Wavit™ Programmers Manual

Document Revision

Rev.	*Date*	*Changed by*	*Comments*
1.6	May 5, 2007	AGJ	Cleaned up for publishing
1.7	May10, 2007	AGJ	Added return error
1.8	May 16, 2007	AGJ	Added SetRelativeMode500
1.9	May 22, 2007	AGJ	Removed SetRelative and Absolute and replaced with SmartPoint, DirectPoint, and GamePoint.
2.0	July 8, 2007	AGJ	Added WI_DistanceScaling, WI_SetCalDistance,WI_GetCalDistance, WI_SetDmin, WI_GetDmin. WI_GetCalScreen, SetCalScreen
2.1	Aug. 13, 2007	AGJ	Added Registry Entries
2.2	Aug 13, 2008	AGJ	Added TalknPoint features
2.4	June 20, 2009	AGJ	Added HID

Note: Content is subject to change without notice.

Wavit™ API

About the Wavit

The Wavit™ 3D remote control & POD allow a user to remotely control all aspect of a PC (or game console). By offering direct absolute pointing, a user can point to icons on a screen or react instantly to point-and-shoot scenes in games. The Wavit™ also provides information about the absolute twist angle of the remote (a.k.a. the roll) and its distance from the screen. These can, for example, be used for turning a remote volume knob or zooming. Other optional degrees of freedom are the angular location of the user from the POD, which can be used to track for example the users' side-to-side motion.

The position sensing is based on optical tracking and as such requires direct line-of-sight between the remote and the POD (Photonic Origin Designator).

The Wavit remote includes a small vibration motor which can be controlled to deliver precise small or strong vibrations, primarily for allow the "feel-the-screen and icons" experience.

Up to 4 Wavits can work with a single POD. Each unit can be individually addressed through the API to read their coordinates or to send vibration commands, for example. When a Wavit is first turned on, it will look to communicate with the nearest POD or the one it most recently communicated with.

Up to 16 different PODs can be used within a ~100ft range, without any interference by setting the PODs to different frequency channels. When a POD is first turned on, it will look for the channel with the lowest RF interference.

The Wavit also allows for controlling all Infrared (IR) devices, such as TVs, DVD etc. The POD can learn practically any IR command and can on-demand blast codes into the room. There is also an IR blaster jack that will permit the use of IR blaster cables that can be directed specifically inside cabinets to control out-of-sight devices.

The Wavit is wireless (2.4GHz) and works on two AA batteries.

There are two different PODs. The original POD which requires the installation of drivers, and the new Wavit USB HID POD which is a plug-and-play device which gets recognized as 5 different devices: 1) HID Keyboard, 2) HID mouse, 3) Multimedia controller, 4) Audio-Class wireless microphone, and 5) a Wavit HID device (vendor specific HID).
The API contains some HID specific commands that only apply to the HID device.
Also, it should be noted that the original POD is set to communicate at 115200kbps. If it has been modified by ThinkOptics to be a "Sound POD" then the baudrate in the

Wavit.xml file needs to be changed to 500,000.

System Requirements

Operating System: Windows XP, Windows MCE2005, Macintosh OS-X, Linux 2.4+

CPU: Pentium V or M processor minimum. Core. Core2 Duo. Apple G5.

Free hard drive space: 100MB for WavitMENU program. 1.5 MB for core API.

How to purchase

Website at www.thinkoptics.com. Go to the shop tab. Also www.wavit.com

Technical Support

If you have any problems with ThinkOptics' Wavit you can contact our tech support by emailing.

mailto:support@thinkoptics.com

Please, make sure you tell us as much information as you can about the problem you are experiencing, including any error or warning messages that may have been displayed.

Inform us about the following:

• Wavit Version

• Operation system version;

• Description of your problem (as much information as possible to help us reproduce the problem).

We'll try to help you as fast as possible, usually in one or two business days, but due to limited resources we cannot guarantee immediate response.

If you have any comments or suggestions for the next releases, please feel free to post them to us, also at mailto:support@thinkoptics.com

End-User License Agreement (EULA)

Getting Started with Wavit™

Software Installation

See the ThinkOptics ReadMe file.

Windows:

Installation:

For first time installation, follow these steps:

1. Insert the Installation CD *BEFORE* you insert the POD USB cable.
2. Select "*INSTALL"*
   (if the CD does not auto-start upon insertion, run autorun.exe in the install folder)
3. After installation is complete (this may take 10 minutes), you will be asked to *REBOOT* the computer.
4. When the computer finishes rebooting, you may *NOW INSERT* the POD's USB cable. Allow Windows to auto-recognize the *ThinkOptics WavIt*™ Device (this will take about 15 seconds).

   ---- Your installation in now complete! ----

5. To start running the WavItMENU, select Windows **"Start"** Menu and select "Start_WavItMENU" under the "All Programs" menu. .

Note: Should you wish to uninstall everything, use Windows *"Add/Remove Programs"* in the Control Panel menu.

Operational Notes:

a. The POD should be placed near the TV screen, preferably right above or below the screen. The Wavit™ tracking system relies on IR light, so please keep strong IR sources away from the near vicinity of the POD. This includes PCs with IrDA ports, live candles, or bare incandescent light bulbs.

b. The Wavit Handset will fall asleep if not used for more than 5 seconds. Push any button on the remote to wake it up.

c. When you insert the Wavit™ POD the first time it will blink 8 times as it finds the best frequency channel to transmit on. When you then turn on the Wavit™ Remote for the first time, it will blink 8 times as it looks for the nearest POD to associate to. If it doesn't find a POD it will blink 4 times slowly and go back to sleep. Once the link has been established it will no-longer need to look for the POD. You will see that there is an active RF link when the PODs visible LED starts blinking fast.

Files:

The *Wavit Installation program* has installed the following files into the different directories (for the PC-version).

Desktop:

Start_WavItMENU.exe - The main executable that starts up the WavIt.exe program (for cursor action) and the button mapping GUI.

Program files/WavitMenuNNN:

WavItDev.dll	- used by the API to allow third-party communication with the Wavit system.
SiUSBXp.dll	- USB driver.
WavitService.exe	- The Windows service program that starts and stops Wavit.exe and WavitDrawBMP.exe when the PC is rebooted or the USB is unplugged and re-plugged.
Wavit.exe	- The Wavit executable that moves the cursor. This Needs to be running for the API calls to work. It will be automatically started by WavitService.exe.
WavitDrawBMP.exe	- This program needs to be running to show multiple icons on the screen when more Wavit users are on. It will be automatically started by WavitService.
API documentation	- Complete description (Application Programming Interface) for how to interface with the Wavit™ system.
Header files	- List of available functions for interfacing to the Wavit™.
Lib files	- Needed by some programming languages for Interfacing
Preinstaller/Uninstaller	- ThinkOptics Wavit MC Driver installation

C:\Documents and Settings\All Users\Application Data\ThinkOptics\WavitData

Support text files	- These files contain the button configurations and the stored IR codes: TOButtonsText, TOButtonsCodes, TOButtonsIRCodes, TOButtonsIRCommands.

C:\Documents and Settings\All Users\Application Data\ThinkOptics

Wavit.xml	- file containing initialization values. If this is edited by hand Wavit.exe needs to be stopped and restarted for the changes to take effect. This is easily achieved by unplugging and replugging the USB cable.

Example programs can be found on the installation CD or on www.thinkoptics.com/Downloads.html

Labview8.5:

Various sample programs	- Example programs and a Userlib for quickly programming the Wavit™ system in Labview 8.5

C#:

Various sample programs	- Example programs in C# for quickly programming the Wavit™ system.

Hardware installation

For best results, the POD should be placed right on top or beneath the display screen. It should be connected by USB cable to the PC. If the retractable cable is not long enough, USB extension cables can be used, provided they are <10ft. Active USB repeaters/booster cables can also be used to extend that distance to up to 40ft for a single booster, for example.

The device should work with any type of screen (DLP, CRT, LCOS, front projection, Plasma, LCD etc.) and will track well provided a few simple safeguards are made:

The optical tracking system is based on IR light and as such it is possible for the system to be confused by strong IR light sources in the near vicinity of the screen. These should be minimized as they may result in cursor freeze or erratic jumping. This includes unshielded incandescent lamps. Some new digital TVs (although this is very rare) may have IrDa ports on the front for a wireless keyboard for example. This may need to be covered up with black tape. There are also some rear speaker systems that communicate via IR. Such a transmitter should be placed a bit away from the TV so as to be outside the field-of-view of the handset.

Your First Software Commands

When the WavIt.exe (or WavItDevice.app for the Mac, or Wavit executable on Linux) program is running, the user can communicate with the Wavit through a series of simple .dll calls (or .framework calls for Mac, or .so calls for Linux).

The basic operating principle is one of “poling” the variables.

Initialize:

First you need to set up the communication port. This is done by calling:

```
WAVITDEV_API int WI_Initialize()
```

From then on you can call any of the API functions to get or write information.

To get the position variables:
You simply call the following routines at any time to get instantaneous information about the (x,y)-position to which you are pointing on the screen, the roll angle of the remote, and the remote's distance from the screen.

```
WAVITDEV_API double WI_GetX(int UserNo);
WAVITDEV_API double WI_GetY(int UserNo);
WAVITDEV_API double WI_GetRoll(int UserNo);
WAVITDEV_API double WI_GetDistance(int UserNo);
```

where UserNo is the number of the WavIt Handset, 0-3. For a single user, UserNo=0.

To get the button presses:
The following command allows you to read the position read the button presses.

```
WAVITDEV_API int WI_GetButtons(bool *nw, bool *n, bool *ne, bool *w, bool *c, bool *e, bool *sw,
bool *s, bool *se, bool *to, bool *ts, bool *powerlow, int userNo);
```

To turn mouse movement on/off:
The Wavit program supports up to 4 simultaneous users. It also allows one of the users to have full control of the movements of the mouse cursor. We use a token sharing approach, whereby any user can take control of the mouse by pressing the central "OK" button on their remote.

The mouse movement can also be enabled/disabled with the following command:

```
WAVITDEV_API int WI_SetActivateCursor(int value);
```

where value=1 will turn the mouse movement on.

Anti-shake:
The Wavit remote has a built-in "anti-shake" algorithm that can help stabilize an unsteady hand when the user is slowing down his motion in order to hit a target. Currently this algorithm is adjusted with two parameters: 1) the speed threshold and 2) the viscosity. To turn off the "anti-shake" feature, call:

```
WAVITDEV_API int WI_SetXYCounterMax(int value);
```

with value=1.

The Settings:
Finally, we note that the several preference values and flags are set in the Wavit.xml file. They can also be changed by calling the appropriate API commands described in this document.

– <Thinkoptics>

```
- <Settings>
  <D0>70.000000</D0>
  <Mx>5.000000</Mx>
  <Bx>-250.000000</Bx>
  <My>5.000000</My>
  <By>-340.000000</By>
  <CPixel>1.000000</CPixel>
  <DistanceThresh>0.040000</DistanceThresh>
  <StabThresh>7.000000</StabThresh>
  <CThetaX>1.000000</CThetaX>
  <CalDistance>3.100000</CalDistance>
  <Dmin>3.200000</Dmin>
  <DistanceCounterMax>12.000000</DistanceCounterMax>
  <XYCounterMax>3</XYCounterMax>
  <RollCounterMax>5</RollCounterMax>
  <SubframeInterpolation>3</SubframeInterpolation>
  <RotationCompensation>1</RotationCompensation>
  <WavItMenu>0</WavItMenu>
  <Sleepingfactor>30</Sleepingfactor>
  <PowerThresh>1</PowerThresh>
  <ClickStill>12</ClickStill>
  <XcalScreen>1280</XcalScreen>
  <YcalScreen>768</YcalScreen>
  <BaudRate>115200</BaudRate>
  <DistanceScaling>0</DistanceScaling>
  <DeviceName>/dev/ttyUSB0</DeviceName>
  <WavitMenuPath>C:\Program
Files\WavItMENU112\Start_WavItMENU.exe</WavitMenuPath>
  </Settings>
  </Thinkoptics>
```


The rest of this document will go into more detail on all aspects of the communicating with the Wavit remote, including sending commands to the built-in vibration motor, or learning and blasting infra-red commands for controlling all IR devices.

The Wavit Reference

The following set of commands constitutes the API for the ThinkOptics Wavit 3D Controller.

Of these, the primary ones of interest to UI development are the ones that relate to "poling" the degrees of freedom or the button presses. (The newest version of the software also sends Windows Messages, so event based handling is enabled. Documentation for this is pending).

Any custom UI will also be responsible for sending and receiving IR commands or sending motor action commands.

All other commands can mostly be ignored, i.e. no set up is required. Some of these commands will be needed later for recalibration or to set some stabilization preferences.

```
//Setup:

WAVITDEV_API int WI_Initialize();
WAVITDEV_API int WI_GetBaudRate();
WAVITDEV_API int WI_SetBaudRate(int value);
WAVITDEV_API unsigned char WI_GetStopBit();
WAVITDEV_API unsigned long WI_GetSleepingFactor();
WAVITDEV_API int WI_SetSleepingFactor(unsigned long value);
WAVITDEV_API double WI_GetD0();
WAVITDEV_API int WI_SetD0(double value);
WAVITDEV_API int WI_MouseClick(int value);
WAVITDEV_API int WI_QuickClick(int value);

WAVITDEV_API double WI_GetMx();
WAVITDEV_API int WI_SetMx(double value);
WAVITDEV_API double WI_GetBx();
WAVITDEV_API int WI_SetBx(double);
WAVITDEV_API double WI_GetMy();
WAVITDEV_API int WI_SetMy(double value);
WAVITDEV_API double WI_GetBy();
WAVITDEV_API int WI_SetBy(double value);
WAVITDEV_API int WI_GetActivateCursor();
WAVITDEV_API int WI_SetActivateCursor(int value);
WAVITDEV_API double WI_GetCpixel();
WAVITDEV_API int WI_SetCpixel(double value);
WAVITDEV_API int WI_GetCalScreen(int *x, int *y);
WAVITDEV_API int WI_SetCalScreen(int x, int y);
WAVITDEV_API int WI_StopMotor();

//Main routines:
#ifdef _WIN32
WAVITDEV_API int WI_GetLatestError();
WAVITDEV_API int WI_GetErrorString(char *strErrorMsg);
#endif
WAVITDEV_API int WI_GetCurrentUser();
WAVITDEV_API double WI_GetX(int userNo);
```

```
WAVITDEV_API double WI_GetY(int userNo);
WAVITDEV_API double WI_GetRawX(int userNo);
WAVITDEV_API double WI_GetRawY(int userNo);
WAVITDEV_API int WI_GetRawData(int *,unsigned char *);
WAVITDEV_API int WI_GetDataPacketInterval(int *, int);
WAVITDEV_API double WI_GetRoll(int userNo);
WAVITDEV_API double WI_GetDistance(int userNo);
WAVITDEV_API double WI_GetThetaX(int userNo);
WAVITDEV_API int WI_SetThetaX(float value);
WAVITDEV_API int WI_StopDevice();
WAVITDEV_API int WI_GetPower1(int userNo);
WAVITDEV_API int WI_GetPower2(int userNo);
WAVITDEV_API int WI_GetButton(int);
WAVITDEV_API int WI_GetXYCounterMax();
WAVITDEV_API int WI_SetXYCounterMax(int value);
WAVITDEV_API int WI_GetDistanceCounterMax();
WAVITDEV_API int WI_SetDistanceCounterMax(int value);
WAVITDEV_API int WI_GetXYCounter(int userNo);
WAVITDEV_API int WI_GetSubframeInterpolation();
WAVITDEV_API int WI_SetSubframeInterpolation(int value);
WAVITDEV_API int WI_GetRotationCompensation();
WAVITDEV_API int WI_SetRotationCompensation(int value);
WAVITDEV_API int WI_GetClickStill();
WAVITDEV_API int WI_SetClickStill(int value);
WAVITDEV_API double WI_GetStabThresh();
WAVITDEV_API int WI_SetStabThresh(double value);
WAVITDEV_API int WI_SetStabilize(int value, int userNo);
WAVITDEV_API int WI_GetStabilize(int userNo);
WAVITDEV_API int WI_SetCalibrateMode(int value);
WAVITDEV_API int WI_SetSleep(int value);
WAVITDEV_API int WI_GetSleep();
WAVITDEV_API double WI_GetAverageRoll(int value, int userNo);
WAVITDEV_API double WI_GetAverageDistance(int value, int userNo);
WAVITDEV_API double WI_GetAverageThetaX(int value, int userNo);
WAVITDEV_API int WI_GetButtons(bool *nw, bool *n, bool *ne, bool *w,
bool *c, bool *e, bool *sw, bool *s, bool *se, bool *to, bool *ts, bool
*powerlow, int userNo);
WAVITDEV_API int WI_GetButtonsExt(bool *nw, bool *n, bool *ne, bool *w,
bool *c, bool *e, bool *sw, bool *s, bool *se, bool *to, bool *vol,
bool *Mic, bool *ts, bool *powerlow, int userNo);
WAVITDEV_API int WI_GetClickWheel(bool *Menu, bool *Next, bool *Play,
bool *Previous, bool *Select, int *CW, bool *ts, bool *powerlow, int
userNo);
WAVITDEV_API int WI_GetVirtualButtons(bool *bck, bool *fw, bool *in,
bool *out, bool *below, bool *top, bool *left, bool *right, int
userNo);
WAVITDEV_API int WI_GetIRActive();
WAVITDEV_API int WI_SetDirectPoint();
WAVITDEV_API int WI_SetGamePoint();
WAVITDEV_API int WI_SetSmartPoint();
WAVITDEV_API int WI_MoveAtBoundary(int value);
WAVITDEV_API int WI_Calibration();
WAVITDEV_API int WI_SmartTracking(bool bValue);
WAVITDEV_API int WI_SetDistanceScaling(bool value);
WAVITDEV_API int WI_SetCalDistance(double);
WAVITDEV_API int WI_GetCalDistance(double *);
WAVITDEV_API  int WI_SetDmin(double);
```

```
WAVITDEV_API int WI_GetDmin(double *);

//Send commands

WAVITDEV_API int WI_SendIRBlast();
WAVITDEV_API int WI_LearnCode();
WAVITDEV_API int WI_StartMotor(int power, int time, int userNo);
WAVITDEV_API int WI_GetLastError();
WAVITDEV_API int WI_GetIRCode(unsigned char *, int *);
WAVITDEV_API int WI_SendIRCode(unsigned char *, int);

WAVITDEV_API int WI_SetExposure(int value);
WAVITDEV_API int WI_SetGain(int value);
WAVITDEV_API int WI_SetThreshold(int value);

WAVITDEV_API int WI_SetDrawBitmaps(int value);
WAVITDEV_API int WI_GetDrawBitmaps(int *value);

WAVITDEV_API int WI_GetHandheldCopyright(char *);
WAVITDEV_API int WI_GetHandheldVersion(char *);
WAVITDEV_API int WI_GetHandheldDate(char *);
WAVITDEV_API int WI_GetPODCopyright(char *);
WAVITDEV_API int WI_GetPODVersion(char *);
WAVITDEV_API int WI_GetPODDate(char *);
WAVITDEV_API int WI_GetHandsetAssignmentTable(char *);

WAVITDEV_API int WI_WavItInfo(char *, char *, char *, char *, char *,
char *, char *);
WAVITDEV_API int WI_Debug(int value);
WAVITDEV_API int WI_Image(int *, int *, int *, unsigned char * );
WAVITDEV_API int WI_GetRFReceiveStrength(unsigned char *buffer);
WAVITDEV_API int WI_GetPODChannel(unsigned char *buffer);
WAVITDEV_API int WI_EnergyMap(unsigned char *buffer);
WAVITDEV_API int WI_SetPODChannel(int Channel);
WAVITDEV_API int WI_SendKeyEvent(int input_code1, int input_code2, int
type, unsigned int delay);
WAVITDEV_API int WI_ProgramID( unsigned char ID_1, unsigned char ID_2,
unsigned char ID_3, unsigned char ID_4, unsigned char SLOT);
WAVITDEV_API int WI_ClearHAT(void);
WAVITDEV_API int WI_GetRawMicArray(unsigned char *SoundPacket);
WAVITDEV_API int WI_GetSoundCount();
WAVITDEV_API int WI_GetMicArray(short *SoundPacket, int count, int
reset);
WAVITDEV_API int WI_SetWavitMenu(bool);
WAVITDEV_API int WI_GetWavitMenu(bool *);
WAVITDEV_API int WI_SetWavitMenuPath(char *path);
WAVITDEV_API int WI_GetWavitMenuPath(char *path);
WAVITDEV_API int WI_SetCenterSWbutton(bool value);
WAVITDEV_API int WI_GetCenterSWbutton(bool *value);

WAVITDEV_API int WI_SetHIDMouse(int value);
WAVITDEV_API int WI_GetHIDDeviceMode(bool *value);
WAVITDEV_API int WI_SetHID_Cal(float xscale, float yscale, float xoff,
float yoff, float xclip, float yclip);
WAVITDEV_API int WI_GetHID_Cal(float *xscale, float *yscale, float
*xoff, float *yoff, float *xclip, float *yclip);
```

```
WAVITDEV_API int WI_SetHID_Antishake(unsigned int thresh, unsigned char
vis);
WAVITDEV_API int WI_GetHID_Antishake(unsigned int *thresh, unsigned
char *vis);
WAVITDEV_API int WI_SetHID_Default();
```

WI_Initialize

Syntax

WAVITDEV_API int WI_Initialize();

Description

This command should be issued BEFORE any other commands. It initializes the WavIt communication with the DLL. Note that WavIt.exe *MUST* be running for all the WavItDev.dll calls to work. If WavIt.exe is not running, this commands will return "-1", indicating an error has occurred. This error message can then be used as a flag to start up WavIt.exe in case it isn't running. Use WI_GetLatestError() to identify the error.

Return Type

Type	Description
INT	Return 1 on successful completion. -1 if no WavIt.exe is running or USB not connected.

Parameter

Type	Parameter	Description
VOID		

WI_GetLatestError

Syntax

WAVITDEV_API int WI_GetLatestError();

Description

Returns the latest logged error.

Error 1: Error connecting to Device - Check that USB cable is connected.

Error 2: Error connecting to WavIt - Please make sure that WavIt.exe is running.

Error 3: The slot number specified is wrong. Possible values are 0,1,2,3

Error 4: Internal error - unable to map memory

Use WI_GetErrorString() to read out the actual error messages.

Return Type

Type	Description
INT	Return error number.

Parameter

Type	Parameter	Description
VOID		

WI_GetErrorString

Syntax

WAVITDEV_API int WI_GetErrorString(char *strErrorMsg);

Description

Returns the latest logged error description. See also WI_GetLatestError.

Return Type

Type	Description
INT	Return error number.

Parameter

Type	Parameter	Description
CHAR	strErrorMsg	Gives the string error message. Error 1: Error connecting to Device - Check that USB cable is connected. Error 2: Error connecting to WavIt - Please make sure that WavIt.exe is running. Error 3: The slot number specified is wrong. Possible values are 0,1,2,3 Error 4: Internal error - unable to map memory

WI_GetX

Syntax
WAVITDEV_API double WI_GetX(int userNo)

Description
Returns the x-coordinate of the location on the screen to which the handset is pointed. (See also WI_GetY).

TIP : Use WI_Calibrate to calibrate the pointing. This will adjust the parameters mx, bx, my, by. Use WI_SetXYCounterMax to adjust the viscosity parameter of "Anti-Shake" function. Use SetStabThresh to adjust the stabilization threshold parameter of the "Anti-Shake" function. Use GetRawX to get the unstabilized, uncalibrated position on the handset sensor.

Return Type

Type	Description
DOUBLE	X-coordinate on screen

Parameter

Type	Parameter	Description
INT	userNo	The user (or Handset) number (0-3).

WI_GetY

Syntax
WAVITDEV_API double WI_GetY(int userNo)

Description
Returns the y-coordinate of the location on the screen to which the handset is pointed. (See also WI_GetX).

TIP : Use WI_Calibrate to calibrate the pointing. This will adjust the parameters mx, bx, my, by. Use WI_SetXYCounterMax to adjust the viscosity parameter of "Anti-Shake" function. Use SetStabThresh to adjust the stabilization threshold parameter of the "Anti-Shake" function. Use GetRawY to get the unstabilized, uncalibrated position on the handset sensor.

Return Type

Type	Description
DOUBLE	Y-coordinate on screen

Parameter

Type	Parameter	Description
INT	userNo	The user (or Handset) number (0-3).

WI_GetRoll

Syntax
WAVITDEV_API double WI_Roll(int userNo)

Description
Returns the handset's instantaneous handset roll angle. (See also WI_GetAverageRoll).

Return Type

Type	Description
DOUBLE	Roll coordinate in radians. 0 corresponds to horizontal. Rolling counterclockwise is negative, clockwise is positive.

Parameter

Type	Parameter	Description
INT	userNo	The user (or Handset) number (0-3).

WI_GetDistance

Syntax
WAVITDEV_API double WI_GetDistance(int userNo)

Description
Returns the handset's instantaneous distance from the POD. (See also WI_GetAverageDistance)

TIP : The calibration of the distance is adjusted with the command: WI_SetDO. Depending on the Version of the WavIt, this distance may not be totally correct as you move away from the front of the POD.

Return Type

Type	Description
DOUBLE	Distance in meters from the screen.

Parameter

Type	Parameter	Description
INT	userNo	The user (or Handset) number (0-3).

WI_GetThetaX

Syntax

WAVITDEV_API double WI_GetThetaX(int userNo)

Description

Returns the handset's location in the horizontal plane, also known as the azimuthal angle. (See also WI_GetAverageThetaX for the averaged data). CThetaX in the WaveIt.ini file is an angle scaling factor. The default value is 1.0.

Parameter

Type	Parameter	Description
INT	userNo	The user (or Handset) number (0-3).

Return Type

Type	Description
DOUBLE	Angle, in horizontal plane, at which handset is located. This is also known as the Azimuthal angle.

WI_GetAverageRoll

Syntax

WAVITDEV_API double WI_GetAverageRoll(int value, int userNo)

Description

Returns the handset's running-averaged roll angle. (See also WI_GetRoll).

Parameter

Type	Parameter	Description
INT	AverageIndex	The number of frames over which the roll variable is being averaged. Allowed values are 1-16. *Example:* WI_GetAverageRoll(1, userNo) will give the same result at WI_GetRoll(userNo).
INT	userNo	The user (or Handset) number (0-3).

Return Type

Type	Description
DOUBLE	Roll angle of Handset, in radians. 0 corresponds to horizontal, negative to counter-clockwise, and positive numbers to clockwise rotation.

WI_GetAverageDistance

Syntax
WAVITDEV_API double WI_GetAverageDistance(int value, int userNo)

Description
Returns the handset's running-averaged distance from screen. (See also WI_GetDistance).

Parameter

Type	Parameter	Description
INT	AverageIndex	The number of frames over which the distance variable is being averaged. Allowed values are 1-16. *Example:* WI_GetAverageDistance(1, userNo) will give the same result at WI_GetDistance(userNo).
INT	userNo	The user (or Handset) number (0-3).

Return Type

Type	Description
DOUBLE	Distance between Handset and POD, in meters.

WI_GetAverageThetaX

Syntax

WAVITDEV_API double WI_GetAverageThetaX(int value, int userNo)

Description

Returns the Handset's running-averaged location in the horizontal plane, also known as the azimuthal angle. (See also WI_GetThetaX for the un-averaged data). CThetaX in the WaveIt.ini file is an angle scaling factor. The default value is 1.0.

Parameter

Type	Parameter	Description
INT	Value	The number of frames over which the distance variable is being averaged. Allowed values are 1-16. *Example:* WI_GetAverageThetaX(1, userNo) will give the same result at WI_GetThetaX(userNo).
INT	userNo	The user (or Handset) number (0-3).

Return Type

Type	Description
DOUBLE	Angle, in horizontal plane, at which handset is located. This is also known as the Azimuthal angle.

WI_GetButtons

Syntax

WAVITDEV_API int WI_GetButtons(bool *nw, bool *n, bool *ne, bool *w, bool *c, bool *e, bool *sw, bool *s, bool *se, bool *to, bool *ts, bool *powerlow, int userNo)

Description

Returns the button presses on the Handset. The 10 buttons are arranged in a 9+1 button cluster, where the 10th lone button is Menu button, or the "TO" button. Among the 9 buttons, they are named according to their location on a compass. The central button is called "c". This function also returns the handset user number (0-3) and the low-battery indicator, "powerlow". With the exception of "userNo" all parameters are pointer that will return the Boolean state of the button pressed, or the state of the battery. "TS" indicates whether the handset is correctly targeting the POD spots (see WI_SmartTracking).

Parameter

Type	Parameter	Description
BOOL	Nw	The state of the North West button. True(1)=depressed.
BOOL	N	The state of the North button. True(1)=depressed.
BOOL	Ne	The state of the North East button. True(1)=depressed.
BOOL	W	The state of the West button. True(1)=depressed.
BOOL	C	The central "OK" button. True(1)=depressed.
BOOL	E	The state of the East button. True(1)=depressed.
BOOL	Sw	The state of the South West button. True(1)=depressed.
BOOL	S	The state of the South button. True(1)=depressed.
BOOL	Se	The state of the South East button. True(1)=depressed.
BOOL	To	The state of the "TO" a.k.a. "Menu" button, True(1)=depressed
BOOL	Ts	The state of the "Ts" "Targeting Spots" button. True(1) means that it is properly tracking to a confirmed POD.
BOOL	Powerlow	Indicates whether the battery power is low (0)
UINT	User	Handset user number (0-3). This is an input.

WI_GetVirtualButtons

Syntax

WAVITDEV_API int WI_GetVirtualButtons(bool *bck, bool *fw, bool *in, bool *out, bool *below, bool *top, bool *left, bool *right, int userNo);

Description

Returns the virtual button presses on the Handset. Boolean variables are returned for various check conditions, such as “Is handset pointed above the screen boundaries?” or “Is handset rotated >20 deg counter clockwise?”.

Parameter

Type	Parameter	Description
BOOL	Bck	Indicates whether handset is rotated <-20 deg. (counter-clockwise).
BOOL	Fw	Indicates whether handset is rotated <-20 deg. (counter-clockwise).
BOOL	In	Indicates whether handset is being pushed fast toward the screen. The speed threshold is set by the DistanceThresh parameter in the WavIt.ini file. A normal value is 0.04.
BOOL	Out	Indicates whether handset is being pushed fast away from the screen. The speed threshold is set by the DistanceThresh parameter in the WavIt.ini file. A normal value is 0.04.
BOOL	Below	Indicates whether handset is pointed below the screen.
BOOL	Top	Indicates whether handset is pointed above the screen.
BOOL	Left	Indicates whether handset is pointed to the left of the screen.
BOOL	Right	Indicates whether handset is pointed to the right of the screen.
INT	UserNo	Handset user number (0-3). This is an input.

WI_GetClickWheel

Syntax

WAVITDEV_API int WI_GetClickWheel(bool *Menu, bool *Next, bool *Play, bool *Previous, bool *Select, int *CW, bool *ts, bool *powerlow, int userNo);

Description

Returns the button presses on the Handset that has a Click-Wheel or Touch-Wheel. One possible arrangement of a touch-wheel is to have 5 buttons arranged in a cluster – Menu (North), Next (East), Play (South), Previous (West), and a Select button (Center). Some version will also have a "touch sensor" flag, ts, which indicates whether any buttons are pressed and valid. Finally, there is generally a linear or circular touch sensor with a graduated scale between 0 and 100. Another output is the "low-battery" indicator, "powerlow". With the exception of "userNo" which can take a value from 0-3, all parameters are pointers that will return the requested values.

Parameter

Type	Parameter	Description
BOOL	Menu	The top "menu" button. True(1)=depressed.
BOOL	Next	The state of the East button. True(1)=depressed.
BOOL	Play	The state of the South button. True(1)=depressed.
BOOL	Previous	The state of the West button. True(1)=depressed.
INT	CW	The reading of the linear or circular touch sensor.
BOOL	Ts	The state of the "Ts" "Touch Sensor" button, True(1)=depressed
BOOL	Powerlow	Indicates whether the battery power is low and they need to be replaced (0).
INT	UserNo	Handset user number (0-3). This is an input.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

WI_StartMotor

Syntax

WAVITDEV_API int WI_StartMotor(int power, int time, int userNo);

Description

Sends the control to the motor to turn on at a prescribed power and for a prescribed length of time.

Parameter

Type	Parameter	Description
INT	Power	The power can be set from at 25, 50, 75, and 100%. The argument needs to be 25, 50, 75, or 100.
INT	Time	The duration of the vibration can be set in ms, from 0 to 2268ms.
INT	userNo	The user (or Handset) number (0-3).

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

WI_GetRawX

Syntax
WAVITDEV_API double WI_GetRawX(int userNo)

Description
Returns the x-coordinate of the location on the screen to which the handset is pointed. This is the un-calibrated version of WI_GetX (See also WI_GetRawY).

Return Type

Type	Description
DOUBLE	Un-calibrated X-coordinate on screen

Parameter

Type	Parameter	Description
INT	userNo	The user (or Handset) number (0-3).

WI_GetRawY

Syntax
WAVITDEV_API double WI_GetRawY(int userNo)

Description
Returns the y-coordinate of the location on the screen to which the handset is pointed. This is the un-calibrated version of WI_GetY (See also WI_GetRawX).

Return Type

Type	Description
DOUBLE	Un-calibrated Y-coordinate on screen

Parameter

Type	Parameter	Description
INT	userNo	The user (or Handset) number (0-3).

WI_GetPower1

Syntax
WAVITDEV_API int WI_GetPower1(int userNo)

Description
Returns the power of LED 1 on the POD as seen by the handset. (See also WI_GetPower2).

Return Type

Type	Description
INT	The power of LED1

Parameter

Type	Parameter	Description
INT	userNo	The user (or Handset) number (0-3).

WI_GetPower2

Syntax
WAVITDEV_API int WI_GetPower2(int userNo)

Description
Returns the power of LED 2 on the POD as seen by the handset. (See also WI_GetPower1).

Return Type

Type	Description
INT	The power of LED2 on the POD

Parameter

Type	Parameter	Description
INT	userNo	The user (or Handset) number (0-3).

WI_GetXYCounterMax

Syntax
WAVITDEV_API int WI_GetXYCounterMax()

Description
Returns the XYCounterMax constant which determines the MAXIMUM VISCOSITY of the motion of the cursor when stabilization (ANTI-SHAKE) is active. This constant is set in the WavIt.ini file and can also be set programmatically using WI_SetXYCounterMax. The larger the number, the more dynamic averaging is done when the cursor is moved very slowly.

TIP : Use WI_SetStabThresh to adjust the stabilization THRESHOLD parameter of the "Anti-Shake". All users share the same anti-shake settings.

Return Type

Type	Description
INT	XYCounterMax (1-20) where a high number signifies very strong active stabilization.

Parameter

Type	Parameter	Description
VOID		

WI_SetXYCounterMax

Syntax
WAVITDEV_API int WI_SetXYCounterMax(int Value)

Description
Sets the XYCounterMax constant which determines the MAXIMUM VISCOSITY of the motion of the cursor when stabilization (ANTI-SHAKE) is active. This constant can also be set in the WavIt.ini file. The larger the number, the more dynamic averaging is done when the cursor is moved very slowly.

TIP : Use WI_SetStabThresh to adjust the stabilization THRESHOLD parameter of the "Anti-Shake". All users share the same anti-shake settings.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
INT	Value	XYCounterMax (1-20) where a high number signifies very strong active stabilization. For no stabilization, set XYCounterMax=1.

WI_GetXYCounter

Syntax
WAVITDEV_API int WI_GetXYCounter(int userNo)

Description
Returns the instantaneous XYCounter which describes how much stabilization is currently active. The slower the cursor moves the larger this number will be, until it maxes out at XYCounterMax. See also GetXYCounterMax.

Return Type

Type	Description
INT	XYCounter (1 to XYCounterMax) where a higher number signifies stronger active stabilization.

Parameter

Type	Parameter	Description
NONE	userNo	The user (or Handset) number (0-3).

WI_GetStabThresh

Syntax
WAVITDEV_API double WI_GetStabThresh()

Description
Returns the StabThresh parameter which determines the cursor-speed threshold below which the stabilization (or ANTI-SHAKE) algorithm kicks in. (See also GetXYCounter which describes how much stabilization is currently active.)

Return Type

Type	Description
DOUBLE	StabThresh parameter with a value generally between 0 and 4. 0 indicates that stabilization will never kick in. For normal use, this parameter is generally set at ~2-3.

Parameter

Type	Parameter	Description
VOID		

WI_SetStabThresh

Syntax
WAVITDEV_API int WI_GetStabThresh(double value)

Description
Sets the StabThresh parameter which determines the cursor-speed threshold below which the stabilization (or ANTI-SHAKE) algorithm kicks in. (See also SetXYCounterMax which describes the Maximum Viscosity of the stabilization algorithm)

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
DOUBLE	value	StabThresh parameter. Value should generally be between 0 and 4. 0 indicates that stabilization will never kick in. For normal use, this parameter should generally be set to ~2-3, but for fast gaming action 0.5 or less may be preferred.

WI_GetClickStill

Syntax
WAVITDEV_API int WI_GetStabThresh()

Description
Returns the ClickStill parameter which describes the number of 35ms increments that the mouse cursor will stop moving after the central OK (select) button has been pressed. This is used to enable the Windows "double-click" function which won't respond if the cursor has moved too far from its initial location before experiencing the second click.

Return Type

Type	Description
INT	ClickStill parameter. The default is 8, which means that the cursor "freezes" for 8x35ms before again continuing to be updated.

Parameter

Type	Parameter	Description
VOID		

WI_SetClickStill

Syntax
WAVITDEV_API int WI_GetStabThresh(int Value)

Description
Sets the ClickStill parameter which describes the number of 35ms increments that the mouse cursor will stop moving after the central OK (select) button has been pressed. This is used to enable the Windows "double-click" function which won't respond if the cursor has moved too far from its initial location before experiencing the second click.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
INT	Value	ClickStill parameter. The default is 8, which means that the cursor "freezes" for 8x35ms before again continuing to be updated.

WI_GetRFReceiveStrength

Syntax
WAVITDEV_API int WI_GetRFReceiveStrength(unsigned char *buffer)

Description
Returns the RF signal strength of the handset as seen by the POD. This call only works if there is only one handset active. If the signal strength falls to near 75, communication will be lost.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Unsigned char pointer	Buffer	The pointer to this buffer contains an array of 2 bytes, the RSSI and Correlation. The first byte is RSSI (Received Signal Strength Indicator.) The RSSI result is returned in dB. There is a -45 db offset and -128. If answer is 75, the actual dBm is 75-128-45 or -98dbm. This is the noise floor. The second byte is the Correlation. The most significant bit (bit 7) indicates if the CRC is ok. If it is set the CRC is ok. The remaining bits (6:0) is a value that indicates the correlation value. The bigger the better.

WI_PODChannel

Syntax
WAVITDEV_API int WI_PODChannel(unsigned char *buffer)

Description
Returns the POD frequency channel from 0 to 15.

Note: Upon initial power-on, the POD will search for the channel with least interference. The LED will blink 8 times before it has completed its search. See also WI_SetPodChannel, and WI_EnergyMap.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Unsigned char pointer	Buffer	A single byte which contains the current frequency channel of the POD (0-15).

WI_EnergyMap

Syntax
WAVITDEV_API int WI_EnergyMap(unsigned char *buffer)

Description
Returns a map of the energies in each of the 16 (0-15) available frequency channels, as seen by the POD upon power on. These are used by the POD to select the channel with the lowest RF interference.

Note: Upon initial power-on, the POD will search for the channel with least interference. The LED will blink 8 times before it has completed its search. See also WI_SetPODChannel and WI_PODChannel.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Unsigned char pointer	Buffer	An array of 16 bytes which contains the power-on snap-shot of the peak energies measured by the POD in each channel. Channel 0 is the first byte, channel 15 is the last byte.

WI_SetPODChannel

Syntax
WAVITDEV_API int WI_SetPODChannel(int Channel)

Description
Sets the POD frequency channel (0-15). When the channel is changed the POD also sends a "go to sleep" command to all handsets. Next time they turn on they will roam searching for the nearest POD to associate with. If an argument of 255 use used in this command, the POD will AUTOSCAN looking for the frequency channel with the lowest interference. This will also update the Energy Map (see WI_EnergyMap).

Note: Upon initial power-on or the WI_SetPODChannel(255) command, the POD will search for the channel with least interference. The LED will blink 8 times before it has completed its search. See also WI_PODChannel.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
INT	Channel	The channel number to which the POD should change (0-15). A value of 255 tells the POD to autoscan for the lowest interference channel.

WI_GetHandsetAssignmentTable

Syntax
WAVITDEV_API int WI_GetHandsetAssignmentTable(char *)

Description
Returns the handset assignment table (HAT) showing which 4 handsets are assigned to this POD.
Once this table fills up, no other handsets will be allowed to connect to this POD. The table can be cleared by holding down the button on top of the POD for 3 seconds. The POD will blink once to acknowledge.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Unsigned char pointer	HAT	Handset Assignment Table (HAT) for the POD, describing which handsets are assigned which logical IDs. Up to 4 handsets can be associated with a single POD. The HAT array contains 16 bytes or 4 sets of handset id numbers each one representing the next logical id. It will send FFFFFF for each time-slot not occupied by a handset. For example (here shown in hex with commas for clarity) if 3 handsets have associated you would get the following bytes with the byte on the left coming in first. 01,02,03,04,05,06,07,08,09,0A,0B,0C,FF,FF,FF,FF would be Logical id 0 04030201h Logical id 1 08070605h Logical id 2 0C0B0A09h Logical id 3 FFFFFFFFh

WI_GetHandheldCopyright

Syntax
WAVITDEV_API int WI_GetHandheldCopyright(char *);

Description
Returns the active Handset's Firmware Copyright information stored in the EEPROM. Only one handset should be active during this query.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Pointer to char	HandheldCopyright	Returns a string of copyright information.

WI_GetHandheldVersion

Syntax
WAVITDEV_API int WI_GetHandheldVersion(char *);

Description
Returns the active Handset's Firmware Version number stored in the EEPROM. Only one handset should be active during this query.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Pointer to char	HandheldVersion	Returns a string containing the version number, such as WavIt 248.

WI_GetPODDate

Syntax
WAVITDEV_API int WI_GetPODDate(char *);

Description
Returns the active POD's Firmware revision date stored in the EEPROM.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Pointer to char	PODDate	Returns a string containing the firmware revision date.

WI_GetPODVersion

Syntax
WAVITDEV_API int WI_GetPODVersion(char *);

Description
Returns the active POD's Firmware Version number stored in the EEPROM.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Pointer to char	PODVersion	Returns a string containing the version number, such as WavIt 248.

WI_GetPODCopyright

Syntax

WAVITDEV_API int WI_GetPODCopyright(char *)

Description

Returns the active POD's Firmware Copyright information stored in the EEPROM.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Pointer to char	PODCopyright	Returns a string of copyright information.

WI_GetInfo

Syntax

WAVITDEV_API int WI_WavItInfo(char *, char *, char *, char *, char *, char *, char *);

Description

Returns the active POD's and Handhelds Firmware information stored in their respective EEPROM.

Note: Only one Handheld should be connected during this call.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Pointer to char	HandheldCopyright	Returns Handheld's copyright information.
Pointer to char	HandheldVersion	Returns Handhelds's firmware version number
Pointer to char	Handheld_Date	Returns Handheld's last revision date
Pointer to char	PODCopyright	Returns POD's copyright information
Pointer to char	PODVersion	Returns POD's firmware Version number
Pointer to char	POD_Date	Returns POD's last revision date.
Pointer to char	HandsetAssignment Table	Returns the Handset Assignment Table which shows which handsets the 0, 1, 2, & 3 user slots are allocated to.

WI_LearnCode

Syntax

WAVITDEV_API int WI_LearnCode();

Description

This command tells the POD to learn an IR code. The POD is placed in a "listening" mode for 10 seconds as it waits for a valid IR signal from an external remote control. Once a good code is received the LED on the POD will blink once. The IR code is then stored in the POD's on-board RAM. See also WI_GetIRCode, WI_GetLastError, and WI_SendIRBlast.

To read the code in the RAM, use WI_GetIRCode.

Note that all other communication to the POD ceases during these 10 seconds of reception.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
None	n/a	n/a

WI_SendIRBlast

Syntax

WAVITDEV_API int WI_SendIRBlast();

Description

This command tells the POD to “blast” the IR code that is stored in is memory (see also WI_LearnIRCode). The powerful blast of IR light will be sent out from the front side of the POD.

Normal use of the IR command set, is as follow:

Start by learning the codes (With WI_LearnCode). These codes can be read off from the POD and stored on the host PC using WI_GetIRCode. During normal operation codes are transferred to the POD (using WI_SendIRCode) before they are blasted into the room (using WI_SendIRBlast). Once stored in the memory, a code can be blasted repeatedly, but if a new code is to be blasted, it needs to first be loaded into memory.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
None	n/a	n/a

WI_GetIRCode

Syntax

WAVITDEV_API int WI_GetIRCode(unsigned char *, int *);

Description

WI_GetIRCode (and its complementary function, WI_SendIRCode) will read (write) the IR code that currently resides in the POD memory. This code can be stored on the PC.

Background information on the code:

All payload data is received (sent) as *bytes*, but in reality they are *words*. The Words are Most Significant Byte first followed by the Least Significant Byte. After the payload has been read, the bytes can be converted to words as MSB*256+LSB.

The first word in an IR code is the frequency. It can be calculated as Freq(Hz)=1/(62.5ns x Word1). It is normally between 37KHz and 40KHz.

The subsequent words describe the number of cycles (periods) that the IR light is ON, then OFF, then ON, etc. The code is terminated by a 0. The codes typically have a length of 100-150 bytes, and a temporal duration of 20-50ms.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Unsigned Char pointer	IRCode	This contains the array of bytes that constitutes an IR code. The maximum length is 512 bytes.
INT pointer	Length	This returns the length of the IR code array.

WI_SendIRCode

Syntax

WAVITDEV_API int WI_SendIRCode(unsigned char *, int);

Description

WI_SendIRCode will write an IR code stored on the host PC into the on-board POD memory. After this is done, the code can be blasted out using the WI_SendIRBlast command. See also WI_GetIRCode., and WI_LearnCode.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Unsigned Char pointer	IRCode	This contains the array of bytes that constitutes an IR code. The maximum length is 512 bytes.
INT	Length	This needs to contain the length of the IR code array that is being sent.

WI_GetD0

Syntax

WAVITDEV_API double WI_GetD0();

Description

This queries the value of D0, which is the calibration scaling factor which is used to properly scale the distance away from the screen. The default value is 31.9 to scale distance into meters. See also WI_SetD0.

Return Type

Type	Description
DOUBLE	Returns value of D0 scaling factor.

Parameter

Type	Parameter	Description
VOID		No input.

WI_SetD0

Syntax

WAVITDEV_API int WI_SetD0(double value);

Description

Sets D0, which is the calibration scaling factor which is used to properly scale the distance away from the screen. The default value is 31.9 to scale distance into meters. See also WI_GetD0.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
DOUBLE	D0	The D0 distance scaling factor.

WI_GetCpixel

Syntax

WAVITDEV_API double WI_GetCpixel();

Description

Queries the value of Cpixel. This is the aspect ratio of the optical sensor array. The default value is 1.093, or 1.00 depending on which model of the WavIt is being used.

Return Type

Type	Description
DOUBLE	Returns value of Cpixel, the sensor array aspect ratio.

Parameter

Type	Parameter	Description
VOID		No input.

WI_SetCpixel

Syntax

WAVITDEV_API int WI_GetCpixel(double value);

Description

Sets the value of Cpixel. This is the aspect ratio of the optical sensor array. The default value is 1.093, or 1.00 depending on which model of the WavIt is being used.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
DOUBLE	Cpixel	The Cpixel value that pertains to the optical sensor array hardware which is being used.

WI_Calibrate

Syntax

WAVITDEV_API int WI_Calibrate(void);

Description

This starts a small program that helps to calibrate the pointing to the screen so that there is a direct relationship between the users pointing direction and where the cursor appears on the screen. Specifically, it calculates the correct Mx, My, Bx, By values and then writes them into the WavIt.INI file.

During the calibration process, a blank screen is shown and the user is instructed to point first to the upper left hand corner of the screen, and then to the lower right hand corner of the screen. With the new calibrations constants loaded the user now has 10 seconds to decide to keep these new calibration settings. If no button is clicked on the remote within this time, the calibration constants revert to the old values.

Note that if the POD is positioned such that the remote does not have it in its field of view during the calibration process, the user will be informed of this and instructed to retry.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
VOID		

WI_GetIRActive

Syntax

WAVITDEV_API int WI_GetIRActive();

Description

Checks to see whether the POD is currently listening for IR signals and hence not able to receive or transmit any other information. When the POD receives a WI_LearnCode command it will enter listening mode for a default time of 10 seconds. As soon as a IR Code is received this flag reverts back to 0.

Return Type

Type	Description
INT	Returns 1 if POD is currently in IR Learn Mode listening for IR signals, otherwise returns 0.

Parameter

Type	Parameter	Description
VOID		

WI_GetCurrentUser

Syntax

WAVITDEV_API int WI_GetCurrentUser();

Description

Returns the number of the handset that currently has control of the mouse cursor.

Return Type

Type	Description
INT	Returns userNo (0-3) for who currently has control of the cursor.

Parameter

Type	Parameter	Description
VOID		

WI_SetSmartPoint

Syntax

WAVITDEV_API int WI_SetSmartPoint(VOID);

Description (currently only for PC)

This function sets the screen cursor movement into a mode that use position increments instead of absolute positioning. It also has an additional cursor speed thresholding to give increased stability, meaning that *when moving below a certain speed, the mouse cursor movement completely stops.* This mode is great for games like World of Warcraft, in which WI_GamePoint or WI_DirectPoint do not work. This mode deviates slightly from an absolute pointer, in that absolute pointing position and actual cursor position can deviate slightly over time, especially if there has been a lot of slow cursor movement. The absolute pointing reference is reset whenever the user points beyond the screen boundaries.

In SmartPoint™ mode, pointing to beyond the screen boundaries will start panning the cursor movement at a steady rate. This is essential for games – Point to the left or right of the screen and the view will pan continuously. See also WI_MoveAtBoundary.

SmartPoint™, GamePoint™, and DirectPoint™ (default) are all slightly different pointing modes.

See also WI_SetDirectPoint and WI_SetGamePoint.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
VOID		

WI_SetDirectPoint

Syntax

WAVITDEV_API int WI_SetDirectPoint(VOID);

Description (currently only for PC)

This is the default pointing mode in which the cursor position is set using absolute coordinates, i.e. where you point is where the cursor appears (after the proper calibration of course). For some games based on DirectX, this mode does not work, as these games look for incremental changes in coordinates.

See WI_SetGamePoint and WI_SetSmartPoint for pointing modes more suitable for gaming.

SmartPoint™, GamePoint™, and DirectPoint™ (default) are all slightly different pointing modes.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
VOID		

WI_SetGamePoint

Syntax

WAVITDEV_API int WI_SetGamePoint(VOID);

Description (currently only for PC)

This is the pointing mode that works best for games most DirectX games, including Battlefield 2 or Halflife 2. In this mode it is still absolute pointing (i.e. where you point is where the cursor goes), but the coordinates are sent to the PC via increments instead of absolute coordinates, and certain PC games only respond to these "increments".

In this mode there is no lower speed threshold below which the cursor will stop moving.

See WI_SetSmartPoint and WI_SetDirectPoint for other pointing modes.

SmartPoint™, GamePoint™, and DirectPoint™ (default) are all slightly different pointing modes.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
VOID		

WI_MoveAtBoundary

Syntax

WAVITDEV_API int WI_MoveAtBoundary(int Panning);

Description (currently only for PC)

This function implements a "panning" function when the user is in "relative" mode (see also WI_SetAbsolute). In this case when the user is pointing outside the boundaries of the screen, "Panning" motion will be sent to the mouse cursor.
Example: When the user points to the right of the screen, continuous "move right by 10 pixels" will be sent. Similar actions apply for pointing above/below/left/right of screen.

This function is useful for compatibility with legacy First-Person-Shooter games where pointing beyond the screen boundaries will allow the user to continuously pan in the specified direction.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
int	Panning	If Panning=1, then the panning feature is turned on.

WI_SetDrawBitmaps

Syntax

WAVITDEV_API int WI_SetDrawBitmaps(int Sprites);

Description (currently only for PC)

When Sprites=1, artificial cursors will be displayed for ALL users. Currently these cursors are different colored squares for the different users. User0= Red, User1=Green, User2=Blue, User3=Yellow.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
int	Sprites	If Sprites=1, the virtual cursors will be shown for each user.

WI_MouseClick

Syntax

WAVITDEV_API int WI_MouseClick(int MouseClick);

Description

By default the center button (the OK button on the remote) emulates the left-click on a mouse. This feature can be turn on and off with this function. See also WI_QuickClick.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
int	MouseClick	If MouseClick=1 (default), the center button will act as a left-mouse button, if MouseClick=0 there will be no mouse-click emulation.

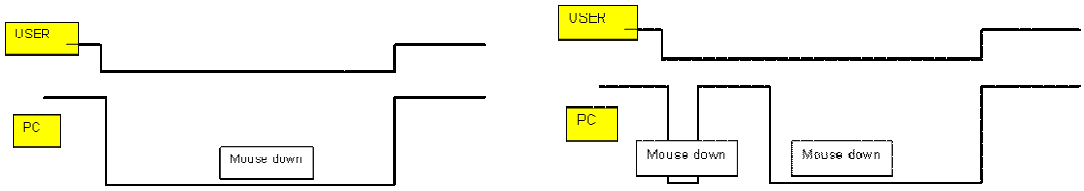
WI_QuickClick

Syntax

WAVITDEV_API int WI_QuickClick(int QuickClick);

Description

By default the center button acts like the left-click on a mouse (see also WI_MouseClick). Moreover, the WavIt has two-click modes called Quick-Click and Normal mode. These are illustrated here:



Normal Mode Quick-Click Mode

The Quick-Click mode (which is the Default mode) was introduced to avoid false detection of "dragging" when you are trying to click on an icon or hyperlink. When the button is pressed, it causes an immediate click *down/up*. This ensures that there will never be a misinterpretation of "dragging" when trying to click to select. If the user continues to hold down the button, it means that he was actually intending to drag. After 300ms of holding down the key, a follow-on command is sent which is the mouse click *down* event. When the button is released it is mouse-click *up* event.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Int	QuickClick	If QuickClick=1 (default), the center button is in QuickClick mode. If MouseClick=0 it is in Normal mode.

WI_SendKeyEvent

(Windows only)

Syntax

WAVITDEV_API int WI_SendKeyEvent(unsigned int input_code1, unsigned int input_code2, int type, unsigned int delay);

Description

Sends a keyboard command to the active DirectX game on the PC as well as most standard Windows applications. This allows the programmer to fairly simply map actions, such as a handset key-press, into game key commands. Example: Send the keyboard command "w" to move forward in the game *Halflife2* by sending WI_SendKeyEvent(17, -1, 0, 35), where 17 is the DirectX **key-code** for "w", -1 means that no second key is sent, 0 means that down/up action was made, and 35 is the millisecond wait between the down and up stroke. The key commands are found in Dinput.h for DirectX9.0.
http://www.koders.com/c/fidBDB37A71845CF044CE6B6E97DBD5C725CFC55580.aspx

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
INT	Input_code1	The DirectX key code for (0-255) that will emulates a keyboard stroke.
INT	Input_code2	This is the optional second key press, similar to Input_code1. "-1" means that no code is sent (default). Having two input codes makes it easy to send, for example, "Ctrl+W".
INT	Type	0 means "down/up" keystroke (default), 1 means "down only", 2 means "up only". Care should be taking when using "down only" as it may end up locking up the keyboard.
INT	Delay	The delay in milliseconds between the emulated down and up stroke.

WI_SmartTracking

Syntax

WAVITDEV_API int WI_SmartTracking(bool bValue);

Description

The WavIt robustness to tracking the POD can be improved by turning on "SmartTracking". SmartTracking is a routine which uses a bit more processing to try to distinguish a bright "bad" IR light source from the "good" POD. The trade-off is that the minimum allowable distance to the screen is increased slightly (~3 ft), and there is now a ~100ms delay in the mouse cursor appearing on the screen when you point away from the POD and quickly back again.

The SmartTracking sensor is always active and can be read from the WI_GetButtons as the TS (track spots) variable. However, when SmartTracking=1, the WavIt only moves the cursor when TS=TRUE.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
BOOL	bValue	If bValue=TRUE (or 1), SmartTracking is active If bValue=FALSE (or 0), SmartTracking is NOT active.

WI_SetDistanceScaling

Syntax

WAVITDEV_API int WI_SetDistanceScaling(bool Dscale);

Description

This sets a DistanceScaling flag to TRUE or FALSE. With DistanceScaling=TRUE the 3D pointing calibration will adjust dynamically as the user moves closer and farther way from the screen, so as to compensate for the fact that a user who is farther away should wave the remote a smaller angle to span the full length of the TV screen. With DistanceScaling=FALSE the pointing calibration is fixed, and relate to the distance from the screen at which the calibration was done.

See also WI_SetCalDistance, WI_GetCalDistance, WI_SetDmin and WI_GetDmin.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
INT	Dscale	Dscale=True (1) turns on automatic calibration of pointing accounting from user distance from screen.

WI_SetCalDistance

Syntax

WAVITDEV_API int WI_SetCalDistance(double caldistance);

Description

Sets the distance at which the calibration was done. This is needed for “DistanceScaling” to work properly.

See also WI_DistanceScaling, WI_SetCalDistance, WI_GetCalDistance, WI_SetDmin and WI_GetDmin.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
DOUBLE	Caldistance	This should be the distance at which the last (or reference) calibration was performed, in meters (ex: 3.20).

WI_GetCalDistance

Syntax

WAVITDEV_API int WI_GetCalDistance(double *caldistance);

Description

Gets the distance at which the calibration was done.

See also WI_DistanceScaling, WI_SetCalDistance, WI_GetCalDistance, WI_SetDmin and WI_GetDmin.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
DOUBLE	Caldistance	Pointer to caldistance. Output parameter.

WI_SetDmin

Syntax

WAVITDEV_API int WI_SetDmin(double dmin);

Description

Sets the minimum distance from the screen that the user can move to and still be able to point directly. Below this distance, the pointing still works, but the calibration constants are adjusted to allow the user to still point to anywhere on the screen without being cut off by the field-of-view of the Handset. This value is only relevant if DistanceScaling is TRUE.

See also WI_DistanceScaling, WI_SetCalDistance, WI_GetCalDistance, WI_SetDmin and WI_GetDmin.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
DOUBLE	Dmin	The Dmin distance in meters. Example: Dmin=1.2

WI_GetDmin

Syntax

WAVITDEV_API int WI_SetDmin(double *dmin);

Description

Gets the minimum distance from the screen that the user can move to and still be able to point directly. Below this distance, the pointing still works, but the calibration constants are adjusted to allow the user to still point to anywhere on the screen without being cut off by the field-of-view of the Handset. This value is only relevant if DistanceScaling is TRUE.

See also WI_DistanceScaling, WI_SetCalDistance, WI_GetCalDistance, WI_SetDmin and WI_GetDmin.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
DOUBLE	Dmin	Pointer to Dmin (distance in meters). Output parameter.

WI_GetCalScreen

Syntax

WAVITDEV_API int WI_GetCalScreen(int *x, int *y);

Description

Returns the x and y screen resolution at which the calibration routine was performed. This is relevant because as the screen resolution changes, the pointing (in x, y position on the screen) needs to be adjusted to account for it so that the pointing still remain “direct”.

See also WI_SetCalScreen, WI_GetX, WI_GetY

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
INT	X	Pointer to X screen resolution at which calibration was done, ex: 1280. This is an output parameter.
INT	Y	Pointer to Y screen resolution at which calibration was done, ex: 768. This is an output parameter.

WI_SetCalScreen

Syntax

WAVITDEV_API int WI_SetCalScreen(int x, int y);

Description

Sets the x and y screen resolution at which the calibration routine was performed. This is relevant because as the screen resolution changes, the pointing (in x, y position on the screen) needs to be adjusted to account for it so that the pointing still remain “direct”.

See also WI_GetCalScreen, WI_GetX, WI_GetY

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
INT	X	X screen resolution at which calibration was done, ex: 1280.
INT	Y	Y screen resolution at which calibration was done, ex: 768.

WI_GetBaudRate

Syntax

WAVITDEV_API int WI_GetBaudRate();

Description

Gets the Baud-rate of communication between the POD and the PC. Regular PODs communicate at 115200 bps, while the new PODs communicate at 500000 bps, specifically PODs paired with the Wavit TalknPoint.

See also WI_SetBaudRate

Return Type

Type	Description
INT	Returns the Baud rate. Valid rates are currently only 500000 and 115200.

Parameter

Type	Parameter	Description
None		

WI_SetBaudRate

Syntax

WAVITDEV_API int WI_SetBaudRate(int value);

Description

Sets the Baud-rate of communication between the POD and the PC. Regular PODs communicate at 115200 bps, while the new PODs communicate at 500000 bps, specifically PODs paired with the Wavit TalknPoint.

See also WI_GetBaudRate

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
INT	Value	Baud rate. Only valid numbers are 500000 or 115200.

WI_GetRawMicArray

Syntax

WAVITDEV_API int WI_GetRawMicArray(unsigned char *SoundPacket);

Description

Gets each raw sound packet that is sent to the POD. Programmers will generally NOT need to use this function, and should instead use the WI_GetRawMicArray.
Raw sound packets consist of an array of 20 bytes. Since sound is actually captured with 10 bit precision and not 8 bit, the 10 bit sound is represented in the following way:

A1,A2, A3, A4, A_Residuals,
B1, B2, B3, B4, B_Residuals,
C1, C2, C3, C4, C_Residuals
D1, D2, D3, D4, D_Residuals

where Residuals contain the extra 2 least significant bits needed to generate 10 bit sound packets. Example for the first 5 bytes:
5x raw packets:
11111100, 11111101, 1111110, 11111111, 00011011
➔
4x 10 bit sound words
1111110011, 1111110110, 1111111001, 1111111100

Note that the least significant bits belong like this (A4, A4, A3, A3, A2, A2, A1, A1).
One approach to build the full 10 bit data is to multiply Byte 1 by 4, then AND the residual byte with 3 (bits 11) and add Byte 1. Then shift the residual byte by 2 bits right and repeat the procedure and so on.
See also WI_GetMicArray

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Uns. char	SoundPacket	Returns an array of 20 bytes containing the sound data.

WI_GetMicArray

Syntax

WAVITDEV_API int WI_GetMicArray(short *SoundPacket, int count, int reset);

Description

Retrieves an array of SoundPackets sent to the POD from the Wavit TalknPoint. The sound is single channel, sent at 7575 Samples/s, 16 bit (although the captured sound is only 10 bit, i.e. -512 to 512). The array is built up with the earliest recorded data in index 0.

Every time a new sound packet is received by the POD, the sound counter is incremented. See WI_GetSoundCount.

When WI_GetMicArray is called the sound counter is reset to 0. So the procedure for reading out sound is to get the array size of the SoundPacket with the WI_GetSoundCount function, and then to retrieve the SoundPacket array with the WI_GetMicArray function. This SoundPacket array can then be appended to the previous SoundPacket to build up a continuously recorded sound stream. The sound packet can also be copied into the appropriate Windows sound buffer.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
SHORT	SoundPacket	Returns an array of 16 bit sound (actually 10 bit, with range -512 to 512).
INT	Count	The size of SoundPacket array.
INT	Reset	“1” Resets the SoundPacket array to size 0 and erases it. “0” should generally be used.

WI_GetSoundCount

Syntax

WAVITDEV_API int WI_GetSoundCount();

Description

Gets the current size of the SoundPacket array. This size is incremented every time a new sound packet is received by the POD from the Wavit TalknPoint. The sound counter is reset to 0 every time the SoundPacket is retrieved using the WI_GetMicArray function call.

Return Type

Type	Description
INT	Return the sound counter.

Parameter

Type	Parameter	Description
NONE		

WI_SetHIDMouse

Syntax

WAVITDEV_API int WI_SetHIDMouse(int HIDOptions);

Description

Sets options for the Wavit HID mouse (for the Wavit HID USB dongle). When the USB HID is plugged in (and if Wavit.exe is not running), the hardware automatically handles the cursor motion and provides some default the button functions. When Wavit.exe runs, it will initialize and turn off the HID button functions, while still letting the cursor motion and OK button be handled by the HID hardware. Calling this function can change these option.
The argument is a bit mask for activating the following functions.

Bit0=All Buttons ON/OFF so software takes care of buttons
Bit1= Mouse Cursor Movement
Bit2= OK button ON(1) or OFF(0).

For example, to turn the Mouse movement and all buttons off, HIDOptions=0
To turn Cursor motion on and OK button on, while leaving other buttons off HIDOptions=2+4=6.

See also WI_SetActivateCursor(int value) which activates/deactivates the Wavit.exe software controlled mouse cursor motion.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Int	HIDOptions	Bit0=All Buttons ON/OFF so software takes care of buttons Bit1= Mouse Cursor Movement Bit2= OK button ON(1) or OFF(0).

WI_GetHIDdeviceMode

Syntax

WAVITDEV_API int WI_GetHIDDeviceMode(bool *HIDon);

Description

Determines whether the attached POD hardware is the original POD or the new Wavit HID POD. Returns 1 if it is a Wavit HID USB dongle.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Int	HIDon	Pointer. Returns 1 if the connected POD is a Wavit USB HID dongle.

WI_SetCenterSWButton

Syntax

WAVITDEV_API int WI_SetCenterSWbutton(bool OKbutton);

Description

Sets the Wavit.exe handling of the Center Button (aka Software button). For the WavitHID device the Center Button is normally set to be controlled by the Wavit HID hardware. However, this can be turned off using WI_SetHIDMouse() function. To turn ON the Wavit.exe software handling of the OK button (i.e. the left-mouse click), set OKbutton=1.

See also WAVITDEV_API int WI_MouseClick() for turning on/off the center mouse button.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Int	OKbutton	Set OKbutton=1 to turn ON the Wavit.exe software OK button to be the left-mouse button. Set OKbutton=0 to turn off left-mouse click.

WI_SetHID_Cal

Syntax

WAVITDEV_API int WI_SetHID_Cal(float xscale, float yscale, float xoff, float yoff, float xclip, float yclip);

Description

Sets the TV screen calibration constant for the Wavit HID device. Adjusting these will change the pointing calibration. The Wavit HID microcontroller will calculate coordinates based on:

X=Xscale*X1+Xoffset
Y=Yscale*Y1+Yoffset

Example for HID_Cal assuming a default screen of 32768x32768.
XSCALE = 1.241212; // 32768/(35200*.75) use 75% of the sensor
YSCALE = 2.206532; //XSCALE * 16/9 make is wide screen ratio, since mouse driver will scale to screen size
XOFFSET =5461.333; //((35200*XSCALE)-32768)/2
YOFFSET =15391.03; //((28800*YSCALE)-32768)/2
XCLIP = 32767.0;
YCLIP = XCLIP

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
float	Xscale, yscale, xoff, yoff, xclip, ycplip	Screen pointing calibration parameters based on equations: X=Xscale*X1+Xoffset Y=Yscale*Y1+Yoffset

WI_SetHID_Antishake

Syntax

WAVITDEV_API int WI_SetHID_Antishake(unsigned int thresh, unsigned char viscosity);

Description

Sets the Anti-shake parameter for the Wavit USB HID dongle. Note that the normal anti-shake functions do not apply to the HID device *IF* the cursor is being controlled by the hardware as opposed to the software. (see also WI_SetXYCounterMax() and WI_SetStabThresh())

Threshold is a value of the pixel speed/frame, after scaling (x100). Below this speed the anti-shake algorithm kicks in. Example: 700 is 7 pixels/frame.

Viscosity is can take of value of 0-16 (default=3). This means that the viscosity will dynamically ramp between 0 and 3, inclusive, depending on whether the pointing speed is above or below the speed Threshold.

We use the IIR filter approach:
out = last(1- 2^(-viscosity)) + current*2^(-viscosity).

Viscosity=0 means no filtering, and =16 means infinite filter (the old value never changes). Values of 1-8 are reasonable. 8 gives an alpha of 256 or a time constant of ~ 2 seconds which is very long indeed. (The IIR is calculated every 8ms)

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
int	thresh	Sets the threshold parameter for the antishake. (default=700)
Unsigned char	Viscosity	Sets the viscosity parameter for antishake. (0-16)

WI_SetWavitMenu

Syntax

WAVITDEV_API int WI_SetWavitMenu(bool WavitMenu);

Description

If WavitMenu=0, Wavit.exe will show a button mapping help screen when the handset MENU button is pressed. Setting WavitMenu=1 will disable this screen.

Normally when the GUI program WavitMENUXXX.exe is running it will turn WavitMenu=1 when it is running thereby disabling the more basic WavitMenu help screen generated by Wavit.exe.

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
bool	WavitMenu	Controls appearance of basic WavitMenu help screen when OK button is pressed. WavitMenu=0 prevents basic screen from appearing.

WI_GetRawData

Syntax

WAVITDEV_API int WI_GetRawData(int *size,unsigned char *RawData);

Description

Returns an array of bytes containing the raw data being received from the handset at about 30 packets/second

There are numerous types of data that can be returned. The primary one is coordinate data (Type=1). For two spots, the length is 14 and for 4 spots is will return 20. Coordinate data looks like this:

Buttons(2), x1(2),y1(2),p1(2),x2(2),y2(2),p2(2),…, p4(2).

where the () indicates the number of bytes for each variable.

Example: when reading the coordinate data when the unit is not pointed at the POD and buttons are not pressed, the data should look like this in byte format

Buttons	X1	Y1	P1	X2	Y2	P2
255, 255,	2,238,	2,238,	0,0,	2,238,	2,238,	0,0

To convert the button and coordinate data to real numbers, we need to convert to "word" format where each word is Big Edian convention, meaning that the Most Significant Byte (MSB) is followed by the Least Significant Byte (LSB). We convert using the following equation:

Word(x) = 256*MSB + LSB
= 256*Byte(0) + Byte(1).

A simple loop will take care of the proper conversion for all variables:

```
For (i=0, Length/2)  Word(i)=256*Byte(2*i) + Byte(2*i + 1);
```

Where Word(i) is the new array derived from the original byte array, Byte(i).

Important: Make sure to recast Byte(i) into a double precision array BEFORE doing the addition. Word should be double precision.

Finally we can interpret the Word array as
Buttons = Word (0)

X1= Word(1) / 100
Y1=Word(2) / 100
P1=Word(3) / 100
X2=Word(4) / 100
Y2=Word(5) / 100
P2=Word(6) /100
..
P4=Word(9)/100

And the Buttons should be interpreted as:
Bit0= Button 1 also known as button NE
Bit1= Button 2 also known as button N
Bit2= Button 3 also known as button NW
Bit3= Button 4 also known as button E
Bit4= Button 5 also known as button C
Bit5= Button 6 also known as button W
Bit6= Button 7 also known as button SE
Bit7= Button 8 also known as button S
Bit8= Button 9 also known as button SW
Bit9= Button 10 also known as button TO
Bit10=Button 12
Bits11=Button 13
Bit12= Button 11 also known as the TS button.
Bits13-14 = Handheld Unit number (0 to 3)
Bit 15 = “Battery low” indicator. 0=low (currently below 2.25V)

Return Type

Type	Description
INT	Return 1 upon successful completion, -1 for error.

Parameter

Type	Parameter	Description
Int	size	Size of the returned array
Unsigned char array	RawData	Returned array containing the raw data