OGI DRIVER FUNCTIONS FOR LABVIEW USERS
Surrey University

NOTES: All functions are declared as stdcall. Integer values are all 32-bit, and Double-precision values are 8-byte (64-bit).
Many functions have an "Enable" argument. In all such cases,
ENABLE = 0(for disable/normal), or 1(for Enable/Trigger).
Please NOTE that setting Enable to 1 will trigger a request for data from the OGI, but the data will not be updated until the reply is received. This is normally very fast, but not instantaneous, so the results returned at the time of triggering will still be the *previous* values.

Initialisation and Finalisation functions

(Not required if using the OGI purely as a DLL, with no OGI_If)

function PrepLink; stdcall;
Returns: void
Arg1 (IPAddress) = C-String Pointer
NOTES: When running the OGI and the Interface DLL on the same computer, use the address: 127.0.0.1
This function MUST be called before using any other of the interface functions.

function CloseLink; stdcall;
Returns: void
Takes no arguments.
This function must be used to cleanly close down the system after use.

function OGIConnected; stdcall;
Returns: 32-bit integer. 0=Disconnected, 1=Connected
Takes no arguments.
A return value of 1 indicates that a socket connection to the OGI is currently open.

Live data functions

function GetAxisLoads; stdcall;
lift, pitch, drag, side, yaw, roll calibrated loads, in the selected units.
Returns: void
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (lift Reading) as POINTER to a double-precision.
Arg3 (pitch Reading) as POINTER to a double-precision.
Arg4 (drag Reading) as POINTER to a double-precision.
Arg5 (side Reading) as POINTER to a double-precision.
Arg6 (yaw Reading) as POINTER to a double-precision.
Arg7 (roll Reading) as POINTER to a double-precision.
NOTES: Set ENABLE=1 to trigger a reading. The returned values give the latest received axis readings. If called within a loop, these will be automatically updated when the response is received.

function GetAxisPercs; stdcall;
lift, pitch, drag, side, yaw, roll load gauge values (+/- 100%)
Returns: void
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (lift Load) as POINTER to a double-precision.
Arg3 (pitch Load) as POINTER to a double-precision.
Arg4 drag Load) as POINTER to a double-precision.
Arg5 (side Load) as POINTER to a double-precision.
Arg6 (yaw Load) as POINTER to a double-precision.
Arg7 (roll Load) as POINTER to a double-precision.
NOTES: Set ENABLE=1 to trigger a reading. The returned values give the latest received loads as percentages of Full Scale. If called within a loop, these will be automatically updated when the response is received.

function SetZero; stdcall;
Takes the current Absolute readings to become the new Tare Loads, thus setting a new Zero condition. Also selects Tared (Net) readings to be returned by the GetAxisLoads function.
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
NOTE: Set ENABLE=1 to send the command.

- *select Absolute readings*

  function SelectAbsolute; stdcall;
  Select Absolute values (rather than Tared values) to be returned by the GetAxisLoads function.
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  NOTE: Set ENABLE=1 to send the command.

- *select Net readings*

  function SelectNet; stdcall;
  Select Net (i.e. Tared) values to be returned by the GetAxisLoads function. Readings will now be relative to the latest Zero condition.
  Returns: 32-bit integer.
  (0=acknowledged, 1=awaiting acknowledgment)
  (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  NOTE: Set ENABLE=1 to send the command.

  function SetPitotZero; stdcall;
  Sets the Windspeed Pitot to read Zero at the present conditions, provided that the current absolute reading does not exceed +/- 5 Pascals.
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  NOTE: Set ENABLE=1 to send the command.

Resolution Centre functions (Vertical & Horizontal offsets)

function GetZOffset; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2: (Current Vertical “Z” offset, from Balance centre to required centre, in mm) as a POINTER to a double-precision value. Up = +ve.
NOTE: Set ENABLE=1 to send the command.

function SetZOffset; stdcall;

Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (New Vertical “Z” Offset in mm) as double-precision, by value.
Up = +ve.
NOTE: Set ENABLE=1 to send the command.

function GetXOffset; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2: (Current Horizontal “X” offset, from Balance centre to required centre, in mm) as a POINTER to a double-precision value. Forward = +ve.
NOTE: Set ENABLE=1 to send the command.

function SetXOffset; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (New Horizontal “X” Offset in mm) as double-precision, by value.
Forward = +ve.
NOTE: Set ENABLE=1 to send the command.

High Speed logging

- *HS start and stop logging*

function StartHSLog; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
NOTE: Set ENABLE=1 to trigger start of HS logging.
The data will be recorded in a file named according to the date and time when logging was started, thus: HSLog_yymmdd_hhmmss.csv

function StopHSLog; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
NOTE: Set ENABLE=1 to terminate HS logging.

- *Set Limit for number of data points to be acquired.*

  function SetHSPointsLimit; stdcall;
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  Arg2 (New Limit, i.e. maximum points to acquire) = 32-bit integer, by Value.
  NOTE: Set ENABLE=1 to send the command.

- *Activate or de-activate HS Points Limit*

  function SetHSPointsLimitStatus; stdcall;
  Returns: void
  Arg1 (Enable) = 32-bit integer, by Value.
  Arg2 (Limit Active=1, Unlimited=0) = 32-bit integer, by Value.
  NOTE: Set ENABLE=1 to send the command.

- *Get Number of HS data points acquired since call to StartHSLog*

  function GetHSDataPoints; stdcall;
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  Arg2 (Points acquired) as Pointer to 32-bit integer.
  NOTE: Set ENABLE=1 to send the command.

Digital sigconsetup

function **USERSigConSetup**; stdcall;
Returns: 32-bit integer.
(0=acknowledged, 1=awaiting acknowledgment)
Arg1 (Enable) = 32-bit integer, by Value.
NOTE: When Enable is set to 1, a window is displayed with the existing SigCon setup values shown. Any or all values can be modified, and if the user clicks OK, then the Setup will be updated.
IMPORTANT: If used remotely, then The GetSigConSetup function must have been called before this function is Enabled, to ensure that the correct existing Tuning values will be shown. This function is not provided for local use.

Units

function GetUnits; stdcall;
Returns: Configuration name as a Pointer to C-String. String contains ForceUnits.DistanceUnits (i.e. two units names, separated by a full stop).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (ForceUnitsCode) as POINTER to a 32-bit integer.
Arg3 (DistanceUnitsCode) as POINTER to a 32-bit integer.
NOTES: Set ENABLE=1 to trigger a reading. The returned values give the latest received units. If called within a loop, this will be automatically updated when the response is received.

ForceUnitsCode numbers:
0 = Unknown
1 = Newtons (N)
2 = kilograms (kg)
3 = Lbf
4 = Poundals

DistanceUnitsCode numbers:
0 = Unknown
1 = metres (m)
2 = feet (ft)
3 = inches (in)

function SetUnits; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (NewForceUnitsNumber) = 32-bit integer, by value
Arg3 (NewDistanceUnitsNumber) = 32-bit integer, by value
NOTE: Set ENABLE=1 to send the command.
Units codes are as for the GetUnits function.

Wind and Fan system

- *Fan Controls*

  function SetFanSystemState; stdcall;
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  Arg2 (Fan ON/OFF State) = 32-bit integer, by Value.
  NOTE: Set ENABLE=1 to send the command.
  Fan ON/OFF State = 0 to Disable the Fan System operation and remove power from the fan. Fan ON/OFF State = 1. to Enable the Fan system and accept fan-control commands.
  When Disabled, the Fan system will not even be able to execute a controlled rapid stop (FanStop function), but if already rotating will free-wheel to a stop. It is however recommended that the system should only be set to the off state when already static.

  function FanStop; stdcall;
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  NOTE: Set ENABLE=1 to trigger FanStop.
  This command will set the fan into Manual mode, and will cause it to stop regardless of any previously requested WindSpeed.

- *Fan speed RPM / %*
  function GetFanSpeed; stdcall;
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  Arg2 (=Fan Speed in RPM) as POINTER to a double-precision.
  NOTES: Set ENABLE=1 to trigger a reading. The returned values give the latest received Speed. If called within a loop, this will be automatically updated when the response is received.

function SetFanspeed; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (Required Fan Speed, as RPM) as double-precision by Value.
NOTE: Set ENABLE=1 to send the command.
This command will set the fan into Manual mode, operating at the fixed speed given by Arg2.

- *Wind speed (m/s)*

function GetWindSpeed; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (=Wind Speed in m/s) as POINTER to a double-precision.
NOTES: Set ENABLE=1 to trigger a reading. The returned value gives the latest computed Wind Speed measurement. If called within a loop, this will be automatically updated when the response is received.

function SetWindSpeed; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (Required WindSpeed m/s) as double-precision, by value.
NOTE: Set ENABLE=1 to send the command.
This command will set the fan into Automatic mode, operating as determined by the WindSpeed control block.

- *Barometric pressure*

function GetCurrentBaroPr_mB; stdcall;
Returns: Barometric pressure (in mbar) as a double-precision value.
Arg1 (Enable) = 32-bit integer, by Value.
NOTES: Set ENABLE=1 to trigger a reading. The returned value gives the latest pressure reading from the user, or from a sensor if fitted. If called within a loop, this will be automatically updated when the response is received.

function SetNewbaroPr_mB; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (New pressure) as double-precision, by value.
NOTE: Set ENABLE=1 to send the command.
This function is provided for use in cases where no Barometric Pressure sensor is installed, and allows a pressure reading from external sources to be entered into the system.

- *Tunnel temperature (deg)*

function GetCurrentTemp_C; stdcall;
Returns: Tunnel temperature (deg C) as a double-precision value.
Arg1 (Enable) = 32-bit integer, by Value.
NOTES: Set ENABLE=1 to trigger a reading. The returned value gives the latest measured Temperature of air in the tunnel. If called within a loop, this will be automatically updated when the response is received.

Road control

- *Road system condition reports*

  function GetRoadStatus; stdcall;
  Returns: Road Status values as a Pointer to C-String. String contains True/False values as T or F, separated by semicolons.
  Arg1 (Enable) = 32-bit integer, by Value.
  NOTES: Set ENABLE=1 to trigger a reading. The returned values give the latest received status. If called within a loop, this will be automatically updated when the response is received.

  Status values:
  E-Stop Active
  Main drives Started
  Belt slip warning
  Belt absent or broken
  Suction established
  Load error (LH Loadcell)
  Load error (RH Loadcell)
  Water temperature alarm active
  Platen temperature alarm active

  function GetRoadInfo; stdcall;
  Returns: Road Information values as a Pointer to C-String. String contains values in ASCII form, separated by semicolons. Values are of type Real (i.e. including a decimal point), except where indicated.
  Arg1 (Enable) = 32-bit integer, by Value.
  NOTES: Set ENABLE=1 to trigger a reading. The returned values give the latest received status. If called within a loop, this will be automatically updated when the response is received.

  Info values:
  RoadINFO; (Literal Id string)
  Belt Travel To Date (km);
  Platen Temperature (FrontLeft);
  Platen Temperature (FrontRight);
  Platen Temperature (RearLeft);
  Platen Temperature (RearRight);
  Water Outlet Temperature (Front);

Water Inlet Temperature (Rear);
Zone1 Suction (Front); (inches water column/gauge)
Zone2 Suction (Mid); (inches water column/gauge)
Zone3 Suction(Rear); (inches water column/gauge)
Primary BoundaryLayer percent peed; (Integer)
Secondary BoundaryLayer percent Speed; (Integer)
LoadCell_kg (Left);
LoadCell_kg (Right);
WTH=x Water Temperature High Alarm, x=T(true) or F(false); (str)
PTH=x Platen Temperature High Alarm, x=T(true) or F(false); (str)

- *Road Drive operation*

function StartRoadDrive; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
NOTE: Set ENABLE=1 to send the command.
Set the main drive for the Road system into its operational state. Note: The road speed must be effectively zero, and the drive not already operational for this command to be accepted.

function StopRoadDrive; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
NOTE: Set ENABLE=1 to send the command.
Set the main drive for the Road system into its stopped (non-operational) state. Note: The drive will be commanded to zero speed, and will then be set to its Off (stopped) condition.

- *Road speed (m/s)*

function GetRoadSpeed_mps; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (RoadSpeed in m/s) as POINTER to double-precision.
NOTES: Set ENABLE=1 to trigger a reading. The returned value in Arg2 gives the latest received Speed. If called within a loop, this will be automatically updated when the response is received.

function SetRoadSpeed_mps; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (Required RoadSpeed m/s) as double-precision, by value.
NOTE: Set ENABLE=1 to send the command. If the drive is not in the "Started" condition then this command has no effect.

function RoadSyncToWind; stdcall;
Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (New Road-Sync-To-Wind-State) = 32-bit integer, by Value.
NOTE: Set ENABLE=1 to send the command.

if Road-Sync-To-Wind-State is 0, then The Road system will only run under road commands, and wind speed will be ignored by the Road system. If the Road-Sync-To-Wind-State is 1 then the road speed will be synchronised with the wind speed.

- *Controlled (slow) stop Road*

function RoadStop; stdcall;
Returns: 32-bit integer.
0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
NOTE: Set ENABLE=1 to trigger RoadStop. This commands the belt speed to drop to zero.

- *Boundary Fans*

function SetBoundaryPerc; stdcall;
Returns: 32-bit integer.
0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (Fan) = 32-bit integer, by Value. 1=Primary, 2 = Secondary.
Arg3 (Percent) = 32-bit integer, by Value.

NOTE: Set ENABLE=1 to send the selected fan to the percent setting.

Yaw control

- *Yaw speed (degrees/ second*)

  function SetYawSpeed_dps; stdcall;
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  Arg2 (NewSpeed, in deg/sec) as double-precision, by value.
  NOTE: Set ENABLE=1 to send the command.

  function GetYawSpeedSetting_dps; stdcall;
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  Arg2 (Current Speed setting in degrees/ second) as POINTER to a double-precision.
  NOTES: Set ENABLE=1 to trigger a reading. The returned value in Arg2 gives the latest received speed setting. If called within a loop, this will be automatically updated when the response is received.

- *Yaw position (Degrees)*

  function GetYawPosition; stdcall;
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  Arg2 (YawPos in degrees) as POINTER to double-precision.
  NOTES: Set ENABLE=1 to trigger a reading. The returned value in Arg2 gives the latest received Position. If called within a loop, this will be automatically updated when the response is received.

  function GotoYawPosition; stdcall;
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.

Arg2 (Required YawPos, in degrees) as double-precision, by value.
NOTE: Set ENABLE=1 to send the command.

- *Yaw Stop*

  function YawStop; stdcall;
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  NOTE: Set ENABLE=1 to trigger YawStop.

- *Conditions / Status information*

- *Yaw, Fan, Road ready*
- *Yaw, Fan, Road running*
- *Yaw, Fan, Road limits active*
- *Yaw, Fan, Road Fault*
- *Yaw, Fan, Road move complete*
- *Yaw, Fan, Road connected*

function GetConditionsInfo; stdcall;
Returns: void
Arg1 (Enable) = 32-bit integer, by Value.
Arg2 (YawControl) as POINTER to 32-bit integer (see details below)
Arg3 (FanControl) as POINTER to 32-bit integer (see details below)
Arg4 (RoadControl) as POINTER to 32-bit integer (see details below)
Arg5 (GLOBAL conditions) as POINTER to 32-bit integer (see details below)
{ Conditions: Yaw, Fan, Road, Globals }
{ AXES: }
{ Bit 0 - Ready }
{ Bit 1 - Running }
{ Bit 2 - Killed }
{ Bit 3 – Following Error }
{ Bit 4 – H/W Limit }
{ Bit 5 - Fault }
{ Bit 6 - unused }
{ Bit 7 - unused }
{ Bit 8 – Move Complete }
{ Bit 9 - unused }
{}
{ GLOBALS: }
{ Bit 0 - unused }
{ Bit 1 - E-STOP active }
{ Bit 2 - unused }
{ Bit 3 - unused }

- *E stop ALL (E-stop)(Boolean)*

  function EStop; stdcall;
  Returns: 32-bit integer. 0=acknowledged, 1=awaiting acknowledgment. (-1 = unable to send command).
  Arg1 (Enable) = 32-bit integer, by Value.
  NOTE: Set ENABLE=1 to trigger EStop.

Auxiliary functions

function GetSmADData; stdcall;
Returns: 16 SmAD (Smart A/D) channels and one flag, as a Pointer to C-String. String contains Message number and “SmADData” Keyword, separated by a space and followed by a semicolon; and then the data values. Data channel values are separated by semicolons. The first 8 (channels 0 to 7) are voltage readings, the remainder (channels 8 to 15) are temperatures, using type T thermocouples. The last data value is followed by a semicolon and either the character “A” or “I”, indicating that the SmAD communications are either Active or Inactive. If the Inactive state occurs during normal operation, this may indicate a loss of communications.

Arg1 (Enable) = 32-bit integer, by Value.
NOTES: Set ENABLE=1 to trigger a reading. The returned values give the latest received values. If called within a loop, this will be automatically updated when the response is received.

08/06/2011