How we calculate mileage.

The question often arises… "How does jet Delivery get its mileage? Google maps say's something different…" - This is a great point because the mileage Jet comes up with is often different then what actual mileage is driven by the driver. In fact often multiple routes can be taken to get between two points. Traffic, freeway closures, road conditions, can all play a role in the route a driver chooses. The point is; without taxi meters it would be difficult to always calculate the exact mileage.

Since April, 1998, when Jet Delivery stopped using its patented Milo-Graph System we have been calculating mileage using the "Haversine formula". That sounds complex, so let's just call it "Rate Miles" for simplicity. Rate Miles are calculated as follows:

1.) First we calculate "air miles" between the origin and destination zip codes. This is sometimes referred to "as the crow flies". Of course this mileage will always be less then what is actually required to drive between two points so we add a second step.
2.) To accommodate for turns, detours, no direct path, etc… We add a 1.40 conversion factor to the "air miles". (ex. 30 air miles = 42 "rate miles")

For the most part we've found the 1.40 conversion to be a very fair number for all parties involved. (That's not to say it's perfect by any means, especially when working around large bodies of water, or mountain ranges where in many cases large detours are necessary).

References:

Calculating air miles: http://en.wikipedia.org/wiki/Haversine_formula / http://en.wikipedia.org/wiki/Great-circle_distance
Zip-code latitude and longitude: http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html

Visual Basic – Source Code:

```
Function LatLonDistance(ByVal lat1, ByVal long1, ByVal lat2, ByVal long2)
 plug = 0.0174532922                                                          '#radians per degree (pi/180)
 i = sin(lat1*plug)*sin(lat2*plug) + (cos(lat1*plug)*cos(lat2*plug)*cos((long2*plug)-(long1*plug))) '# calculate
if i > -1 and i < 1 then                                                      '# make sure -1 < i < 1
 tlong = Arccos(i)                                                            '# take the inverse cosine
 miles = 69.093 * (tlong * 57.29577951)                                       '# convert radians back to degrees (180/pi)
else
 miles = 0                                                                    '<--- same coordinate
end if
 LatLonDistance = Round(miles)                                                '# round to ones place
End Function

Function Arccos(invar)
 Arccos = Atn(-invar / Sqr(-invar * invar + 1)) + 2 * Atn(1)                   '# get the inverse cosine
End Function
```