checked, close the HTML document.

Creating Swatches

This method of creating a dynamic legend relied heavily on custom images that were to make up a HTML document that was loaded in the TOCFrame frame of the viewer. The swatches contained two parts. The left side is the symbol for the feature being displayed and the right side contains the name of the feature. While creating these swatches were time consuming, the advantage was in its customizable format. It is important to note that the swatches should each be the same size. This aided in the coding for the HTML Image tags when writing the swatch images to the TOCFrame. For this exercise, two different size swatches were created. The "City of Hemet" can be written on one line and was created at 180 x 24 pixels. However the "City of Desert Hot Springs" needed two lines and was created at 180 x 38 pixels. Figure 3 shows the two swatches.

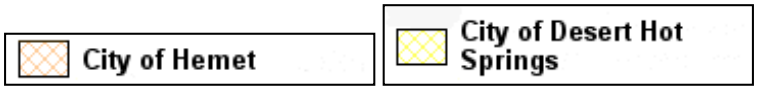

Figure 3. Comparison of the two sizes of images used for the dynamic legend.

Creating A New HTML Document in TOCFrame

The dynamic legend process started with the buildlegend function. The buildlegend function was called from the afterMapRefresh function in MapFrame.htm, so that the buildlegend function is run each time the map was drawn.

All customized code written by county staff was added to the JavaScript file rivco.js to avoid interference with IMS Designer generated code. Rivco.js was added as the last JavaScript reference in the Header section of MapFrame.htm. The code to run the buildlegend function in MapFrame.htm is:

```
function afterMapRefresh() {
  parent.MapFrame.buildlegend();
}
```

The buildlegend function opened TOCFrame for writing with the line:

```
  parent.TOCFrame.document.open();
  parent.TOCFrame.document.writeln('<html><meta http-equiv="Content-Type" content="text/html; ');
  parent.TOCFrame.document.writeln('charset=' + charSet + '"><head>');
  parent.TOCFrame.document.writeln('<style type="text/css">a {text-decoration:none;}</style>');
  parent.TOCFrame.document.writeln('</head>');
  parent.TOCFrame.document.writeln('<body BGCOLOR="White" text="Black" leftmargin=0 topmargin=0 ');
  parent.TOCFrame.document.writeln('rightmargin=0 link="Black" vlink="Black" alink="Black">');
  parent.TOCFrame.document.writeln('<center>');
```

Stepping Through Visible Layers

The next step of the buildlegend function was to gather all the layers currently visible in the map extent and add them to an array. Layer visibility was determined by the map scale factors set in the configuration file for the website. The code for this is:

```
  for (var i=0;i     if ((!t.hideLayersFromList) || ((t.hideLayersFromList) && (!t.noListLayer[i]))) {
     if ((t.listAllLayers) || ((t.mapScaleFactor>=t.LayerMinScale[i]) && (t.mapScaleFactor<=t.LayerMaxScale[i]))) {
       if (t.LayerVisible[i] == 1) {
         sd$layers[sd$layerindex] = t.LayerID[i];
         sd$layerindex = sd$layerindex + 1;
       }
     }
   }
  }
  sd$numberoflayers = sd$layerindex;
```

Once the array was populated, the contents of the first element were sent the to getthefeatures function. This function took the argument thelayerid and the global variables that made up the envelope to create the variable XMLRequest. The code for the getthefeatures function is shown below:

```
  function getthefeatures(thelayerid) {
    var theType = 1000;
    var URLString = imsURL + '&CustomService=Query';
    var XMLRequest = '<ARCXML version="1.1"><REQUEST>\n';
```

```
  XMLRequest = XMLRequest + '<GET_FEATURES attributes="true" beginrecord="0" featurecount="30" ';
  XMLRequest = XMLRequest + 'outputmode="xml" geometry="false" envelope="false" >\n';
  XMLRequest = XMLRequest + '<LAYER id="' + thelayerid + '" /><SPATIALQUERY >< ';
  XMLRequest = XMLRequest + 'SPATIALFILTER relation="area_intersection">\n';
  XMLRequest = XMLRequest + '<ENVELOPE minx="' + theminx + '" miny="' + theminy + '" ';
  XMLRequest = XMLRequest + 'maxx="' + themaxx + '" maxy="' + themaxy + '" />\n';
  XMLRequest = XMLRequest + '</SPATIALFILTER></SPATIALQUERY></GET_FEATURES> ';
  XMLRequest = XMLRequest + '</REQUEST></ARCXML>\n';

  sendToServer(URLString,XMLRequest,theType);
}
```

Figure 4 shows the reply from the server that was sent to case 1000 in the processXML function in the aimsXML.js file. Case 1000 (see below) took the reply from the server and checked to see if any features were found while running the ArcIMS justGetFeatureCount function. If no features were found, then increment sd$layerindex by 1 and check to see if the program cycled through the entire array. If it has not finished, then run the getthefeatures function for the next element in the sd$layers array.

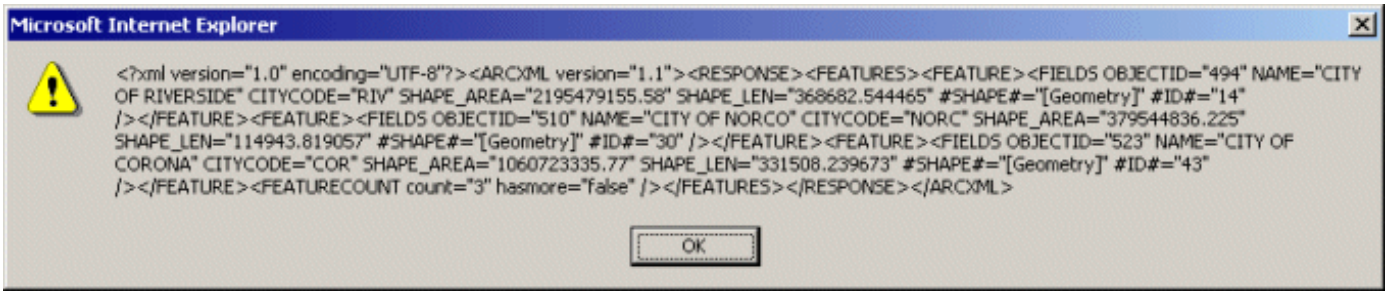

Figure 4. The reply from the ArcIMS server.

```
case 1000:
var thefeaturecount = justGetFeatureCount(theReply);
if (thefeaturecount > 0) {
  switch (sd$layers[sd$layerindex]) {
    case 'cities':
      citieslegend();
      break;
    case 'sup_districts':
      sup_districtslegend();
      break;
    case 'counties':
      countieslegend();
      break;
    case 'highways':
      highwayslegend();
      break;
    case 'parcels':
      parcelslegend();
      break;
    case 'roads':
      roadslegend();
      break;
    case 'waterbodies':
      waterbodieslegend();
      break;
  }
}
sd$layerindex = sd$layerindex + 1;
if (sd$layerindex < sd$numberoflayers) {
  getthefeatures(sd$layers[sd$layerindex]);
}
```

Writing Features To The New Document

If features were found, the value of the current element in the sd$layers array was used to determine which feature the reply refers to. This step had to be done because there is no reference to the layer in the server reply.

A JavaScript switch statement acted as a filter to control customized functions depending on the layer in the array element location. Refering back to Figure 4, shows the reply from the server, three of the twenty five cities were found in the current envelope. The feature count was three, so the switch statement checked to see what the value of the sd$layers array element is. In this case, the content of sd$layers[sd$layerindex] was "cities", thus the citieslegend function was run.

The citieslegend function stepped through the global variable theReply and extracted the name of the city. The name was added to a temporary array if it does not all ready exist in it. The end result was an array of unique city names. The array was next stepped through, appending the city name to the Image Tag HTML code that was written to the TOCFrame.

```
function citieslegend() {
  var myarray = new Array();
  var zork;
  var a;
  var b;
  var citytest = false;
  var startpos = 0;
  var endpos = 0;
  // Loop through theReply to extract the features
  for (var a = 1; a <= thefeaturecount; a++) {
    startpos = theReply.indexOf("CITYNAME=",endpos);
    startpos = startpos + 10;
    endpos = theReply.indexOf("SDE.CITIES",startpos);
    endpos = endpos - 2;
    thefeature = theReply.substring(startpos,endpos);
    // Cycle through myarray to compare each element with the feature.
    // If the feature is not in the array, then add it to myarray.
    citytest = true
    if (thefeature.length > 0) {
      zork = myarray.length;
      for (b = 0; b <= zork; b++) {
        if (thefeature == myarray[b]) {
          citytest = false;
        }
      }
      if (citytest) {
        myarray[zork] = thefeature;
      }
    }
  }
  myarray = myarray.sort();
  // Start to write the cities legend as an HTML document to TOCFrame
  var grog;
  var theimage;
  // Cycle through myarray and write to myarray[] to the HTML document.
  for (a = 0; a < myarray.length; a++) {
    myarray[a] = myarray[a].replace(/\s/g,"");
    if ( myarray[a]!="") {
     var b = myarray[a].toLowerCase();
      if (b == "deserthotsprings") {
        theimage = "<IMG ALT=City SRC=../website/images/cities/" +";
        myarray[a].toLowerCase() + ".gif HEIGHT=38 WIDTH=180 /><br>";
      }
      else {
        theimage = "<IMG ALT=City SRC=../website/images/cities/" + ";
        myarray[a].toLowerCase() + ".gif HEIGHT=24 WIDTH=180 /><br>";
      }
      parent.TOCFrame.document.writeln(theimage);
    }
  }
}
```

Closing The Document

After the the last layer has been drawn write these lines to the HTML document to close it:

```
parent.TOCFrame.document.writeln('</body></html>');
parent.TOCFrame.document.close();
```

After the new HTML document is closed, the website may look like Figure 5. For example, in the envelope shown the map only shows one of the twenty five cities in county, two of the five supervisors, roads, parcel boundaries, and no water features. Only these features are shown in the legend.

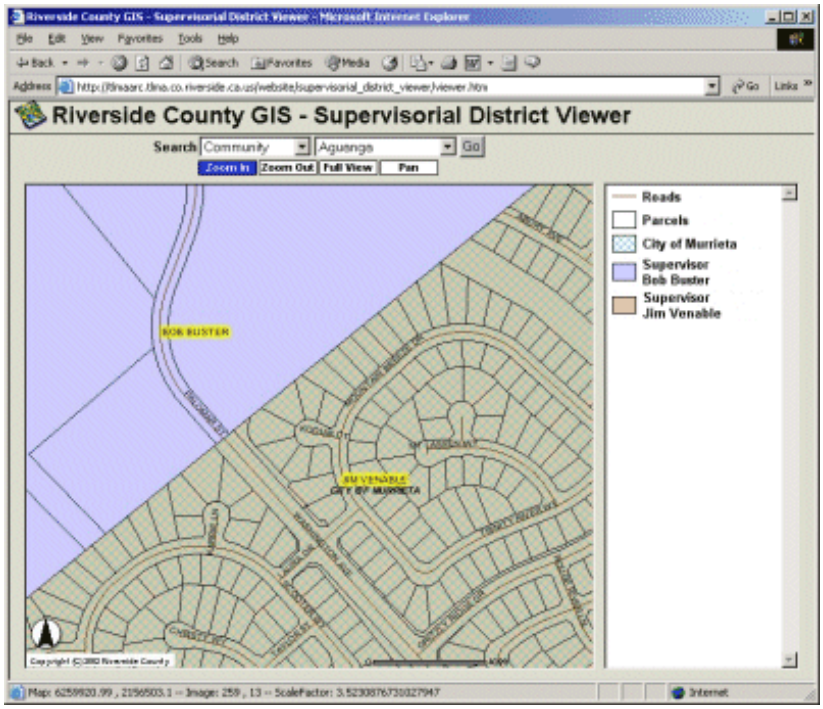

Figure 5. Final product of the process to create a dynamic legend.

Discussion

Riverside County staff has successfully developed a methodology to create an ArcIMS legending process that only displays those features displayed in the current envelope.

Several features can be seen in Figure 5 that enhance the readablity of the map as compared to the default legending of ArcIMS. The legend is easier to read because of the customized images used. More control is availible to create the legend swatch. Only four of the eleven possible layers are in the legend, because only those four layers are visible in the envelope. There is no need to display a layer in the legend if it can not be seen. Finally, of those layers that can be seen, not all the features are displayed. In this example, only two of the twenty five cities in Riverside County and two of the five Supervisors are in the legend.

This is not a perfect methodology. There are three main deficiencies that need to be addressed before this methodology should be implemented. The first is the increased number of transactions between the client and the ArcIMS server. Depending upon the number of layers and connection speed, the increase in time before the website completely refreshes may not be acceptable. The second is the development and maintenance of the swatches used for the website. Again, depending the number of layers and features in the website, it may be prohibitive to create and maintain so many images. Finally, code needs to be maintained as well. Whenever a new layer is introduced to the website, customized code will be needed to extract the features from the server's feature inquiry in order to obtain the correct images to be added to the legend.

Acknowledgments

I thank Bob Bell for his help on the layout of the pilot project website, HTML and JavaScript programming techniques.

For more information contact...

Robert Pincus
Business Systems Analyst
County of Riverside
4080 Lemon St. 7th Floor
P.O. Box 1090
Riverside, Ca. 92502-1090
Tel: 909-955-1875
Fax: 909-955-6879
Email: rpincus@co.riverside.ca.us