# Name:
# G4Track Library – G4Track.dll – Windows
# libG4Track.so - Linux

**Description:**

The G4Track library is the main host interface to the Polhemus G4Tracker. Implemented as a Dynamic Link Library (Windows) or a Shared Library (Linux), it is a set of interfaces (functions, macros, structures) that provide a low-level interface to user host programs. All of the G4 features and data are available to be set or retrieved through these interfaces. This library will provide the ability for programmers to develop applications that communicate with the G4 tracker. All Linux programmers, and Windows programmers who wish to write portable code, must use this library. For Windows-only programmers a more user-friendly Windows G4 SDK is available. This SDK provides a larger API that wraps some of the complexity of this library.

**Function Summary:**

The library has four function calls that it uses to interface with the G4 Trackers. Please see the documentation for each function for more information.

**g4_init_sys** – This function is called to initialize the library and send the proper source information to each of the G4 trackers.

**g4_set_query** – This is the main configuration function. It has many different combinations of parameters and is used to configure the output, as well change filtering, boresighting, frame-of-reference and more.

**g4_get_frame_data** – This is the main data retrieval function of the library. It must be called to extract the position and orientation data of each sensor on each hub.

**g4_close_tracker** – This function should be called when the application quits; closing all the communication channels and cleanly shutting down all the library threads.

The typical scenario would be for an application to call **g4_init_sys()** on startup, then call **g4_set_query()** for each configuration parameter that required setting (or getting). This may require a number of calls. After initialization and configuration the application would call **g4_get_frame_data()** as often as it required tracking data. Calls to g4_set_query can also be made again at any time. Finally when the application is ready to quit, it should call **g4_close_tracker()**.

# Structures

Following are the structures used by the host application to interface with the G4Track Library:

### G4_SENFRAMEDATA Structure

```
typedef struct _G4_SEN_FRAMEDATA {
        uint32_t id;
        float pos[3];
        float ori[4];
}*LPG4_SENFRAMEDATA,G4_SENFRAMEDATA;
```

This structure is used to receive the position and orientation data from each sensor. An array of G4_SENFRAMEDATA structures are embedded in the G4_FRAMEDATA structure to return the data for each sensor on the hub.

Members:

*id*: The zero-based sensor number.

*pos*: An array of three floats to hold the x,y, and z data respectively. The units for these values will be that set by the g4_set_query command. Default is inches.

*ori*: An array of three or four floats to hold the orientation data. This may be three values if the the library has been configured to return Euler Degrees, or Euler Radians. These values will be azimuth (yaw), elevation (pitch), and roll respectively. This array will hold four valid values if the library has been configured to return quaternions. The default is Euler Degrees.

**G4_FRAMEDATA**

```
typedef struct _G4_FRAMEDATA {
        uint32_t hub;
        uint32_t frame;
        uint32_t stationMap;
        uint32_t dig_io;
        G4_SENFRAMEDATA sfd[G4_SENSORS_PER_HUB];
}*LPG4_FRAMEDATA,G4_FRAMEDATA;
```

This structure is used in the *g4_get_frame_data* function to retrieve the most recent frame's tracker data. It contains an array of G4_SENFRAMEDATA structures, one for each sensor on the hub. One G4_SENFRAMEDATA structure is created to retrieve the information for one hub and its sensors. An application must create an array of G4_SENFRAMEDATA structures, one for each hub for which it is requesting data, and pass this array into the *g4_get_frame_data* function.

Members:

*hub*: This is the hub id as reported by the system.

*frame*: This is the frame number to which the position and orientation data corresponds.

*stationMap*: This is a bit-map to indicate which sensors are active on the hub. Sensor 0 corresponds to bit 0, sensor 1 to bit 1, and sensor 2 to bit 2. If the bit is set to 1 then the sensor is present and active, otherwise the bit will be set to 0.

*dig_io*: This is an eight-bit bit map that corresponds to the digital I/O ports on the hub. These ports are for user-specific applications.

*sfd*: An array of G4_SENFRAMEDATA that will hold the position and orientation data for each sensor on the hub. See description of G4_ SENFRAMEDATA.

**G4_CMD_DATA_STRUCT**

This is the structure that identifies the parameters for the individual configuration commands. This structure is embedded in the G4_CMD_STRUCT that gets passed to the *g4_set_query* function to read or write configuration values from or to the tracker library. Typically the id will be created by the G4_CREATE_ID macro, the action will indicate reading or writing, and the iParam and pParam members are command specific.

Members:

*id*: The value created by the G4_CREATE_ID macro. It is a combination of the system number, hub number, and sensor number.

*action*: Must be one of the enums G4_ACTION_SET, G4_ACTION_GET, or G4_ACTION_RESET.

*iParam*: Command specific 32-bit parameter. See specific command documentation.

*pParam*: Command specific pointer parameter. See specific command documentation.

**G4_CMD_STRUCT**

```
typedef struct _G4_CMD_STRUCT {
        int cmd;
        G4_CMD_DATA_STRUCT cds;
}*LPG4_CMD_STRUCT,G4_CMD_STRUCT;
```

This is the structure that is passed to the g4_set_query function. It is used to set or read all configuration data in the tracker. A G4_CMD_DATA_STRUCT structure is embedded in this structure that contains the parameters and variables necessary to write or read the specific configuration that corresponds to the specific command. The values in the G4_CMD_DATA_STRUCT are command specific.

Members:

*cmd*: One of the following enums:

- G4_CMD_WHOAMI,
- G4_CMD_GETMAXSRC,
- G4_CMD_BORESIGHT,
- G4_CMD_FILTER,
- G4_CMD_INCREMENT,
- G4_CMD_FOR_ROTATE,
- G4_CMD_FOR_TRANSLATE,
- G4_CMD_TIP_OFFSET,
- G4_CMD_UNITS,
- G4_CMD_GET_ACTIVE_HUBS,
- G4_CMD_GET_STATION_MAP,
- G4_CMD_GET_SOURCE_MAP,
- G4_CMD_FRAMERATE,
- G4_CMD_RESTORE_DEF_CFG,
- G4_CMD_BLOCK_CFG,
- G4_TOTAL_COMMANDS

*cds*: A G4_CMD_DATA_STRUCT that contains the necessary information or variables that correspond to the specific command in *cmd*. See the documentation for each command.

**G4_SRC_MAP**

```
typedef struct _G4_SRC_MAP {
        int id;
        int freq;
        float pos[3];
        float att[4];
}*LPG4_SRC_MAP,G4_SRC_MAP;
```

This structure is used to retrieve position and orientation information about each of the sources active in the system. One structure is required for each source.

Members:

*id*: This is the zero-based id of the source. It will be between 0 and 7 and corresponds to the order in which the source was placed in the source configuration file.

*freq*: This is the zero-based frequency value, also between 0 and 7. These values correspond to the A-H frequency designators marked on the source themselves. 0->A, 1->B, 2->C, etc.

*pos*: An array of three floats that correspond to the sources positional location. The values are X, Y, and Z respectively in the units that the system has been configured. Default is inches.

*att*: An array of three or four floats depending on the system's orientation configuration. If the orientation has been configured to Euler Degrees or Euler Radians, the array will hold three valid values that correspond to Azimuth (Yaw), Elevation (Pitch), and Roll respectively. If the orientation has been configured to output quaternions, the array will hold four valid values corresponding to the four quaternion values.

**G4_CMD_BLOCK_STRUCT**

```
typedef struct _G4_CMD_BLOCK_STRUCT {
        int units[2];                       // pos, att
        uint8_t version_info[50];
        float filter_params[2][4];      // pos,att
        float increment[G4_SENSORS_PER_HUB][2];             // pos, att
        float rot_angles[3];    // az, el, rl
        float translate_xyz[3];
        float tip_offset[G4_SENSORS_PER_HUB][3];
}*LPG4_CMD_BLOCK_STRUCT,G4_CMD_BLOCK_STRUCT;
```

This structure is used only when the G4_CMD_BLOCK_CFG command is used with the *g4_set_query* function. It can be used to set or read many configurations at once rather than in a one-at-a-time sequence. A pointer to this structure would be passed into the pParam variable of the G4_CMD_DATA_STRUCT structure when using the block read/write command.

Members:

*units*: An array of two ints that correspond to the position and orientation units. If setting the system units will be set to the values contained in this array, if getting, the values in this array will be set to the current system units. See documentation on G4_CMD_UNITS.

*version_info*: Get only. The system version information will be copied here. See documentation on G4_CMD_WHOAMI.

*filter_params*: Two arrays of four elements each. The first is for the position filter values, the second is for the orientation filter values. If setting the system filters will be set to these values, if getting these values will be set to current system filter values. See documentation on G4_CMD_FILTER.

*increment*: An array of two elements each for each sensor on a hub. One is for the positional increment value, the second for the orientation increment command. See documentation for G4_CMD_INCREMENT.

*rot_angles*: An array of three floats that will either set or return the frame-of-reference rotation values of the system. These values correspond to Azimuth (Yaw), Elevation (Pitch), and Roll respectively. See documentation for G4_CMD_FOR_ROTATE.

*translate_xyz*: An array of three float that will either set or return the frame-of-reference translation values of the system. These values correspond to the X, Y, and Z values of the translation respectively. See documentation for G4_CMD_FOR_TRANSLATE.

*tip_offset*: An array of three floats for each sensor on the hub. These values correspond to the X, Y, and Z values of the offset respectively. See documentation for G4_CMD_TIP_OFFSET.

## enums

Following are the enums that host software applications must use to interface with the G4Track Library:

```
enum {
        G4_CMD_WHOAMI,
        G4_CMD_GETMAXSRC,
        G4_CMD_BORESIGHT,
        G4_CMD_FILTER,
        G4_CMD_INCREMENT,
        G4_CMD_FOR_ROTATE,
        G4_CMD_FOR_TRANSLATE,
        G4_CMD_TIP_OFFSET,
        G4_CMD_UNITS,
        G4_CMD_GET_ACTIVE_HUBS,
        G4_CMD_GET_STATION_MAP,
        G4_CMD_GET_SOURCE_MAP,
        G4_CMD_FRAMERATE,
        G4_CMD_RESTORE_DEF_CFG,
        G4_CMD_BLOCK_CFG,
        G4_TOTAL_COMMANDS
};
```

These are the valid values to be passed into the G4_CMD_STRUCT.cmd variable when using the *g4_set_query* function.

```
enum {G4_ACTION_SET,G4_ACTION_GET,G4_ACTION_RESET};
```

These are the valid values to determine whether a configuration is being set, read, or reset. This value would be passed into the G4_CMD_DATA_STRUCT.action member.

```
enum dataType {G4_DATA_POS,G4_DATA_ORI};
```

These values indicate whether a command is getting/setting a position or orientation configuration.

```
enum {
        G4_TYPE_EULER_DEGREE,
        G4_TYPE_EULER_RADIAN,
        G4_TYPE_QUATERNION,
        G4_TYPE_INCH,
        G4_TYPE_FOOT,
        G4_TYPE_CM,
        G4_TYPE_METER,
};
```

These values specify the different position and orientation units/formats supported by the G4Track Library.

## Macros

These two macros should be used to generate the value for G4_CMD_DATA_STRUCT.id.

**G4_CREATE_ID(sys,hub,sensor)**
This macro will combine the system id, the hub id, and the sensor id to create one value to be passed in the G4_CMD_DATA_STRUCT.id value.

**G4_CREATE_ID_SENS_MAP(sys,hub,sensorMap)**
Similar to the above macro except that instead of a single sensor, a bitmap of sensors can be passed in to configure multiple sensors on a hub at once.

## Errors

These are the errors that may be returned by the various functions in the G4Track Library

```
enum {
        G4_ERROR_NONE,
        G4_ERROR_NO_FRAME_DATA_AVAIL=-100,
        G4_ERROR_UNSUPPORTED_ACTION,
        G4_ERROR_UNSUPPORTED_TYPE,
        G4_ERROR_UNSUPPORTED_COMMAND,
        G4_ERROR_INVALID_STATION,
        G4_ERROR_NO_CONNECTION,                 //-95
        G4_ERROR_NO_HUBS,
        G4_ERROR_FRAMERATE_SET,
        G4_ERROR_MEMORY_ALLOCATION,
        G4_ERROR_INVALID_SYSTEM_ID,
        G4_ERROR_SRC_CFG_FILE_OPEN,             //-90
        G4_ERROR_INVALID_SRC_CFG_FILE,
        G4_ERROR_UNABLE_TO_START_TIMER,
        G4_ERROR_HUB_NOT_ACTIVE,
        G4_ERROR_SYS_RESET_FAIL,
        G4_ERROR_DONGLE_CONNECTION,             //-85
        G4_ERROR_DONGLE_USB_CONFIGURATION,
        G4_ERROR_DONGLE_USB_INTERFACE_0,
        G4_ERROR_DUPLICATE_SYS_IDS,
        G4_ERROR_INVALID_WILDCARD_USE,
        G4_ERROR_TOTAL
};
```

Note that zero indicates a no-error condition while negative values (starting at -100 and counting toward zero) indicate various errors.

# Functions

The following four functions define the host application interface to the G4Track Library:

**g4_init_sys**

uint32_t **g4_init_sys**(int* *pDongleId*,const char* *src_cfg_file*,void* *reserved*)

This function should be called to initialize the G4Track Library. It must be called before any interaction with the library can take place. This function will send the proper source configuration data to the system and start all collection and I/O threads necessary to begin the tracker host interface communications. This function will return a pointer to system ID in the *pDongleId* parameter. The *src_cfg_file* parameter is the path to a file that has been created by one of the Source File Configuration Creation programs supplied with this library. To create a custom Source File Creation application or to embed it in your application, see the documentation for the Source File Configuration Library.
Note: the *reserved* parameter must be NULL.

**Return Value**: G4_ERROR_NONE if successful, one of the other errors if not.

**Parameters**:

*pDongleId*: A pointer to the system ID will be returned here.
*src_cfg_file*: The source configuration file.
*reserved*: Must be NULL.

**Example**:

```
const char* src_file=”sample_src_cfg_file.g4c”;
int sysId;
int res;

res=g4_init_sys(&sysId,src_file,NULL);
if (res!=G4_ERROR_NONE)
        return false;
// otherwise connection is ok, proceed with business

// when ready to quit
g4_close_tracker();
```

**g4_get_frame_data**

uint32_t **g4_get_frame_data**(G4_FRAMEDATA* *fd_array*, int *sysId*,
const int* *hub_id_list*, int *num_hubs*)

This is the function that enables the host application to retrieve position and orientation data from the G4Track Library. The host application must provide a list of G4_FRAMEDATA structures to receive the data, and a list of hub ids to identify which hubs' data to retrieve. This function must be called for each frame of data required by the host application.

**Return Value:** A 32-bit value where the upper 16-bit word contains the total number of active hubs on the system, and the lower 16-bit word contains the number of hubs worth of data returned in *fd_array*.

**Parameters:**

*fd_array*: An array of G4_FRAMEDATA structures. One must be supplied for each hub. The library will fill each structure with the data for the corresponding hub.

*sysId*: This must contain the system ID that was returned in the **g4_init_sys** function.

*hub_id_list*: An array of hub ids that the host application is requesting the position and orientation data for.

*num_hubs*: The number of hubs requesting data for. This should also be the number of elements in *fd_array* and *hub_id_list*.

**Example:**

```
// Determine number of hubs
G4_CMD_STRUCT cs;
int hubs;

cs.cmd=G4_CMD_GET_ACTIVE_HUBS;
cs.cds.id=G4_CREATE_ID(sysId,0,0);
cs.cds.action=G4_ACTION_GET;
cs.cds.pParam=NULL;
int res=g4_set_query(&cs);
hubs=cs.cds.iParam;          // number of hubs in cs.cds.iParam

        // create hub list and G4_FRAMEDATA array
int* hubList=new int[hubs];
cs.cds.pParam=hubList;
res=g4_set_query(&cs);
G4_FRAMEDATA* fd=new G4_FRAMEDATA[hubs];

res=g4_get_frame_data(fd,sysId,hubList,hubs);
```

```
int num_hubs_read=res&0xffff;
int tot_sys_hubs=res>>16;

// clean up
delete[] hubList;
delete[] fd;
```

**g4_set_query**

uint32_t **g4_set_query**(LPG4_CMD_STRUCT *pcs*)

This function is used by the host application to read and write all configuration parameters to the G4Track Library.

**Return Value:** G4_ERROR_NONE if successful, one of the other errors if not.

**Parameters:**

*pcs*: A pointer to a G4_CMD_STRUCT struct that defines the action and configuration to be performed by this function. See Remarks for illustrations on how to use this function for each of the different commands.

**Remarks:**

Every command requires different parameters in the G4_CMD_STRUCT. Each is illustrated below. All assume that **g4_sys_init** has been successfully called in previous code and **g4_close_tracker** will be called in later code. Note that most error checking has been omitted for clarity.

Command: *Who am I*

This command will return the version information of the G4Track library.

G4_CMD_STRUCT.cmd=G4_CMD_WHOAMI
G4_CMD_DATA_STRUCT.id=Not used
G4_CMD_DATA_STRUCT.action= Not used
G4_CMD_DATA_STRUCT.iParam= Not used
G4_CMD_DATA_STRUCT.pParam=buffer of at least 32 bytes

Example:

```
G4_CMD_STRUCT cs;
char buf[50];
cs.cmd=G4_CMD_WHOAMI;
cs.cds.pParam=buf;
rv=g4_set_query(&cs);
```

Command: *Get Max Sources*

This command will return the maximum number of sources allowed by the installed firmware.

G4_CMD_STRUCT.cmd=G4_CMD_GETMAXSRC
G4_CMD_DATA_STRUCT.id=not used

G4_CMD_DATA_STRUCT.action= not used
G4_CMD_DATA_STRUCT.iParam=Max number of sources returned here
G4_CMD_DATA_STRUCT.pParam= not used

Example:

```
G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_GETMAXSRC;
rv=g4_set_query(&cs);
int max_sources=cs.cds.iParam;
```

Command: *Boresight*

This command will either boresight or unboresight individual sensors. See manual for a definition of boresight.

G4_CMD_STRUCT.cmd=G4_CMD_BORESIGHT
G4_CMD_DATA_STRUCT.id=sys,hub,sensor id
For id use G4_CREATE_ID(sys id,hub id,sensor id)
or G4_CREATE_ID_SENS_MAP(sys id,hub id,sensor map)
if hub is -1, boresight is performed on all hubs, sensors on indicated system(set/reset only)

G4_CMD_DATA_STRUCT.action=G4_ACTION_SET or G4_ACTION_GET or G4_ACTION_RESET. Use G4_ACTION_RESET to unboresight.
G4_CMD_DATA_STRUCT.iParam=G4_TYPE_EULER_DEGREE or G4_TYPE_EULER_RADIAN or G4_TYPE_QUATERNION
G4_CMD_DATA_STRUCT.pParam=float* to array of boresight angles(set) or array to receive angles(get)

Example:

```
float vals[3]={0.0f,0.0f,0.0f};

G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_BORESIGHT;
cs.cds.id=G4_CREATE_ID(sysId, hubId,0);          //  sens 0
cs.cds.action=G4_ACTION_SET;
cs.cds.iParam=G4_TYPE_EULER_DEGREE;
cs.cds.pParam=vals;
rv=g4_set_query(&cs);                  // sensor 0 set boresighted to 0,0,0

cs.cds.id=G4_CREATE_ID_SENS_MAP(sysId, hubId,7); // for all 3 sensors
rv=g4_set_query(&cs);                  // boresights all sensors (0x0111) of hub
```

```
cs.cmd=G4_CMD_BORESIGHT;
cs.cds.id=G4_CREATE_ID(sysId, hubId,0);
cs.cds.action=G4_ACTION_GET;
cs.cds.iParam=G4_TYPE_EULER_DEGREE;
cs.cds.pParam=vals;
rv=g4_set_query(&cs);                // sensor 0 boresight angles returned in vals.
```

Command: *Filter Settings*

This command allows the host application to modify the filter settings for position and orientation. See manual for a description of the G4 Filters.

G4_CMD_STRUCT.cmd=G4_CMD_FILTER
G4_CMD_DATA_STRUCT.id= sys, hub id -- use G4_CREATE_ID(sys id, hub id, 0)
if hub is -1, action is performed on all hubs on indicated system(set/reset only)

G4_CMD_DATA_STRUCT.action=G4_ACTION_SET or G4_ACTION_GET or G4_ACTION_RESET
G4_CMD_DATA_STRUCT.iParam=G4_DATA_POS or G4_DATA_ORI
G4_CMD_DATA_STRUCT.pParam=float* to array of new filter values(set) or array to receive filter values(get)

Example:

```
float filt[4]={0.1f,0.1f,0.1f,0.1f};

G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_FILTER;
cs.cds.id=G4_CREATE_ID(sysId, hubId,0);
cs.cds.action=G4_ACTION_SET;
cs.cds.iParam=G4_DATA_POS;
cs.cds.pParam=filt;
rv=g4_set_query(&cs);                // filters for hub  set to 0.1,0.1,0.1,0.1


memset(filt,0,sizeof(filt));
cs.cmd=G4_CMD_FILTER;
cs.cds.id=G4_CREATE_ID(sysId, hubId,0);
cs.cds.action=G4_ACTION_GET;
cs.cds.iParam=G4_DATA_POS;
cs.cds.pParam=filt;
```

```
rv=g4_set_query(&cs);                    // filter values for hub placed in filt

cs.cmd=G4_CMD_FILTER;
cs.cds.id=G4_CREATE_ID(sysId, hubId,0);
cs.cds.action=G4_ACTION_RESET;
cs.cds.iParam=G4_DATA_POS;
cs.cds.pParam=NULL;
rv=g4_set_query(&cs);              // filter values for hub  reset to default
```

Command: *Increment*

This command allows the host application to set or get the position or orientation increment. See the manual for a description of the increment and auto increment command.

G4_CMD_STRUCT.cmd=G4_CMD_INCREMENT
CMD_DATA_STRUCT.id=sys,hub,sensor id
For id use CREATE_ID(sys id,hub id,sensor id)
or CREATE_ID_SENS_MAP(sys id,hub id,sensor map)
if hub is -1, action is performed on all hubs,sensors on indicated system(set/reset only)
G4_CMD_DATA_STRUCT.action=G4_ACTION_SET or G4_ACTION_GET or G4_ACTION_RESET
G4_CMD_DATA_STRUCT.iParam=This variable should be loaded with the units for the pos in the upper 16-bit word, and the units for the orientation in the lower 16-bit word.
G4_CMD_DATA_STRUCT.pParam=float* to an array of 2 new increment values (set) or array to receive existing values (get)
Array must be large enough to hold two floats. The first value is the position increment and the second value is the orientation increment. The units of the values must be the same as that of the output format setting.

Note: a negative value for pos or orientation increment will turn on auto increment.

Example:

```
float incr[2]={0.2f,1.5};

G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_INCREMENT;
cs.cds.id=G4_CREATE_ID(sysId, hubId,0);
cs.cds.iParam=(G4_TYPE_INCH<<16)|G4_TYPE_EULER_DEGREE;
cs.cds.action=G4_ACTION_SET;
```

```
cs.cds.pParam=incr;
rv=g4_set_query(&cs);        // increments set for hub 1,
                             // sensor 1 (param is zero based)

cs.cds.id=G4_CREATE_ID_SENS_MAP(sysId, hubId,7); // for all 3 sensors
rv=g4_set_query(&cs);        // sets increments for all sensors (0x0111) of hub


incr[0]=incr[1]=0.0f;
cs.cds.id=G4_CREATE_ID(sysId, hubId,0);
cs.cds.action=G4_ACTION_GET;
cs.cds.pParam=incr;
rv=g4_set_query(&cs);        // increment for hub 3, sensor 0 placed in incr

incr[0]=incr[1]=-1.0;
cs.cds.id=G4_CREATE_ID(sysId, hubId,0);
cs.cds.action=G4_ACTION_SET;
cs.cds.pParam=incr;
rv=g4_set_query(&cs);              // increments set to auto incr
                                   // for hub 3, sensor 0

cs.cds.id=G4_CREATE_ID(sysId, hubId,0);
cs.cds.action=G4_ACTION_RESET;
rv=g4_set_query(&cs);              // increments reset to zero
```

Command*: Frame of Reference Rotation*

This command allows the host application to rotate the frame of reference in azimuth (yaw), elevation (pitch), and roll.

```
G4_CMD_STRUCT.cmd=G4_CMD_FOR_ROTATE
G4_CMD_DATA_STRUCT.id=sys id -- use G4_CREATE_ID(sysid,0,0)
G4_CMD_DATA_STRUCT.action=G4_ACTION_SET or G4_ACTION_GET or
                          G4_ACTION_RESET
G4_CMD_DATA_STRUCT.iParam=G4_TYPE_EULER_DEGREE or
                          G4_TYPE_EULER_RADIAN or
                          G4_TYPE_QUATERNION
G4_CMD_DATA_STRUCT.pParam=float* to array of new rotation values(set) or array
                                 to receive rotation values(get)
```

Example:

```
float rot[3]={0.0f,30.0f,60.0f};
```

```
G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_FOR_ROTATE;
cs.cds.id=G4_CREATE_ID(sysId,0,0);
cs.cds.action=G4_ACTION_SET;
cs.cds.iParam=G4_TYPE_EULER_DEGREE;
cs.cds.pParam=rot;
rv=g4_set_query(&cs);                    // set system rotation to 0,30,60


cs.cds.action=G4_ACTION_GET;
cs.cds.iParam=G4_TYPE_EULER_DEGREE;
cs.cds.pParam=rot;
rv=g4_set_query(&cs);          // get current rotaton values in rot

cs.cds.action=G4_ACTION_RESET;
cs.cds.iParam=G4_TYPE_EULER_DEGREE;
cs.cds.pParam=NULL;
rv=g4_set_query(&cs);                    // reset rotation values to 0,0,0 for system
```

Command: *Frame of Reference Translation*

This command allows the user host application to translate the frame of reference in X, Y, and Z.

G4_CMD_STRUCT.cmd=G4_CMD_FOR_TRANSLATE
G4_CMD_DATA_STRUCT.id=sys id -- use G4_CREATE_ID(sysid,0,0)
G4_CMD_DATA_STRUCT.action=G4_ACTION_SET or G4_ACTION_GET or
G4_ACTION_RESET
G4_CMD_DATA_STRUCT.iParam=G4_TYPE_INCH or G4_TYPE_FOOT or
G4_TYPE_CM or G4_TYPE_METER
G4_CMD_DATA_STRUCT.pParam=float* to array of new translation values(set) or
array to receive translation values(get)

Example:

```
float trans[3]={3.0f,-1.0f,-3.0f};

G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_FOR_TRANSLATE;
cs.cds.id=G4_CREATE_ID(sysId,0,0);
cs.cds.action=G4_ACTION_SET;
cs.cds.iParam=G4_TYPE_INCH;
cs.cds.pParam=trans;
rv= g4_set_query(&cs);                   // sets translation values 3,-1,-3 for system
```

```
memset(trans,0,sizeof(trans));
cs.cds.action=G4_ACTION_GET;
cs.cds.pParam=trans;
rv= g4_set_query(&cs);        // gets translation for system and puts in trans
```

Command: *Tip Offsets*

This command allows the user to extend or offset the center of the sensor by values of X,Y, and Z.

```
G4_CMD_STRUCT.cmd=G4_CMD_TIP_OFFSET
G4_CMD_DATA_STRUCT.id=sys, hub, sens id
                    For id use G4_CREATE_ID(sys id,hub id,sensor id)
                    or G4_CREATE_ID_SENS_MAP(sys id, hub id, sensor map)
                    if hub is -1, tip offset is performed on all hubs,sensors on indicated
                        system(set/reset only)
G4_CMD_DATA_STRUCT.action=G4_ACTION_SET or G4_ACTION_GET or
                                G4_ACTION_RESET
G4_CMD_DATA_STRUCT.iParam=G4_TYPE_INCH or G4_TYPE_FOOT or
                                G4_TYPE_CM or G4_TYPE_METER
G4_CMD_DATA_STRUCT.pParam=float* to array of new offset values(set) or array to
                                    receive offset values(get)
```

Example:

```
float toff[3]={0.5f,0.0f,0.5f};


G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_TIP_OFFSET;

cs.cds.id=G4_CREATE_ID(sysId, hubId,2); // for sensor 3
                                        // (parameter is zero based)
cs.cds.action=G4_ACTION_SET;
cs.cds.iParam=G4_TYPE_CM;
cs.cds.pParam=toff;
rv=g4_set_query(&cs);           //sets tip offset for sensor 2 to
                                //      0.5,0.0,0.5

cs.cds.id=G4_CREATE_ID_SENS_MAP(sysId, hubId,7); // for all 3 sensors
rv=g4_set_query(&cs);           // sets tip offset for all sensors (0x0111) of
                                // hub
```

```
memset(toff,0,sizeof(float)*3);
cs.cds.id=G4_CREATE_ID(sysId, hubId,2);
cs.cds.action=G4_ACTION_GET;
cs.cds.iParam=G4_TYPE_CM;
cs.cds.pParam=toff;
rv=g4_set_query(&cs);          // writes sensor 2 tip offset to toff buffer

cs.cds.id=G4_CREATE_ID(sysId,3,2);
cs.cds.action=G4_ACTION_RESET;
rv=g4_set_query(&cs);                  // resets hub 3, sensor 2 tip offset to 0,0,0
```

Command: *Units*

CMD_STRUCT.cmd=G4_CMD_UNITS
CMD_DATA_STRUCT.id=sys id, hub id -- use G4_CREATE_ID(sys id,0,0)
CMD_DATA_STRUCT.action=G4_ACTION_SET or G4_ACTION_GET or
G4_ACTION_RESET
CMD_DATA_STRUCT.iParam=G4_DATA_POS or G4_DATA_ORI
CMD_DATA_STRUCT.pParam=points to int (one of the type enums) to set, int* to get.

Units is specific to a system.

Example:

```
G4_CMD_STRUCT cs;
int quat_unit=G4_TYPE_QUATERNION;
cs.cmd=G4_CMD_UNITS;
cs.cds.id=G4_CREATE_ID(sysId,0,0);
cs.cds.action=G4_ACTION_SET;
cs.cds.iParam=G4_DATA_ORI;
cs.cds.pParam=(void*)&quat_unit;
rv=g4_set_query(&cs);                  // sets orientation units to quaternions

int type;
cs.cmd=G4_CMD_UNITS;
cs.cds.id=G4_CREATE_ID(sysId,0,0);
cs.cds.action=G4_ACTION_GET;
cs.cds.iParam=G4_DATA_ORI;
cs.cds.pParam=&type;
rv=g4_set_query(&cs);                  // get the orientation units in variable type
```

Command: *Get Active Hubs*

This command will return the quantity and ids of all hubs active on a system. This command should be called first with pParam set to NULL to determine the number of active hubs. Then an array large enough can be provided on the second call to get the hub ids.

G4_CMD_STRUCT.cmd=G4_CMD_GET_ACTIVE_HUBS
G4_CMD_DATA_STRUCT.id= Sys Id -- use G4_CREATE_ID(sys id,0,0)
G4_CMD_DATA_STRUCT.action=unused -- get action only
G4_CMD_DATA_STRUCT.iParam= the quantity of active hubs is always returned here
G4_CMD_DATA_STRUCT.pParam= points to an array of ints large enough to hold all the active hub ids

Example:

```
G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_GET_ACTIVE_HUBS;
cs.cds.id=G4_CREATE_ID(sysId,0,0);
cs.cds.pParam=NULL;                          // return qty in iParam
rv=g4_set_query(&cs);

int* hubarr=new int[cs.cds.iParam];
cs.cds.pParam=hubarr;
rv=g4_set_query(&cs);                  // hubarr is filled with list of active hubs
//... // when finished clean up memory
delete[] hubarr;
```

Command: *Get Station Map*

This command allows the host application to query any specific hub to find the active sensors. This information is also available in the G4_FRAMEDATA struct that is returned from the **g4_get_frame_data** function.

G4_CMD_STRUCT.cmd=G4_CMD_GET_STATION_MAP
G4_CMD_DATA_STRUCT.id= Sys and Hub Id -- use G4_CREATE_ID(sys id,hub,0)
G4_CMD_DATA_STRUCT.action=unused -- get action only
G4_CMD_DATA_STRUCT.iParam= The hub's station map is returned here
G4_CMD_DATA_STRUCT.pParam= unused

Example:

```
G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_GET_STATION_MAP;
cs.cds.id=G4_CREATE_ID(sysId, hubId,0);
rv=g4_set_query(&cs);
int stationMap=cs.cds.iParam;
```

Command: *Get Source Map*

This command allows the host application to obtain a list of the system sources. This list contains their ids, frequencies, position, and orientation data. This command should be called first with pParam set to NULL to determine the number of sources. Then an array large enough can be provided on the second call to get the source info.

G4_CMD_STRUCT.cmd=G4_CMD_GET_SOURCE_MAP
G4_CMD_DATA_STRUCT.id= Sys Id -- use G4_CREATE_ID(sys id,0,0)
G4_CMD_DATA_STRUCT.action=unused -- get only
G4_CMD_DATA_STRUCT.iParam= The quantity of sources is always returned here. If pParam is not NULL, this variable should be loaded with the units for the pos in the upper 16-bit word, and the units for the orientation in the lower 16-bit word.
G4_CMD_DATA_STRUCT.pParam=Points to an array of SRC_MAP structures large enough to hold all the source info or NULL if getting quantity only.

Example:

```
G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_GET_SOURCE_MAP;
cs.cds.id=G4_CREATE_ID(sysId,0,0);
cs.cds.pParam=NULL;
rv=g4_set_query(&cs);

LPG4_SRC_MAP map=new G4_SRC_MAP[cs.cds.iParam];        // create large
                                                       //enough array
cs.cds.pParam=map;
cs.cds.iParam=(G4_TYPE_INCH<<16)|G4_TYPE_EULER_DEGREE;
rv=g4_set_query(&cs);
//... // when finished clean up memory
delete[] map;
```

Command: *Restore Default Configuration*

This command will restore the G4Library to it's default configuration.

G4_CMD_STRUCT.cmd=G4_CMD_RESTORE_DEF_CFG
G4_CMD_DATA_STRUCT.id= Sys Id -- use G4_CREATE_ID(sys id,0,0)
This is a system(dongle) wide reset

G4_CMD_DATA_STRUCT.action=unused
G4_CMD_DATA_STRUCT.iParam=unused
G4_CMD_DATA_STRUCT.pParam=unused

Example:

```
G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_RESTORE_DEF_CFG;
cs.cds.id=G4_CREATE_ID(sysId,0,0);
rv=g4_set_query(&cs);
```

Command: *Block Read/Write*

This command allows a host application to set or read multiple configuration parameters with just one call to **g4_set_query**. The command utilizes the G4_CMD_BLOCK_STRUCT structure.

G4_CMD_STRUCT.cmd=G4_CMD_BLOCK_CFG
G4_CMD_DATA_STRUCT.id= SysId, hubId -- use G4_CREATE_ID(sys id,hub,0)
G4_CMD_DATA_STRUCT.action=G4_ACTION_SET or G4_ACTION_GET or G4_ACTION_RESET
G4_CMD_DATA_STRUCT.iParam=this variable should be loaded with the units for the pos in the upper 16-bit word, and the units for the orientation in the lower 16-bit word.
G4_CMD_DATA_STRUCT.pParam=points to a G4_CMD_BLOCK_STRUCT struct to receive current data, or containing data to set.

Example:

```
LPG4_CMD_BLOCK_STRUCT pBlock=new G4_CMD_BLOCK_STRUCT;

// read current data
G4_CMD_STRUCT cs;
cs.cmd=G4_CMD_BLOCK_CFG;
cs.cds.id=G4_CREATE_ID(sysId,hubId,0);
cs.cds.action=G4_ACTION_GET;
cs.cds.iParam=(G4_TYPE_INCH<<16|G4_TYPE_EULER_DEGREE);
cs.cds.pParam=pBlock;
rv=g4_set_query(&cs);

// modify values.....
//.........
// write back
cs.cds.action=G4_ACTION_SET;
```

```
// or reset
cs.cds.action=G4_ACTION_RESET;

rv=g4_set_query(&cs);          // write or reset

delete pBlock;
```

**g4_close_tracker**

void **g4_close_tracker**(void)

This function should be called when the application wishes to close the connection with the G4Library. This will allow the library to cleanly close down all I/O ports and collection threads.

**Return Value:** None.

**Parameters:** None

**Example:**

```
const char* src_file=”sample_src_cfg_file.g4c”;
int sysId;
int res;

res=g4_init_sys(&sysId,src_file,NULL);
if (res!=G4_ERROR_NONE)
        return false;
// otherwise connection is ok, proceed with business

// when ready to quit
g4_close_tracker();
```