V&H Coordinate Conversion Object Library

Quentin Sager Consulting
20429 Ring Neck Road
Altoona, FL 32702
USA

www.quentinsagerconsulting.com
support@quentinsagerconsulting.com

The V&H Coordinate Conversion Library is an object oriented class library that provides methods to calculate the distances and convert between vertical and horizontal coordinates pairs and latitude and longitude coordinate pairs.

The library is currently available in the C++ programming language and is delivered in source code format only and includes the following files.

clConvert.h	Overall library definition to be included into user application
clGeoCoordinate.h	Geographic latitude and longitude object definition
clGeoCoordinate.cpp	Geographic latitude and longitude object implementation
clVHCoordinate.h	V&H coordinate object definition
clVHCoordinate.cpp	V&H coordinate object implementation
clVHConvert.mak	Sample Microsoft Visual C++ makefile
clVHConvert.dep	Make file dependency declarations

Library include files and data types

To use the V&H conversion objects you must first add or include the C++ header file *clConvert.h* to your application source code. This file contains common constants, data type definitions, and class forward declarations. It also contains the include directives to "pull in" the include files *clVHCoordinate.h* and *clGeoCoordinate.h* which contain the class definitions for the library objects.

The library uses a few typedef'd C structures to defined coordinate pair primitives to simplify the use of internal variables.

VHCoordinate declares the Vertical and Horizontal coordinate pair data type.

```
typedef struct tagVHCoordinate
   {
   double H;
   double V;
   }VHCoordinate;

typedef struct tagVHCoordinate *LPVHCoordinate;
typedef const struct tagVHCoordinate *LPCVHCoordinate;
```

LLCoordinate declares the Latitude and Longitude coordinate pair data type to store these values when using decimal degree format.

```
typedef struct tagLLCoordinate
   {
   double Lat;
   double Long;
   }LLCoordinate;

typedef struct tagLLCoordinate *LPLLCoordinate;
typedef const struct tagLLCoordinate *LPCLLCoordinate;
```

DMSSubCoordinate declares the subcomponents of a DMS formatted latitude and longitude coordinate pair data type.

```
typedef struct tagDMSSubCoordinate
   {
   double Degrees;
   double Minutes;
   double Seconds;
   }DMSSubCoordinate;
```

DMSCoordinate declares the Latitude and Longitude coordinate pair data type to store these values when using *Degree, Minutes, Seconds* format.

```
typedef struct tagDMSCoordinate
   {
   DMSSubCoordinate Latitude;
   DMSSubCoordinate Longitude;
   }DMSCoordinate;

typedef struct tagDMSCoordinate *LPDMSCoordinate;
typedef const struct tagDMSCoordinate *LPCDMSCoordinate;
```

The V&H Coordinate Object

The V&H Coordinate Object interface is defined in the include file *clVHCoordinate.h.* This include file is accessed through *clConvert.h*.

```
class clVHCoordinate
   {
   private:

      VHCoordinate m_VHCoordinate;
      LLCoordinate m_LLCoordinate;

      mutable char m_cVertical[100];
      mutable char m_cHorizontal[100];

   public:

      // helper methods

      static double distance(double v1, double h1, double v2, double h2);
      static double radians(double degrees);
      static double degrees(double radians);

      // object constructors

      clVHCoordinate();
      clVHCoordinate(double dVertical,double dHorizontal);
      clVHCoordinate(const clVHCoordinate& Source);
      clVHCoordinate(const clLLCoordinate& Source);

      virtual ~clVHCoordinate();

      // set or get current value

      double& Vertical();
      double& Horizontal();

      // get current value

      const char *VerticalAsString() const;
      const char *HorizontalAsString() const;

      // coordinate conversions

      const LPVHCoordinate toVH(double lat,double lon);
      const LPLLCoordinate toLatLon(double v0,double h0);
      const LPLLCoordinate toLatLon(int v0,int h0);

      // coordinate assignment

      clVHCoordinate& operator=(const clLLCoordinate& Source);
      clVHCoordinate& operator=(const clVHCoordinate& Source);

      // distance calculation

      double operator-(const clVHCoordinate& Source);
      double operator-(const clLLCoordinate& Source);
   };
```

double clVHCoordinate::distance(double v1,double h1,double v2,double h2)

Static method providing alternate means to calculate the airline mileage between two V&H coordinate pairs as specified in NECA FCC Tariff No. 4. The returned value may contain fractional mileage values. Generally these values will be rounded to the next highest whole value depending on a particular carrier's tariff.

Parameters:

v1
: First V&H coordinate pair vertical value.

h1
: First V&H coordinate pair horizontal value.

v2
: Second V&H coordinate pair vertical value.

h2
: Second V&H coordinate pair horizontal value.

Returns:

Airline mileage distance between the coordinate pairs as miles.

Examples:

```
#include "clConvert.h"

// because this method is static there is no requirement to formally
// declare a clVHCoordinate object variable ...

double miles = clVHCoordinate::distance(6363.235,2250.700,5568.001,1582.999);

// may have a fractional value, to round to next whole number value ..
   miles = ceil(miles);
```

double clVHCoordinate::radians (double degrees)

Static helper method to convert decimal degree format coordinate value to radians.

Parameters:

degrees
Decimal degree format latitude or longitude coordinate value.

Returns:

Input value converted to radians.

double clVHCoordinate::degrees (double radians)

Static helper method to convert from radians to decimal degree.

Parameters:

radians
Latitude or longitude coordinate value in radians.

Returns:

Input value converted to decimal degree.

Constructors

clVHCoordinate::clVHCoordinate()

Default object constructor.

clVHCoordinate::clVHCoordinate(double dVertical,double dHorizontal)

Create the object and specify the initial Vertical and Horizontal coordinate values.

clVHCoordinate::clVHCoordinate(const clVHCoordinate& Source)

Copy constructor. Create the object and initialize with the current values from a second V&H coordinate object.

clVHCoordinate::clVHCoordinate(const clLLCoordinate& Source)

Copy constructor. Create the V&H object and initialize it from a Latitude and Longitude coordinate object performing coordinate system conversion.

clVHCoordinate::~clVHCoordinate()

Virtual destructor. Provided to ensure correct object destruction sequence when the object is inherited.

double& clVHCoordinate::Vertical()
double& clVHCoordinate:: Horizontal()

Return by reference methods allowing the current Vertical or Horizontal coordinate value to be set or retrieved.

Returns:

Current value in get operation else value assignment.

Examples:

```
#include "clConvert.h"

   // instatiate the object
   clVHCoordinate cHV;

   // set the object's vertical and horizontal values
   cHV.Vertical()   = 6363.235;
   cHV.Horizontal() = 2250.700;

   // now simply retrieve the values we just set
   double v = cHV.Vertical();
   double h = cHV.Horizontal();

   // alternately we could instantiate the object like this
   clVHCoordinate cHV(6363.235,2250.700);

   // and then retrieve the values we just set
   double v = cHV.Vertical();
   double h = cHV.Horizontal();
```

const char *clVHCoordinate::VerticalAsString() const
const char *clVHCoordinate::HorizontalAsString() const

Return a pointer to the current Vertical or Horizontal coordinate value as an ASCII nul terminated string. The most common use for these methods is to display or print the current values.

Returns:

Pointer to nul terminated ASCII string.

const LPVHCoordinate clVHCoordinate::toVH(double lat,double lon)

Convert the specified latitude and longitude coordinate pair to Vertical and Horizontal coordinates and return a pointer to the newly assigned values. This is an alternative method to perform conversion. The same results could be achieved via the object's assignment operators.

Parameters:

lat

Latitude coordinate in decimal degree notation.

lon

Longitude coordinate in decimal degree notation.

Returns:

Constant pointer to the object's internal V&H coordinate pair structure.

Examples:

```
#include "clConvert.h"

   // instatiate the object
   clVHCoordinate cHV;
   double lat = 47.0004;
   double lon = -52.9669;

   // set new V&H values from latitude and longitude
   // and return pointer to storage record
   const LPVHCoordinate lpVh = cHV.toVH(lat,lon);

   // and now we can retrieve the new value directly
   double v = lpVh->V;
   double h = lpVh->H;
```

const LPLLCoordinate clVHCoordinate::toLatLon(double v0, double h0)
const LPLLCoordinate clVHCoordinate::toLatLon(int v0,int h0)

Convert the specified Vertical and Horizontal coordinate pair to latitude and longitude and return a pointer to the newly converted values. The overloaded interface to these methods allows transparent usage of whole and fractional values for the V&H coordinates.

Parameters:

v0

Vertical coordinate value.

h0

Horizontal coordinate value.

Returns:

Constant pointer to the object's internal latitude and longitude coordinate pair structure.

clVHCoordinate& clVHCoordinate::operator=(const clLLCoordinate& Source)

This assignment operator allows the object's vertical and horizontal coordinate values to be set from a latitude and longitude coordinate object. The clLLCoordinate object's values are first converted to V&H and then assigned to the clVHCoordinate object.

clVHCoordinate& clVHCoordinate::operator=(const clVHCoordinate& Source)

This assignment operator allows the object's vertical and horizontal coordinate values to be set directly from a second clVHCoordinate object.

double clVHCoordinate::operator-(const clVHCoordinate& Source)
double clVHCoordinate::operator-(const clLLCoordinate& Source)

The difference operator is used to calculate the Airline Mileage between two V&H coordinate objects. The Airline Mileage is calculated per NECA FCC Tariff No. 4 specifications. The resulting value may contain a fractional mileage value. Generally these values will be rounded to the next highest whole value depending on a particular carrier's tariff.

```
#include "clConvert.h"

   clVHCoordinate cHV_Src(6363.235,2250.700);      // create a V&H coordinate object
   clLLCoordinate cLL_Src(30.778438,-091.651669);  // create a latitude and longitude object
   clVHCoordinate cHV_Des = cLL_Src;               // create second V&H from the latitude and longitude object

   // to calculate the Airline Mileage between our V&H coordinate
   // objects we simply subtract one from the other ...
   double dHVdistance = (cHV_Src - cHV_Des);

   // we could also do this
   double dHVdistance = (cHV_Src - cLL_Src);
```

The Latitude and Longitude Coordinate Object

The Latitude and Longitude Coordinate Object interface is defined in the include file *clVHCoordinate.h.* This include file is accessed through *clGeoCoordinate.h.*

```
class clLLCoordinate
   {
   private:

      double m_dLatitude;
      double m_dLongitude;

      mutable char m_cLattitude[100];
      mutable char m_cLongitude[100];

      DMSCoordinate m_DMSCoordinate;

   public:

      // misc helpers

      static double radians(double dValue);
      static double degrees (double radians);
      static double DMStoDecimal(double dDegrees,double dMinutes,double dSeconds,int isLongitude);
      static void DecimaltoDMS(double dDecimal,DMSSubCoordinate& Source);

      // constructors

      clLLCoordinate();
      clLLCoordinate(double dLatitude,double dLongitude);
      clLLCoordinate(const clLLCoordinate& Source);
      clLLCoordinate(const DMSCoordinate& Source);
      clLLCoordinate(const clVHCoordinate& Source);

      virtual ~clLLCoordinate();

      // get set operations

      double& Latitude();
      double& Longitude();
      operator DMSCoordinate&();

      // get current value for display

      const char *LatitudeAsString() const;
      const char *LongitudeAsString() const;

      // assignment operations

      clLLCoordinate& operator=(const DMSCoordinate& Source);
      clLLCoordinate& operator=(const clVHCoordinate& Source);

      // distance calculations

      double operator-(const clLLCoordinate& Source);
   };
```

double clLLCoordinate::radians (double degrees)

Static helper method to convert decimal degree format coordinate value to radians.

Parameters:

degrees
Decimal degree format latitude or longitude coordinate value.

Returns:

Input value converted to radians.

double clLLCoordinate::degrees (double radians)

Static helper method to convert from radians to decimal degree.

Parameters:

radians
Latitude or longitude coordinate value in radians.

Returns:

Input value converted to decimal degree.

double clLLCoordinate::DMStoDecimal(double dDegrees, double dMinutes,double dSeconds,int isLongitude)

Static helper method to convert a latitude or longitude specified via explicit component values to a single decimal degree format.

Parameters:

dDegrees
Degree component of a latitude or longitude coordinate value.

dMinutes
Minutes component of a latitude or longitude coordinate value.

dSeconds
Seconds component of a latitude or longitude coordinate value.

isLongitude
0, the values specify a longitude, non zero, the value specifies a latitude.

Returns:

Coordinate value as a single decimal degree value.

void clLLCoordinate::DecimaltoDMS(double dDecimal, DMSSubCoordinate& Source)

Static helper method to convert a latitude or longitude coordinate specified in decimal degree format to its *Degree, Minutes, Seconds* equivalent.

Parameters:

dDecimal
Decimal degree value to convert.

Source
Reference to a DMSSubCoordinate data type to receive the converted values.

Constructors

clLLCoordinate::clLLCoordinate()

Default object constructor.

clLLCoordinate::clLLCoordinate(double dLatitude,double dLongitude)

Create the object and specify the initial Latitude and Longitude coordinate values.

clLLCoordinate::clLLCoordinate(const DMSCoordinate& Source)

Create the object and specify the initial Latitude and Longitude coordinate values from a *Degree, Minutes, Seconds* coordinate pair..

clLLCoordinate::clLLCoordinate(const clLLCoordinate& Source)

Copy constructor. Create the object and initialize with the current values from a second Latitude and Longitude coordinate object.

clLLCoordinate::clLLCoordinate(const clVHCoordinate& Source)

Copy constructor. Create the Latitude and Longitude object and initialize it from a V&H coordinate object performing coordinate system conversion.

clLLCoordinate::~clLLCoordinate()

Virtual destructor. Provided to ensure correct object destruction sequence when the object is inherited.

double& clLLCoordinate::Latitude()
double& clLLCoordinate::Longitude()

Return by reference methods allowing the current Latitude or Longitude coordinate value to be set or retrieved.

Returns:

Current value in get operation else value assignment.

const char *clLLCoordinate::LatitudeAsString() const
const char *clLLCoordinate::LongitudeAsString() const

Return a pointer to the current Latitude or Longitude coordinate value as an ASCII nul terminated string. The most common use for these methods is to display or print the current values.

Returns:

Pointer to nul terminated ASCII string.

clLLCoordinate::operator DMSCoordinate&()

Casting operator allowing the object assignment to a DMSCoordinate data structure.

clLLCoordinate& clLLCoordinate::operator=(const DMSCoordinate& Source)

This assignment operator allows the object's latitude and longitude coordinate values to be set from a *Degree, Minutes, Seconds* formatted coordinate pair data type.

clLLCoordinate& clLLCoordinate::operator=(const clVHCoordinate& Source)

This assignment operator allows the object's latitude and longitude coordinate values to be set from a V&H coordinate object by converting the Vertical and Horizontal coordinates to decimal degree latitude and longitude.

double clLLCoordinate::operator-(const clLLCoordinate& Source)

The difference operator is used to calculate the Great Circle Distance between two latitude and longitude coordinate objects.