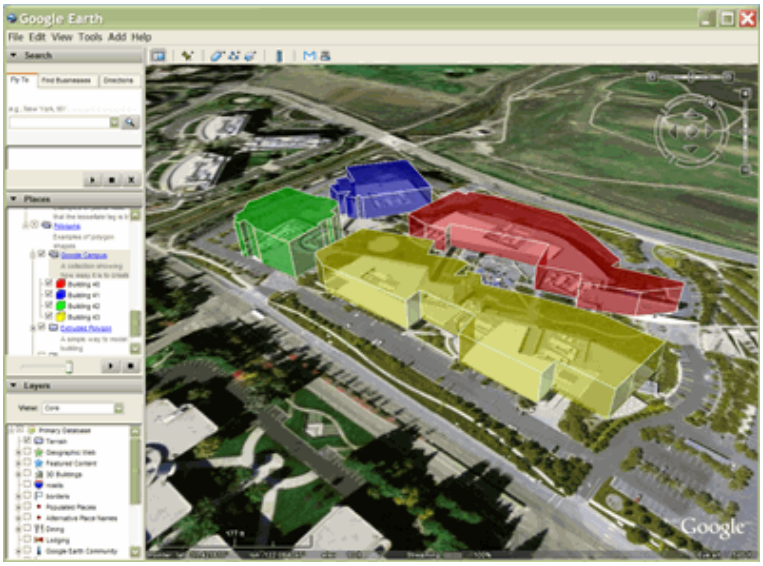
Google code

## KML

# KML Tutorial

KML is a file format used to display geographic data in an Earth browser such as Google Earth, Google Maps, and Google Maps for mobile. KML uses a tag-based structure with nested elements and attributes and is based on the XML standard. All tags are case-sensitive and must be appear exactly as they are listed in the KML Reference. The Reference indicates which tags are optional. Within a given element, tags must appear in the order shown in the Reference.




If you're new to KML, explore this document and the accompanying samples files (*SamplesInEarth* and *SamplesInMaps*) to begin learning about the basic structure of a KML file and the most commonly used tags. The first section describes features that can be created with the Google Earth user interface. These features include placemarks, descriptions, ground overlays, paths, and polygons. The second section describes features that require authoring KML with a text editor. When a text file is saved with a *.kml* or *.kmz* extension, Earth browsers know how to display it.

**Tip:** To see the KML "code" for a feature in Google Earth, you can simply right-click the feature in the 3D Viewer of Google Earth and select Copy. Then Paste the contents of the clipboard into any text editor. The visual feature displayed in Google Earth is converted into its KML text equivalent. Be sure to experiment with this feature.

All of the examples described here are in the KML Samples file. Begin by downloading that file to view the examples in Google Earth.

## For More Information

The KML 2.2 Reference provides details about the KML file format. If you're familiar with XML, you will also be interested in the KML 2.2 Schema.

For a discussion of how to use some of the key features in KML, see the Developer's Guide.

## Table of Contents

## 1 Basic KML Documents

The simplest kind of KML documents are those that can be authored directly in Google Earth—that is, you don't need to edit or create any KML in a text editor. Placemarks, ground overlays, paths, and polygons can all be authored directly in Google Earth.

### Placemarks

A Placemark is one of the most commonly used features in Google Earth. It marks a position on the Earth's surface, using a yellow pushpin as the icon. The simplest Placemark includes only a <Point> element, which specifies the location of the Placemark. You can specify a name and a custom icon for the Placemark, and you can also add other geometry elements to it.

Open the KML Samples file in Google Earth and expand the Placemarks subfolder. This folder includes three different types of placemark: *simple*, *floating*, and *extruded*. The KML code for the simple placemark looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Simple placemark</name>
    <description>Attached to the ground. Intelligently places itself
       at the height of the underlying terrain.</description>
    <Point>
      <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

The structure of this file breaks down as follows:

- An XML header. This is line 1 in every KML file. No spaces or other characters can appear before this line.
- A KML namespace declaration. This is line 2 in every KML 2.2 file.
- A Placemark object that contains the following elements:
  - A *name* that is used as the label for the Placemark
  - A *description* that appears in the "balloon" attached to the Placemark
  - A *Point* that specifies the position of the Placemark on the Earth's surface (*longitude, latitude,* and optional *altitude*)

If you were wondering where the Placemark is, it's right over Google's Building 41, where we developed Google Earth!

What users commonly think of as a "placemark" in Google Earth is actually a <Placemark> element with a <Point> child in KML. A Point Placemark is the only way to draw an icon and label in the 3D Viewer of Google Earth. By default, the icon is the familiar yellow pushpin. In KML, a <Placemark> can contain one or more geometry elements, such as a LineString, Polygon, or Model. But only a <Placemark> with a Point can have an icon and label. The Point is used to place the icon, but there is no graphical representation of the Point itself.

### Descriptive HTML in Placemarks

The KML Samples file has an example of almost everything you can do with Placemark text. You can add links, font sizes, styles, and colors, and specify text alignment and tables. If you'd like to see the full list, copy and paste the "Descriptive HTML" Placemark example (in the Styles and Markup folder) into a text editor.

#### Auto-Markup in Google Earth (Release 4.0 and later)

Google Earth 4.0 has an auto-markup feature that automatically converts text such as *www.google.com* into active hyperlinks that the user can click. Text inside the <description> tag, the <Snippet> tag, and the <text> element of <BalloonStyle> are all automatically transformed into standard HTTP hyperlinks. You don't need to add the <a href= ...> tags yourself.

### Using the CDATA Element

If you want to write standard HTML inside a <description> tag, you can put it inside a CDATA tag. If you don't, the angle brackets need to be written as entity references to prevent Google Earth from parsing the HTML incorrectly (for example, the symbol **>** is written as **>** and the symbol **<** is written as **<**). This is a standard feature of XML and is not unique to Google Earth.

Consider the difference between HTML markup with CDATA tags and without CDATA. First, here's the <description> with CDATA tags:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <Placemark>
      <name>CDATA example</name>
      <description>
        <![CDATA[
          <h1>CDATA Tags are useful!</h1>
          <p><font color="red">Text is <i>more readable</i> and
          <b>easier to write</b> when you can avoid using entity
          references.</font></p>
        ]]>
      </description>
      <Point>
        <coordinates>102.595626,14.996729</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

And here's the <description> without CDATA tags, so that special characters must use entity references:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <Placemark>
      <name>Entity references example</name>
      <description>
                &lt;h1&gt;Entity references are hard to type!&lt;/h1&gt;
                &lt;p&gt;&lt;font color="green"&gt;Text is
          &lt;i&gt;more readable&lt;/i&gt;
          and &lt;b&gt;easier to write&lt;/b&gt;
          when you can avoid using entity references.&lt;/font&gt;&lt;/p&gt;
      </description>
      <Point>
        <coordinates>102.594411,14.998518</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

## Ground Overlays

Ground overlays enable you to "drape" an image onto the Earth's terrain. The <Icon> element contains the link to the *.jpg* file with the overlay image. Here is the example ground overlay in the KML Samples file, which shows Mount Etna erupting in 2001:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
```

```
  <Folder>
    <name>Ground Overlays</name>
    <description>Examples of ground overlays</description>
    <GroundOverlay>
      <name>Large-scale overlay on terrain</name>
      <description>Overlay shows Mount Etna erupting
          on July 13th, 2001.</description>
      <Icon>
        <href>http://code.google.com/apis/kml/documentation/etna.jpg</href>
      </Icon>
      <LatLonBox>
        <north>37.91904192681665</north>
        <south>37.46543388598137</south>
        <east>15.35832653742206</east>
        <west>14.60128369746704</west>
        <rotation>-0.1556640799496235</rotation>
      </LatLonBox>
    </GroundOverlay>
  </Folder>
</kml>
```

Notice that the file begins with the same two lines as the first example: the XML header and KML namespace declaration.

This example uses a Folder (titled "Ground Overlays") as a mechanism to group and label its contents. Notice how the Folder appears in the Places panel when you load the KML Samples file into Google Earth.

The positioning of a ground overlay is controlled by the <LatLonBox> tag. Bounding values are given for the north and south latitudes, and east and west longitudes. In addition, rotation values are given for images whose *y*-axis doesn't coincide with grid north. This example uses a JPEG image for the overlay. Google Earth also supports BMP, GIF, TIFF, TGA, and PNG formats.

## Paths

Many different types of paths can be created in Google Earth, and it is easy to be very creative with your data. In KML, a path is created by a <LineString> element. Take a look at the "Absolute Extruded" example in the Paths folder and you can see how the shape has been generated by the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <name>Paths</name>
    <description>Examples of paths. Note that the tessellate tag is by
default
      set to 0. If you want to create tessellated lines, they must be
authored
      (or edited) directly in KML.</description>
    <Style id="yellowLineGreenPoly">
      <LineStyle>
        <color>7f00ffff</color>
        <width>4</width>
      </LineStyle>
      <PolyStyle>
        <color>7f00ff00</color>
      </PolyStyle>
    </Style>
    <Placemark>
      <name>Absolute Extruded</name>
      <description>Transparent green wall with yellow outlines</description>
      <styleUrl>#yellowLineGreenPoly</styleUrl>
      <LineString>
        <extrude>1</extrude>
        <tessellate>1</tessellate>
        <altitudeMode>absolute</altitudeMode>
        <coordinates> -112.2550785337791,36.07954952145647,2357
          -112.2549277039738,36.08117083492122,2357
```

```
          -112.2552505069063,36.08260761307279,2357
          -112.2564540158376,36.08395660588506,2357
          -112.2580238976449,36.08511401044813,2357
          -112.2595218489022,36.08584355239394,2357
          -112.2608216347552,36.08612634548589,2357
          -112.262073428656,36.08626019085147,2357
          -112.2633204928495,36.08621519860091,2357
          -112.2644963846444,36.08627897945274,2357
          -112.2656969554589,36.08649599090644,2357
        </coordinates>
      </LineString>
    </Placemark>
  </Document>
</kml>
```

Notice how it is really just one line drawn at altitude above the ground. The **<tessellate>** tag breaks the line up into smaller chunks, and the **<extrude>** tag extends the line down to the ground.

### Polygons

You can use Polygons to create simple buildings and other shapes. Check out the Polygons folder in the KML Samples file for examples.

The Pentagon example is generated by drawing simple inner and outer shells and then extruding them down to the ground. Here is the code :

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>The Pentagon</name>
    <Polygon>
      <extrude>1</extrude>
      <altitudeMode>relativeToGround</altitudeMode>
      <outerBoundaryIs>
        <LinearRing>
          <coordinates>
            -77.05788457660967,38.87253259892824,100
            -77.05465973756702,38.87291016281703,100
            -77.05315536854791,38.87053267794386,100
            -77.05552622493516,38.868757801256,100
            -77.05844056290393,38.86996206506943,100
            -77.05788457660967,38.87253259892824,100
          </coordinates>
        </LinearRing>
      </outerBoundaryIs>
      <innerBoundaryIs>
        <LinearRing>
          <coordinates>
            -77.05668055019126,38.87154239798456,100
            -77.05542625960818,38.87167890344077,100
            -77.05485125901024,38.87076535397792,100
            -77.05577677433152,38.87008686581446,100
            -77.05691162017543,38.87054446963351,100
            -77.05668055019126,38.87154239798456,100
          </coordinates>
        </LinearRing>
      </innerBoundaryIs>
    </Polygon>
  </Placemark>
</kml>
```

## 2 Advanced KML Documents

This section describes some of the KML elements that must be authored using a text editor, such as shared styles for geometry, highlighted icons for Placemarks, and screen overlays. Authoring KML "by hand" is a bit more advanced than using the Google Earth interface to create and modify features, but with a small amount of practice, most users are comfortable editing KML files to add these effects.

## Styles for Geometry

Once you've created features within Google Earth and examined the KML code Google Earth generates, you'll notice how styles are an important part of how your data is displayed. Power users will want to learn how to define their own styles.

If you define a Style at the beginning of a KML Document and also define an ID for it, you can use this style in Geometry, Placemarks, and Overlays that are defined elsewhere in the Document. Because more than one element can use the same Style, styles defined and used in this way are referred to as *shared styles*. You define a given Style once, and then you can reference it multiple times, using the <styleUrl> element. If the Style definition is within the same file, precede the Style ID with a # sign. If the Style definition is in an external file, include the complete URL in the <styleUrl> element.

The KML Samples file contains a number of shared styles, each defined with an ID at the beginning of the file. Note that it's easiest if your IDs are descriptive strings so that you can easily tell what their effect is. Here's an example of a style ("transBluePoly") that defines a transparent blue color for the polygon faces and a line width of 1.5 (and default color of white) for the edges of the polygon. This style is used by Building 41 in the Google Campus example (in the Polygons folder):

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <Style id="transBluePoly">
      <LineStyle>
        <width>1.5</width>
      </LineStyle>
      <PolyStyle>
        <color>7dff0000</color>
      </PolyStyle>
    </Style>
    <Placemark>
      <name>Building 41</name>
      <styleUrl>#transBluePoly</styleUrl>
      <Polygon>
        <extrude>1</extrude>
        <altitudeMode>relativeToGround</altitudeMode>
        <outerBoundaryIs>
          <LinearRing>
            <coordinates> -122.0857412771483,37.42227033155257,17
              -122.0858169768481,37.42231408832346,17
              -122.085852582875,37.42230337469744,17
              -122.0858799945639,37.42225686138789,17
              -122.0858860101409,37.4222311076138,17
              -122.0858069157288,37.42220250173855,17
              -122.0858379542653,37.42214027058678,17
              -122.0856732640519,37.42208690214408,17
              -122.0856022926407,37.42214885429042,17
              -122.0855902778436,37.422128290487,17
              -122.0855841672237,37.42208171967246,17
              -122.0854852065741,37.42210455874995,17
              -122.0855067264352,37.42214267949824,17
              -122.0854430712915,37.42212783846172,17
              -122.0850990714904,37.42251282407603,17
              -122.0856769818632,37.42281815323651,17
              -122.0860162273783,37.42244918858722,17
              -122.0857260327004,37.42229239604253,17
              -122.0857412771483,37.42227033155257,17
            </coordinates>
          </LinearRing>
        </outerBoundaryIs>
      </Polygon>
    </Placemark>
  </Document>
</kml>
```

Note that the <Style> element is a child of <Placemark> (not of the Geometry it affects).

## Styles for Highlighted Icons

The "Highlighted Icon " in the Styles and Markup folder shows you how to create a roll-over effect with KML. The Document defines two styles, one for the "normalPlacemark" and one for the "highlightPlacemark" (shown when the cursor rolls over the Icon). The <StyleMap> element has two key/value pairs that map each icon style to an icon state. There are two icon states: *normal* and *highlight*.

The basic steps shown here are as follows:

1. Define a <Style> for the Placemark's normal icon, and assign an ID to it (here, "normal Placemark"). The <Style> includes an <Icon> with an <href> to the actual image to use, as shown below.
2. Define a <Style> for the Placemark's highlight icon and assign an ID to it (here, "highlightPlacemark").
3. Create the <StyleMap> element and assign an ID to it ("exampleStyleMap"). The Placemark will refer to this ID.
4. In the <StyleMap> element, specify "#normalPlacemark" for the *normal* state.
5. In the <StyleMap> element, specify "#highlightPlacemark" for the *highlight* state.
6. In the Placemark, add a <styleUrl> element that refers to "#exampleStyleMap."

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <name>Highlighted Icon</name>
    <description>Place your mouse over the icon to see it display the new
icon</description>
    <Style id="highlightPlacemark">
      <IconStyle>
        <Icon>
          <href>http://maps.google.com/mapfiles/kml/paddle/red-
stars.png</href>
        </Icon>
      </IconStyle>
    </Style>
    <Style id="normalPlacemark">
      <IconStyle>
        <Icon>
          <href>http://maps.google.com/mapfiles/kml/paddle/wht-
blank.png</href>
        </Icon>
      </IconStyle>
    </Style>
    <StyleMap id="exampleStyleMap">
      <Pair>
        <key>normal</key>
        <styleUrl>#normalPlacemark</styleUrl>
      </Pair>
      <Pair>
        <key>highlight</key>
        <styleUrl>#highlightPlacemark</styleUrl>
      </Pair>
    </StyleMap>
    <Placemark>
      <name>Roll over this icon</name>
      <styleUrl>#exampleStyleMap</styleUrl>
      <Point>
        <coordinates>-122.0856545755255,37.42243077405461,0</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

## Screen Overlays

Screen overlays cannot be authored directly within Google Earth and are thus more difficult to create than ground overlays. A comprehensive collection of samples is included in the Screen Overlays folder in the KML Samples file.

As an example, enable the "Absolute Positioning: Top left" folder in the KML Samples file and you will see a screen overlay at the top left of the view window. This was created with the following KML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <ScreenOverlay>
    <name>Absolute Positioning: Top left</name>
    <Icon>

<href>http://code.google.com/apis/kml/documentation/top_left.jpg</href>
    </Icon>
    <overlayXY x="0" y="1" xunits="fraction" yunits="fraction"/>
    <screenXY x="0" y="1" xunits="fraction" yunits="fraction"/>
    <rotationXY x="0" y="0" xunits="fraction" yunits="fraction"/>
    <size x="0" y="0" xunits="fraction" yunits="fraction"/>
  </ScreenOverlay>
</kml>
```

Positioning is controlled by mapping a point in the image specified by <overlayXY> to a point on the screen specified by <screenXY>. In this case, the top-left corner of the image (0,1) has been made coincident with the same point on the screen.

Check the other examples in the folder to see how it is possible to obtain other fixed positions, and to create images that size dynamically with screen size. (Note that *xunits* and *yunits* can also be specified as "pixels" for precision control.) For further detail, see the KML 2.2 Reference.

## Network Links

A network link contains a <Link> element with an <href> (a hypertext reference) that loads a file. The <href> can be a local file specification or an absolute URL. Despite the name, a <NetworkLink> does not necessarily load files from the network.

The <href> in a link specifies the location of any of the following:

- An image file used by icons in icon styles, ground overlays, and screen overlays
- A model file used in the <Model> element
- A KML or KMZ file loaded by a Network Link

The specified file can be either a local file or a file on a remote server. In their simplest form, network links are a useful way to split one large KML file into smaller, more manageable files on the same computer.

So far, all of our examples have required that the KML code be delivered to Google Earth from the local machine. Network links give you the power to serve content from a remote location and are commonly used to distribute data to large numbers of users. In this way, if the data needs to be amended, it has to be changed only at the source location, and all users receive the updated data automatically.

### CGI Scripting for KML

In addition to pointing to files containing static data, a network link's <href> can point to data that is dynamically generated—for example, by a CGI script located on a network server. With some knowledge of a scripting language such as PHP, Python, or Perl, you can create a script that delivers a stream (or file) of KML data to each network link.

Two things are necessary for delivering KML through a network CGI:

When a call is made from the client (Google Earth) to the server, the server must (1) return a response code of HTTP 200 and (2) set the response's content type to `text/plain` or `application/vnd.google-earth.kml+xml`.

The response must be valid KML. For complex applications, proper error handling is very important.

**Tip:** A simple way to handle errors is to parse the server's error as the text for a folder name. For example, you could have the server return `<Folder><name>database inaccessible</name></Folder>` as a string. This is more informative (and more user-friendly) than letting the connection drop.

The following examples use Python, but they are equally valid in any other scripting language.

### Generating a Random Placemark

The following Python script generates random integer values for *latitude* and *longitude* and then inserts those values into the <coordinates> element of a <Point>. Whenever the network link is refreshed, the Python script runs again and

generates KML with new latitude and longitude values.

```
#!/usr/bin/python

import random

latitude = random.randrange(-90, 90)
longitude = random.randrange(-180, 180)
kml = (
   '<?xml version="1.0" encoding="UTF-8"?>\n'
   '<kml xmlns="http://www.opengis.net/kml/2.2">\n'
   '<Placemark>\n'
   '<name>Random Placemark</name>\n'
   '<Point>\n'
   '<coordinates>%d,%d</coordinates>\n'
   '</Point>\n'
   '</Placemark>\n'
   '</kml>'
   ) %(longitude, latitude)
print 'Content-Type: application/vnd.google-earth.kml+xml\n'
print kml
```

Here is an example of a KML file containing a Network Link that loads this Python script:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Folder>
    <name>Network Links</name>
    <visibility>0</visibility>
    <open>0</open>
    <description>Network link example 1</description>
    <NetworkLink>
      <name>Random Placemark</name>
      <visibility>0</visibility>
      <open>0</open>
      <description>A simple server-side script that generates a new random
        placemark on each call</description>
      <refreshVisibility>0</refreshVisibility>
      <flyToView>0</flyToView>
      <Link>
        <href>http://yourserver.com/cgi-bin/randomPlacemark.py</href>
      </Link>
    </NetworkLink>
  </Folder>
</kml>
```

### View-Based Refresh Queries

A standard network link is a uni-directional link: data flows only from the server to Google Earth. The view-based refresh enables bi-directional communication. When the view-based refresh is active, Google Earth returns the view coordinates to the server at a specified time. This may be every *n* seconds, minutes, or hours, or once a certain amount of time has elapsed since the view stopped moving. See <viewRefreshMode> in the KML 2.2 Reference.

The coordinates are returned to the server by means of an HTTP GET that appends the coordinates as follows (this is the default bounding box information):

```
GET /path/to/sever/script/query?BBOX=[longitude_west, latitude_south,
longitude_east, latitude_north] HTTP/1.1
```

If the request were made while the user was looking down on San Francisco, the coordinates might look as follows:

```
GET /path/to/server/script/query?BBOX=-122.497790,37.730385,-
122.380087,37.812331 HTTP/1.1
```

This feature can be used for some very creative applications, but to get you started, a simple example is presented below.

### Tracking a Point Directly Under Your View

The following server-side Python script parses the return message sent by Google Earth and responds with a Placemark at the center of the screen. Each time the Network Link is refreshed, a new Placemark is generated.

```
#!/usr/bin/python

import cgi

url = cgi.FieldStorage()
bbox = url['BBOX'].value
bbox = bbox.split(',')
west = float(bbox[0])
south = float(bbox[1])
east = float(bbox[2])
north = float(bbox[3])

center_lng = ((east - west) / 2) + west
center_lat = ((north - south) / 2) + south

kml = (
   '<?xml version="1.0" encoding="UTF-8"?>\n'
   '<kml xmlns="http://www.opengis.net/kml/2.2">\n'
   '<Placemark>\n'
   '<name>View-centered placemark</name>\n'
   '<Point>\n'
   '<coordinates>%.6f,%.6f</coordinates>\n'
   '</Point>\n'
   '</Placemark>\n'
   '</kml>'
   ) %(center_lng, center_lat)

print 'Content-Type: application/vnd.google-earth.kml+xml\n'
print kml
```

And here is the KML for the Network Link that loads the Python script:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Folder>
    <name>Network Links</name>
    <visibility>0</visibility>
    <open>0</open>
    <description>Network link example 2</description>
    <NetworkLink>
      <name>View Centered Placemark</name>
      <visibility>0</visibility>
      <open>0</open>
      <description>The view-based refresh allows the remote server to
calculate
         the center of your screen and return a placemark.</description>
      <refreshVisibility>0</refreshVisibility>
      <flyToView>0</flyToView>
      <Link>
        <href>http://yourserver.com/cgi-bin/viewCenteredPlacemark.py</href>
        <refreshInterval>2</refreshInterval>
        <viewRefreshMode>onStop</viewRefreshMode>
        <viewRefreshTime>1</viewRefreshTime>
      </Link>
    </NetworkLink>
  </Folder>
</kml>
```

The principle illustrated in this example can be used for some very complex applications. For example, if you have a

database of geographic information, you can extract the coordinates of the viewer, make a call to the database for the data specific to the view, and return it to Google Earth as KML.

## 3 KML MIME Types

When responding to a request from Google Earth (or any Earth browser), a KML server must follow a certain set of rules so that Google Earth can correctly interpret its responses.

Upon success, the server must return a response code of HTTP 200 and set the response's content-type to a suitable MIME type, as described here.

Google Earth reads KML and KMZ files. The MIME type for KML files is

- **`application/vnd.google-earth.kml+xml`**

The MIME type for KMZ files is

- **`application/vnd.google-earth.kmz`**

For Apache, add these lines to the *httpd.conf* file:

- `AddType application/vnd.google-earth.kml+xml .kml`
- `AddType application/vnd.google-earth.kmz .kmz`

See the Microsoft documentation for details on setting up MIME types on Microsoft's IIS.

The body of the response must contain valid KML data, including the XML declaration (`<?xml version="1.0" encoding="UTF-8"?>`). If the server returns invalid KML, the Network Link will stop, deactivate, and output an error message.

## What's Next?

Ready for more? Check out the *Developer's Guide*, which describes key KML features. Also, browse the *KML Reference* for information on specific elements.