



**ThinkOptics' WavIt™
API Document**

Document Revision

<i>Rev.</i>	<i>Date</i>	<i>Changed by</i>	<i>Comments</i>
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

Note: Content is subject to change without notice.

© ThinkOptics, Inc. All rights reserved

ThinkOptics WavIt™ API

About the WavIt

The WavIt™ 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.

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

How to purchase

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

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 is 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.

Windows/system32:

WavItDev.dll - used by the API to allow third-party communication with the WavIt system.

SiUSBXp.dll - USB driver.

My Documents/WavIt/:

API documentation - Complete explanation (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
WavIt.exe - ThinkOptics WavIt MC Driver installation
- the WavIt executable that moves the cursor. This Needs to be running for the API calls to work.

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

Example programs can be found on the installation CD.

Labview8:

Various sample programs - Example programs and a Userlib for quickly programming the WavIt™ system in Labview 8.2.1.

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 WavItService.exe when running as a Service on the PC) program is executed, the user can communicate with the WavIt through a series of simple .dll calls (or .framework calls).

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 Windows registry (or the Wavit.ini file for the Mac). They can be changed directly using Windows RegEdit in a Command prompt, or they can be changed by calling the appropriate API commands described in this document.

This is an example of what the HKEY_LOCAL_MACHINE/SOFTWARE/Thinkoptics/WavIt/Settings registry looks like:

Name	Type	Data
(Default)	REG_SZ	(value not set)
ActivateCursor	REG_DWORD	0x00000001 (1)
Button10ApplicationPath	REG_SZ	C:\Documents and Settings\Anders\My Documents\WavIt\Start_WavItMENU.exe
Button1ApplicationPath	REG_SZ	notepad.exe
Button2ApplicationPath	REG_SZ	...
Button3ApplicationPath	REG_SZ	...
Button4ApplicationPath	REG_SZ	...
Button5ApplicationPath	REG_SZ	...
Button6ApplicationPath	REG_SZ	...
Button7ApplicationPath	REG_SZ	...
Button8ApplicationPath	REG_SZ	...
Button9ApplicationPath	REG_SZ	...
ButtonApplicationDialog	REG_DWORD	0x00000000 (0)
Bx	REG_SZ	-1200.00
By	REG_SZ	-300.00
CalDistance	REG_SZ	3.10
ClickStill	REG_DWORD	0x00000008 (8)
Cpixel	REG_SZ	1.0000
CThetaX	REG_SZ	1.00
D0	REG_SZ	70.00
DistanceCounterMax	REG_SZ	12.00
DistanceScaling	REG_DWORD	0x00000001 (1)
DistanceThresh	REG_SZ	0.0400
Dmin	REG_SZ	3.20
Mx	REG_SZ	9.50
My	REG_SZ	8.70
PowerThresh	REG_DWORD	0x00000001 (1)
ReadBufferLengthDelay	REG_DWORD	0x00000000 (0)
RollCounterMax	REG_DWORD	0x00000005 (5)
RotationCompensation	REG_DWORD	0x00000001 (1)
Sleepingfactor	REG_DWORD	0x0000001e (30)
StabThresh	REG_SZ	7.00
StartFlag	REG_DWORD	0x00000001 (1)
SubframeInterpolation	REG_DWORD	0x00000003 (3)
WavItMenu	REG_DWORD	0x00000000 (0)
XcalScreen	REG_DWORD	0x00000500 (1280)
XYCounterMax	REG_DWORD	0x00000003 (3)
YcalScreen	REG_DWORD	0x00000300 (768)

The Quick-and-Dirty button programming:

WavItMENU.exe is an application that is provided that gives extensive options for programming up the WavIt Controller's buttons to have different functions. There is an even simpler way that does not require any programming or use of WavItMENU: Simply add an application path to the specific "ButtonXApplicationPath" registry, where X refers to the Button number, example: button 10 = "Menu" button, Button "1"=Red "B" button. Now when you press the button, that specific application is started. Example, of Button1ApplicationPath= notepad.exe, then pressing the Red "B" button on the handset will start up Windows "notepad.exe". Note that setting ApplicationDialog=1 causes a pop-up dialog to appear before executing the named application. Also, if WavItMenu=1, then all the ButtonXApplication paths are inactive.

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 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_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_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);
#ifdef _WIN32
WAVITDEV_API int WI_SendKeyEvent(int input_code1, int input_code2, int
type, unsigned int delay);
#endif
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);

```

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

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. <i>Example:</i> 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. <i>Example:</i> 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. <i>Example:</i> 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	<p>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.</p> <p>The HAT array contains 16 bytes or 4 sets of handset id numbers each one representing the next logical id. It will send FFFFFFFF 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.</p> <p>01,02,03,04,05,06,07,08,09,0A,0B,0C,FF,FF,FF,FF</p> <p>would be</p> <p>Logical id 0 04030201h Logical id 1 08070605h Logical id 2 0C0B0A09h Logical id 3 FFFFFFFFh</p>

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 its 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 $\text{Freq(Hz)} = 1 / (62.5\text{ns} \times \text{Word1})$. 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 MouseButton);

Description

By default the center button (the OK button on the remote) emulates the left-click on a mouse. This feature can be turned 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	MouseButton	If MouseButton=1 (default), the center button will act as a left-mouse button, if MouseButton=0 there will be no mouse-click emulation.

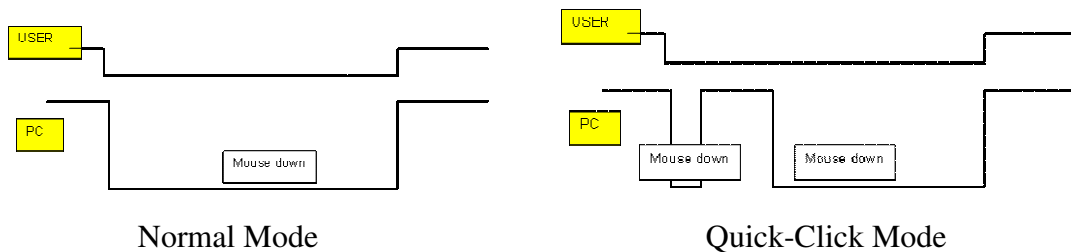
WI_QuickClick

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 WayIt has two-click modes called Quick-Click and Normal mode. These are illustrated here:



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 QuickClick=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 emulate 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 taken 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.