Vuzix Eyewear
Software Development Kit
Version 3.0

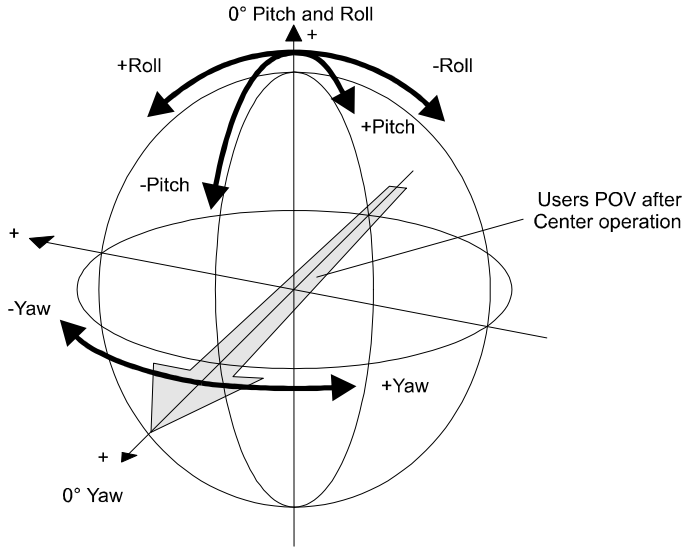
This document describes the details of the Eyewear SDK for Microsoft Windows™ operating systems. The SDK is composed of functions that allow applications to interface to the Vuzix Eyewear tracking system.

TRACKER DRIVER

Tracker Coordinate Reporting

The **IWRGetTracking** function call will return values between -32768 to 32768.
This approach allows default coordinates of the head tracker to be consistent with a normal joystick i.e., yaw centered is looking straight ahead. In the current implementation, yaw and roll are valid over the range of +/-180° and pitch is valid over the range of +/-90°. The zero position is forward for yaw, incrementing to the left; forward for pitch, incrementing as you look up; and forward for roll, incrementing as you turn your head towards your right shoulder.

HMD Axes Mapping



TRACKER API

DWORD IWROpenTracker (void)

Initiate session with the head tracker.

Returns:

DWORD Result of open call

- ERROR_SUCCESS - Tracker opened
- ERROR_DEV_NOT_EXIST - Tracker not attached to computer
- ERROR_NOT_ENOUGH_MEMORY - Error with thread storage

void IWRCloseTracker (void)

Terminate a session with the head tracker.

void IWRZeroSet (void)

Marks the user's current head position as "straight ahead and level".

The user's current head postion is saved as a set of offsets so all future readings are taken relative to the current head position.

void IWRSetFilterSate (BOOL *on*)

Sets the state of the low-pass digital filter (VR920 only).

Parameters:

on

- TRUE = filter is active
- FALSE = filter is inactive

BOOL IWRGetFilterState (void)

Returns the current state of the low-pass digital filter (VR920 only).

Returns:

BOOL state of filter

- TRUE = filter is active
- FALSE = filter is inactive

DWORD IWRBeginCalibrate (void)

Start the calibration procedure for the tracker to adapt the sensors to the user's location.

The magnetic sensors on the tracker are affected by the user's surroundings. To provide the smoothest tracking possible, the user must undergo a process of finding the maximum and minimum values the sensors will "see" in the current environment.

Calling this function removes any previous calibration data and begins the process in an "ideal" state.

Returns:

DWORD Result of this function call

- ERROR_SUCCESS - Tracker opened
- ERROR_DEV_NOT_EXIST - Tracker not attached to computer

void IWREndCalibrate (BOOL *save*)

Finish the calibration procedure for the tracker and optionally save the calibration data.

Parameters:

save

- TRUE = save calibration data
- FALSE = abandon calibration data

DWORD IWRGetTracking (LONG * *yaw*, LONG * *pitch*, LONG * *roll*)

Get the user's current head position.

Returns the current yaw, pitch, and roll positions of the user's head. All position values are in the range of -32768 to 32768.

Parameters:

yaw receives the user's head position in the yaw axis, i.e. looking left or right. Where:

- -32768 = -180 degrees = looking over user's right shoulder
- 0 = looking straight ahead
- 32767 = 180 degrees = looking over user's left shoulder

pitch receives the user's head position in the pitch axis i.e. looking up or down. Where:

- -16384 = -90 degrees = looking down
- 0 = looking straight ahead
- 16383 = 90 degrees = looking up

roll receives the user's head position in the roll axis i.e. is the head tilted left or right.

- -32768 = -180 degrees = tilting left
- 0 = head is straight up and down
- 32768 = 180 degrees = tilting right

Returns:

DWORD Result of this call

- ERROR_SUCCESS - Tracker responded, data is good
- ERROR_DEV_NOT_EXIST - Tracker not attached to computer

DWORD IWRGet6DTracking(

LONG *yaw,
LONG *pitch,
LONG *roll,
LONG *xtrn,
LONG *ytrn,
LONG *ztrn

)

Returns the current yaw, pitch, and roll positions and the motion of the user's head. (Wrap920 only).

Parameters:

yaw receives the user's head position in the yaw axis, i.e. looking left or right. Where:

- -32768 = -180 degrees = looking over user's right shoulder
- 0 = looking straight ahead
- 32767 = 180 degrees = looking over user's left shoulder

pitch receives the user's head position in the pitch axis i.e. looking up or down. Where:

- -16384 = -90 degrees = looking down
- 0 = looking straight ahead
- 16383 = 90 degrees = looking up

roll receives the user's head position in the roll axis i.e. is the head tilted left or right.

- -32768 = -180 degrees = tilting left
- 0 = head is straight up and down
- 32768 = 180 degrees = tilting right

xtrn receives the motion along the X axis i.e. moving forward or back.
ytrn receives the motion along the Y axis i.e. moving left or right.
ztrn receives the motion along the Z axis i.e. moving up or down.

Return Values:

DWORD Result of this call

- ERROR_SUCCESS - Tracker responded, data is good
- ERROR_DEV_NOT_EXIST - Tracker not attached to computer

NOTE:

In this release of the SDK (3.0), this function should be considered ”in beta”. The xtrn, ytrn, and ztrn values should only be used as indicators of general motion.

DWORD IWRGetSensorData(IWRSENSDATA **sensdataptr*)

Get the latest raw sensor values from the tracker. (Wrap920 only)

Parameters:

sensdataptr a pointer to a IWRSENSDATA structure, allocated by the caller, that receives the raw sensor data.

Returns:

DWORD Result of this call

- ERROR_SUCCESS - Tracker responded, data is good
- ERROR_NOT_SUPPORTED – Tracker does not support this fuction (i.e. VR920)
- ERROR_DEV_NOT_EXIST - Tracker not attached to computer

The example program **HelloTracker3** illustrates how to use this function.

IWRSENSDATA Structure

```
typedef struct tag_IWRMAGSENSOR {
    unsigned char magx_lsb;
    unsigned char magx_msb;
    unsigned char magy_lsb;
    unsigned char magy_msb;
    unsigned char magz_lsb;
    unsigned char magz_msb;
} IWRMAGSENSOR, *IWRPMAGSENSOR;

typedef struct tag_IWRACCELSENSOR {
    unsigned char accx_lsb;
    unsigned char accx_msb;
    unsigned char accy_lsb;
    unsigned char accy_msb;
    unsigned char accz_lsb;
    unsigned char accz_msb;
} IWRACCELSENSOR, *PIWRACCELSENSOR;

// High Bandwidth gyros 2000 deg per second
typedef struct tag_IWRGYROSENSOR {
    unsigned char gyx_lsb;
    unsigned char gyx_msb;
    unsigned char gyy_lsb;
    unsigned char gyy_msb;
    unsigned char gyz_lsb;
    unsigned char gyz_msb;
} IWRGYROSENSOR, *PIWRGYROSENSOR;

// Low Bandwidth gyros 500 deg per second
typedef struct tag_IWRLBGYROSENSOR {
    unsigned char gyx_lsb;
    unsigned char gyx_msb;
    unsigned char gyy_lsb;
    unsigned char gyy_msb;
    unsigned char gyz_lsb;
    unsigned char gyz_msb;
} IWRLBGYROSENSOR, *PIWRLBGYROSENSOR;

typedef struct tag_IWRSENSDATA {
    IWRMAGSENSOR mag_sensor;
    IWRACCELSENSOR acc_sensor;
    IWRGYROSENSOR gyro_sensor;
    IWRLBGYROSENSOR lbgyro_sensor;
} IWRSENSDATA, *PIWRSENSDATA;
```

magx_msb,magx_lsb – can be combined into a single 16-bit 2' compliment number with a range of -2048 – 2047 for the magnetic sensor in the x-direction.
magy_msb,magy_lsb – can be combined into a single 16-bit 2' compliment number with a range of -2048 – 2047 for the magnetic sensor in the y-direction.
magz_msb,magz_lsb – can be combined into a single 16-bit 2' compliment number with a range of -2048 – 2047 for the magnetic sensor in the z-direction.

accx_msb,accx_lsb – can be combined into a single 16-bit 2' compliment number with a range of -512 - 511 for the accelerometer sensor in the x-direction.
accy_msb,accy_lsb – can be combined into a single 16-bit 2' compliment number with a range of -512 – 511 for the accelerometer sensor in the y-direction.

accz_msb,accz_lsb – can be combined into a single 16-bit 2' compliment number with a range of -512 – 511 for the accelerometer sensor in the z-direction.
The accelerometers have a resolution of 1mg / count.

gyx_msb,gyx_lsb – can be combined into a single 16-bit 2' compliment number with a range of -2048 – 2047 for the gyroscope sensor in the x-direction.
gyy_msb,gyy_lsb – can be combined into a single 16-bit 2' compliment number with a range of -2048 – 2047 for the gyroscope sensor in the y-direction.
gyz_msb,gyz_lsb – can be combined into a single 16-bit 2' compliment number with a range of -2048 – 2047 for the gyroscope sensor in the z-direction.

The high bandwidth gyro sensor resolution is 0.824 degrees per second per count. The low bandwidth resolution is 0.205 degrees per second per count.

DWORD IWRGetFilteredSensorData(

LONG *ax,
LONG *ay,
LONG *az,
LONG *lgx,
LONG *lgy,
LONG *lgz,
LONG *gx,
LONG *gy,
LONG *gz,
LONG *mx,
LONG *my
LONG *mz

)

Get the latest filtered and stablilized sensor values from the tracker. (Wrap920 only).

Parameters:

ax – acceleration in the x axis
ay – acceleration in the y axis (all axes are 1mg per count)
az – acceleration in the z axis
lgx – low bandwidth gyro in the x axis
lgy – low bandwidth gyro in the y axis (all axes are 0.205 deg per sec per count)
lgz – low bandwidth gyro in the z axis
gx – high bandwidth gyro in the x axis
gy – high bandwidth gyro in the y axis (all axes are 0.824 deg per sec per count)
gz – high bandwidth gyro in the z axis
mx – magnetic field reading in the x axis
my – magnetic field reading in the y axis (mag axis data is unitless)
mz – magnetic field reading in the z axis

Returns:

DWORD Result of this call

- ERROR_SUCCESS - Tracker responded, data is good
- ERROR_NOT_SUPPORTED – Tracker does not support this fuction (i.e. VR920)
- ERROR_DEV_NOT_EXIST - Tracker not attached to computer

The example program **HelloTracker3** illustrates how to use this function.

Stereo Control

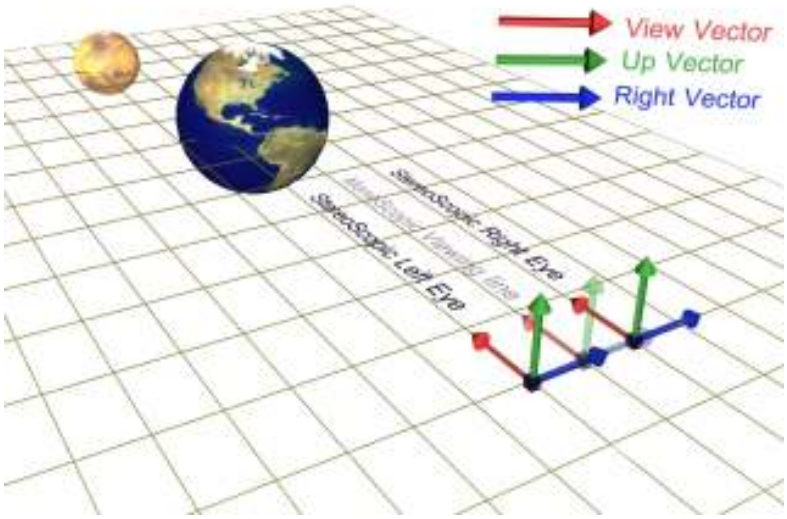
For the VR920, the Stereo Control API allows an application or kernel mode driver to designate on which eye the current frame is to be drawn. There are both user mode and kernel mode interfaces to this driver. In user mode the iwrstdrv.dll can be linked implicitly via an import library or explicitly via the LoadLibrary() function in the Win32 API. An example of how to use the user mode API is given in the VR920_Single_Device project in the examples folder.

A kernel mode driver can find and open the VR920 stereo control driver by registering for PnP notification for the interface identified by a specific GUID. An example of how to use this technique is given in the iwshelpr project in the examples folder. This is a very contrived yet simple example to demonstrate how to open and call the stereo control driver via a timer. The timer would not necessarily be the synchronization method of choice for your driver.

The Wrap920/VGA adapter combination only supports side-by-side stereo. The stereo API can only be used to switch the Wrap920 into and out of stereo mode.

iWear VR920 Stereo Rendering

The preferred method for stereoscopic rendering is presented in the SDK project workspace <VR920_SingleDevice_Stereoscopy>. The rendering engine depicted here is for a left handed system.
The standard view vectors are constructed in the application for creation of a cameras view into the virtual world. The entire rendering and presenting process for best stereo viewing, is defined in the following steps:



1. Render the “StereoScopic Left eye” by sliding the cameras <position and look at vectors> along the right vector to the left of the “Monoscopic viewing line” by the Stereoscopic IPD distance. Parallel stereo rendering.
2. Render the “StereoScopic Right eye” by sliding the cameras <position and look at vectors> along the right vector to the right of the “Monoscopic viewing line” by the Stereoscopic IPD distance. Parallel stereo rendering. NOTE: Do not allow object movement during right eye rendering.
3. Upon completion of the rendering of the left eye frame, ensure the VR920 is ready to accept the left eye frame. Present the left eye frame to the graphics engine and communicate to the VR920 a new left eye frame is ready for update.
4. Upon completion of the rendering of the right eye frame, ensure the VR920 is ready to accept the right eye frame. Present the right eye frame to the graphics engine and communicate to the VR920 a new right eye frame is ready for update.
5. Repeat to step #1.

The Above steps #3 + #4 is desirable to complete within one frame time.
The rendering of the right eye, is desirable to be such that no objects in the scene are to move from the left eyes position.
The project workspace demonstrates methods that are by example only. See the project notes as displayed and documented therein, for specific details.
The “Stereoscopic IPD” value is determined by the geometry being rendered and possibly adjustable by the user. Larger IPD values can cause eye strain and provide for an uncomfortable viewing experience.
Smaller IPD values can result in no stereo depth perception, provided objects are too distant.

User Mode Stereo Control API

HANDLE IWRSTEREO_Open (void)

Initiate session with the stereo control driver.

Returns:

HANDLE Result of open call or INVALID_HANDLE_VALUE upon fail.

Remarks:

If the function returns INVALID_FILE_HANDLE, use GetLastError() to retrieve the failure code:

- ERROR_INVALID_FUNCTION – the VR920 or Wrap920 has incorrect firmware to support stereo
- ERROR_DEV_NOT_EXIST – the VR920 or Wrap920 is not attached to the computer

void IWRSTEREO_Close (HANDLE *hDev*)

Terminate a session with the stereo control.

Parameters:

hDev handle returned in previous call to IWRSTEREO_Open.

BOOL IWRSTEREO_SetStereo (HANDLE *hDev*, BOOL *on*)

Set the VR920 or Wrap920 for monoscopic or stereoscopic operation.

Parameters:

hDev handle returned in previous call to IWRSTEREO_Open.
on 1 = stereoscopic mode; 0 = monoscopic mode

Returns:

TRUE = operation succeeded; FALSE = operation failed.

BYTE IWRSTEREO_WaitForAck (HANDLE *hDev, BOOL eye*)

Blocks until the VR920 “acknowledge” that is has seen the transition in the IWRSTEREO_SetLR call from left ot right eye (or right to left eye). The VR920 asserts this signal on the VSYNC following the transition.

Parameters:

hDev handle returned in previous call to IWRSTEREO_Open.
eye unused

Returns:

0 = left, 1 = right, 0xFF = error

BOOL IWRSTEREO_SetLR (HANDLE *hDev*, BOOL *eye*)

Select the eye the data in the current frame is meant for.

This function is typically called immediately after calling the Present method in DirectX. The PresentationInterval in the calling application should be set to D3DPRESENT_INTERVAL_ONE. VR920 only.

Parameters:

hDev handle returned in previous call to IWRSTEREO_Open.
eye = IWRSTEREO_RIGHT_EYE or IWRSTEREO_LEFT_EYE

Returns:

TRUE = operation succeeded; FALSE = operation failed

BOOL IWRSTEREO_GetVersion (PIWRVERSION ver)

Returns the version number for the IWRSTDRV.DLL.

This function can be called before IWRSTEREO_Open if needed.

Parameters:

ver pointer to a IWRVERSION structure that receives the version information

Returns:

TRUE = operation succeeded; FALSE = operation failed

IWRVERSION Structure

```
typedef struct tag_IWRVERSION {
        unsigned short DLLMajor;
        unsigned short DLLMinor;
        unsigned short DLLSubMinor;
        unsigned short DLLBuildNumber;
        char USBFirmwareMajor;
        char USBFirmwareMinor;
        char TrackerFirmwareMajor;
        char TrackerFirmwareMinor;
        char VideoFirmware;
} IWRVERSION, *PIWRVERSION;
```

- DLLMajor – major version of IWRSTDRV.DLL
- DLLMinor – minor version of IWRSTDRV.DLL
- DLLSubMinor – sub-minor version of IWRSTDRV.DLL
- DLLBuildNumber – build number of IWRSTDRV.DLL
- All other fields are unused

STEREO CONTROL TIMING

The following diagram illustrates how the VR920 responds to the stereo control USB commands.

A rendered frame is “presented” to the video hardware (using DirectX terminology). After the present, the application should use the IWRSTEREO_SetLR call to identify the eye for which the presented frame is destined. On the Vsync following the present and SetLR call, the video hardware starts transmitting the video data to the VR920 over the VGA port. The VR920 must detect a level transition on the SetLR signal in order to pass the video information arriving on the VGA port to the display and the frame buffer.

The terminology used in the diagram is as follows:
- Lx = Left frame x; e.g. L1 = left frame 1
- Rx = Right frame x; e.g. R2 = right frame 2
- Present = The sequence of rendered frames presented to the video card
- Vsync = the Vertical sync signal on the VGA port
- SetLR = the point at which the Left/Right control line needs to change levels
- Ssync = this is the acknowledge signal from the VR920 to the PC indicating the VR920 is latching the frame to the display and the frame buffers
- L_Display = the data appearing in the left display following a Vsync
- R_Display = the data appearing in the right display following a Vsync