VK5DJ_sunmoon.dll - how to use it

The version using extended variables is no longer supported as it was a problem keeping two versions up to date. To enable access by users of C compilers I have decided to limit the accuracy to using 'double' floats (8 byte). There is no effective difference, the limit to accuracy is the algorithm rather than the choice of variables. The DLL was compiled in Delphi.

The purpose of the DLL is to provide an easily used platform for the calculation of the position of the sun or the moon and to provide the associated information often required for amateur radio operators interested in EME. Due to the use of stdcall the DLL library should work with any of Windows versions of Delphi, Visual Basic, C and versions of these compilers designed for Linux. The first and most important procedure is that of DataToDLL. It is this procedure that sends the required data to the DLL to enable the main calculations to be made. Most of the functions described below extract result data from the DLL following this call. Some do a separate calculation such as bearing/distance, GetLocator, GetLatLong.

DataToDLL has 12 parameters. They are in order:

Year,Month,Day,Hour,Minute,Second: two byte word variables

HtASL, Latitude, Longitude: all of which are double real variables RefractSwitch,SunMoonSW: all of which are Boolean and
Band: which is a double variable).
To send the data to the DLL you load the appropriate variables with the values required and use:
DataToDLL(Year, Month, Day, Hour, Minute, Second, HtASL, Latitude,
Longitude, RefractSwitch,SunMoonSW,Band);
Typical values for the variables might be:
Year: 2008 (note full number required and all date time variables must be in UTC)
Month: 7 (values from 1-12)
Day: 1 (values from 1-31)
Hour: 13 (values from 0-23)
Minute: 10 (values from 0-59)
Second: 30 (values from 0-59
HtASL: 10 (values from 0.0 – whatever your height may be in metres)
Latitude:-37.5833 (your Latitude +ve for northern hemisphere, -ve for southern)
Longitude: 140.3831 (your longitude +ve for East and –ve for West)
RefractSwitch: true (values true or false. True includes optical refraction adjustment)
SunMoon:false (true= sun calculation, false=moon calculation)
Band: 1296.0 (Band in use in MHz used for Doppler calculations
Note that str8 is a type of string with 8 characters. It has been defined like this in GetLocator and GetRtAscHHMMSS Type Str8 = string[8];

Because the main procedure and its supporting functions are all external and contained in VK5DJ_sunmoon.dll, you need to tell your compiler and in the case of Delphi this would be in the declaration part of the interface or implementation section. Note the addition of the word **stdcall** and the name of the DLL in quotes. The DLL should be in the same directory as your main program.

The procedures and functions are first declared in your unit like this:

Procedure **DataToDLL**(Year,Month,Day,Hour,Minute,Second:word; HtASL, Latitude, Longitude:double; RefractSwitch,SunMoonSW:boolean;Band:word); StdCall; external 'VK5DJ_sunmoon.dll'

function GetElevation:double; stdcall; external 'VK5DJ_sunmoon.dll'

function GetAzimuth:double; stdcall; external 'VK5DJ_sunmoon.dll'

function GetDoppler:double; stdcall; external 'VK5DJ_sunmoon.dll'

Function GetLibration:double;stdcall;external 'VK5DJ_sunmoon.dll'

function GetDistance:double; stdcall; external 'VK5DJ_sunmoon.dll'

function GetAngularDiam:double; stdcall; external 'VK5DJ_sunmoon.dll'

function GetRightAsc:double; stdcall; external 'VK5DJ_sunmoon.dll'

function GetRtAscHHMMSS:str8; stdcall; external „VK5DJ_sunmoon.dll‟

function GetGHA:double; stdcall; external 'VK5DJ_sunmoon.dll';

function GetLHA:double; stdcall; external 'VK5DJ_sunmoon.dll';

function GetDeclination:double; stdcall; external 'VK5DJ_sunmoon.dll'

function GetDecTopo:double; stdcall; external 'VK5DJ_sunmoon.dll';

function GetDXdistance(Longitude,Latitude,RemoteLong, RemoteLat:double;Dist_B:boolean): double; stdcall; external 'VK5DJ_sunmoon.dll'

function GetPolarity:double; stdcall; external 'VK5DJ_sunmoon.dll'

function GetDB:double; stdcall; external 'VK5DJ_sunmoon.dll'

function GetLocator(Latitude,Longitude:double):str8; stdcall; external 'VK5DJ_sunmoon.dll'

function GetLatLong(Locator:Str8; LatOrLong:boolean):double; stdcall;
external 'VK5DJ_sunmoon.dll'

function GetVersion:double; stdcall; external 'VK5DJ_sunmoon.dll';

function GetMoonIllum:double; stdcall; external 'VK5DJ_sunmoon.dll';

function GetMoonPhase:double; stdcall; external 'VK5DJ_sunmoon.dll';

function GetQuarter:str8; stdcall; external 'VK5DJ_sunmoon.dll';

function GetPhaseAngle:double; stdcall; external 'VK5DJ_sunmoon.dll';

Notes
In this document I have split the listings of the main procedure DataToDLL, function GetDXDistance, and GetLatLong across several lines for legibility reasons. Depending on your compiler this may be allowable or each may need to be placed as one long line. Be careful that you include the required punctuation.

In GetLocator, GetRtAscHHMMSS and GetQuarter there is a type of "Str8" used. This is typed as Str8=String[8];

When the main procedure DataToDLL is called the sun or moon calculations are performed and data is created and stored in the DLL library.

To extract the data you call the appropriate functions:
function **GetElevation**:double; returns the elevation of the object from your home location in degrees in a double variable. When printing this to screen you should round it to no better than the nearest 0.01 degree.

function **GetAzimuth**:double; returns the azimuth of the object from your home location in degrees in a double variable. When printing this to screen you should round it to no better than the nearest 0.01 degree.

function **GetDoppler**:double; returns the Doppler shift home/home if DataToDLL call was loaded with home data, and it returns the Doppler shift DX/DX if the DataToDLL call was loaded with the remote station data. Each shift is doubled from the perspective of the station concerned as listening to his/her own signal.

To calculate the mutual Doppler shift the single direction shifts are taken into account, added and averaged. See my sample program TestDLL. Note that to obtain the required data for both the home station and the DX station the procedure DataToDLL must be called with the appropriate information before reading back the Doppler shift. Each read provides the doubled Doppler shift as if the station was hearing its own echoes. The Doppler shift between Home and DX is then calculated using something like this:
DopplerHome_DX:= DopplerHome/2 +DopplerRemote/2

Internally Doppler uses a time period 1 minute earlier, the current time and the time forward 1 minute to calculate the difference in distance between the moon and the station. This distance difference is turned into a Doppler shift using the speed of light and the frequency in use.

The **GetDoupler**:double**;** returns the value of Doppler into the double variable Doppler.

function **GetLibration**:double; returns the libration expressed as spread in Hertz of a received frequency

function **GetDistance**:double; returns the distance to the sun/moon in kilometres.

function **GetAngularDiam**:double; returns the angular diameter of the sun/moon in degrees.

function **GetRightAsc**:double; returns the RightAscension of the sun/moon in decimal hours.

function **GetRightAscHHMMSS**:Str8; returns the RightAscension of the sun/moon as a short string of 8 characters i.e. string[8] in the form of HH:MM:SS. Note the type of the function MUST be a short string of 8 characters. See variable types above.

function **GetDeclination**:double; returns the geocentric declination of the sun/moon in degrees.

function **GetDecTopo**:double; returns the topocentric declination of the sun/moon in degrees.

function **GetGHA**:double; stdcall; returns the Greenwich Hour Angle for the object in degrees.

function **GetLHA**:double; stdcall; returns the Local Hour Angle in degrees

function **GetDXdistance**(Longitude,Latitude,RemoteLong,RemoteLat,True): double;
returns the distance of the remote station in kilometres when "True" or its great circle bearing if "False". Typical values for the variables might be:
Longitude: 140.8 (dec degrees longitude +ve for East and –ve for West)
Latitude: -37.8 (dec degrees Latitude +ve for Northern hemisphere, -ve for southern)
RemoteLong: -122.00 (DX station longitude +ve for East and –ve for West)
RemoteLatitude: 24.982 (DX station Latitude, +ve for N, -ve for S)
Dist_B: true (true = distance calculation in Km, false= bearing calculated in degrees)

function **GetPolarity**:double; returns the polarity shift as measured from a moon's perspective. See my example as to how to combine the readings from home station and DX station to identify the possible rotation required between the two antennas (Home and DX). The polarity is most easily read as a displacement + or – from the DX alignment so I have used an equation to extract it once each polarity is determined:
PolarityShift:=PolarityHome-PolarityRemote;
If PolarityShift>90 then PolarityShift:=PolarityShift-180;
The resultant is the direction the polarization of the home station must be rotated to match another station using the same polarization as you. Keep in mind that below 1296 MHz Faraday rotation can be such as to make this figure meaningless. Treat it as a guide.

function **GetDB**:double; returns the relative signal strength in dB compared with the mean position of the moon. The strength will be greater at Perigee and weaker at Apogee. This figure is an approximate guide as there are many other factors that affect signals. It takes into account the distance of the moon and the size of the moon.

The next two functions are routines that may prove useful in your program.
function **GetLocator**(Latitude,Longitude:double):string[8]; returns the Locator for the station when the latitude and longitude is passed to the function.

function **GetLatLong**(Locator:Str8; LatOrLong:boolean):double; returns a latitude and longitude in decimal degrees from a provided locator. The routine is called with the Locator and a Boolean variable LatOrLong as parameters. Locator has six characters eg QF02ek and the LatOrLong variable is set true to recover the Latitude and false to recover the Longitude. Because of the nature of Locators the Lat and Long are approximate but sufficiently accurate for calculations provided by the DLL.
Example:
Latitude:=GetLatLong('QF02ek', true); sets Latitude to -37.5625 while
Longitude:=GetLatLong('QF02ek',false); sets Longitude to 140.375

function **GetVersion**:double; returns the version number of the DLL as a double number. A typical external call might look like this:
label27.Caption:='DLL version '+floattostr(GetVersion);

function **GetMoonIllum**:double; returns the illumination of the moon in a number from 0-1.
Label29.caption:='Moon phase '+floattostr(GetMoonIllum);

function **GetMoonPhase**:double; returns the phase of the moon in a number from 0-1.
Label28.caption:='Moon phase '+floattostr(GetMoonPhase);
0 to 0.49 is a waxing moon, 0.5 is full moon while 0.51 to 1.00 is a waning moon
The function takes two seconds to determine if the moon is waxing (growing) or waning (decreasing) in illumination, consequently the function must be in a time loop. The routine takes two seconds to settle. Phase and illumination are similar but phase takes into account the waxing and waning of the moon.

Function **GetQuarter**: Str8; returns the quarter of the moon (viz 1^{st}, 2^{nd}, 3^{rd}, 4^{th})

Function **GetPhaseAngle**:double returns an angle between 0 and 180 being defined as the phase angle formed at the moon by two lines, one from the sun to the moon and one from earth to moon. An angle of 0 represents a full moon, whereas an angle of 180 represents the new/old moon point when we would see a full exclipse of the sun. Some sources adjust this angle to represent illumination with 0 as new moon,180 as full moon and 360 as the old moon. Astronomical texts use the first interpretation and so this is used here.

Summary
I have included a sample program written in Delphi to illustrate how the VK5DJ_sunmoon.dll may be used. Note that when printing the figures to a screen you would normally not show the double variable value as I have mostly done. This was done only to minimize the example code for you and keep it uncluttered by print statements. I have provided a couple of rounded printouts to show how it might be done.

I recommend the following printout accuracy.
Azimuth and **Elevation**: 2 decimal places in degrees
RightAscension and **Declination**: normally 2 decimal places but I have included a function within the DLL to output hours:mins:secs for Right Ascension.
Distance to moon: nearest km
Angular diameter of moon: 3 decimal places in degrees
Distance of DX in km round to nearest 10km (short distances to 1 km but large distances due to irregularities of earth limit accuracy to 0.1%)
Bearing of DX in degrees round to 1 decimal place (irregularities limit to 0.1%)
Doppler readings to nearest Hz
Libration to nearest Hertz
Antenna polarity to nearest degree (if you are an optimist)
GetDB could be printed to a rounded 1 or 2 decimal places. Again, this is a rough guide only.
GetLocator returns string 6 characters long and should be printed in its entirety.
GetLatLong returns a double variable in decimal degrees and would normally be converted to Deg:Min:Sec. You will need to program the conversion; it is not available from the DLL.
Moon Phase and **Illumination** to two decimal points.
Phase angle rounded to three places of integer.
Moon **quarter** as a string
GHA and **LHA** to two decimal places

Enjoy these routines. No guarantees are offered, use at your own peril. The results seem to be pretty good. The improved algorithms for Doppler in this version are from a program by Gerald (K5GW). They are more accurate than my effort. Thanks Gerald. I acknowledge libration coding by Keith Burnett and George Rosenberg.

If you use this DLL library I would hope that you will keep its name intact and acknowledge the author's work in bringing together the knowledge of many people.

Acknowledgements:

Author	**Title**	**Published**
Jean Meeus	Astronomical Algorithms 2nd Edition 1998 with corrections 2005	Willmann-Bell, Inc
Peter Duffett-Smith	Astronomy with your personal computer	Cambridge University Press 1985
John Morris	Amateur Radio Software	RSGB 1985
Gerald Williamson	MC011011.bas re doppler	
Keith Burnett and others	Spreadsheet re libration	

John Drew
VK5DJ

DLL version 2.15 March 2013
Updated 6 September 2014: Corrected topocentric distance and distinguished illumination and phase. Added quarter and Phase Angle.