UPDD Settings

Revision 1.4 – 9th April 2014
www.touch-base.com\documentation\API

Location Notes Example updates API Calls Adhoc Settings Contact

The driver's settings are held in the registry (Windows) and a settings file, tbupdd.ini

Settings Location	Description
tbupddsu – click link for settings definitions HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\	Setting that relate to the Windows kernel mode element of the driver.
tbupddwu - click link for settings definitions HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\	Setting that relate to the Windows user mode element of the driver.
Tbupdd.ini - click link for settings definitions	Main driver and device settings

Operating system	Location
Windows	UPDD Application folder (e.g. c:\program files\UPDD)
Linux	/opt/tbupddlx
Mac OS X	4.1.10 - /tbupddmx 5.0.x - /Library/Preferences
Solaris	/opt/tbupddso
CE	\Windows

Setting file notes

File location

The settings file is managed by the driver and ***must always*** be located in the same folder as the active copy of the driver module tbupddwu (or the parent folder on 64 bit windows systems).

Settings file backups

UPDD maintains backups the settings files, tbupdd.ini, to allow for recover in the case of system crashes where the updd settings are corrupted.

Tbupdd.ini.bak.date-time, e.g. Tbupdd.ini.bak.20100821-010703

Each time settings are changed a backup is created. These files are small in size and in normal use this is unlikely to cause an issue; however in test systems sometimes a larger number of files are created and even though this is insignificant in practice it has raised concerns with some integrators. This behaviour can now be suppressed by creating a working copy of the settings file tbupdd.ini called tbupdd.ini.master. In the event of a recovery being needed this master file is used. NB this will return all settings to those in effect at the time the master copy is created. With this file in place no further backups are created and existing ones can be manually deleted or they will be in any case be deleted automatically as they expire.

When the settings file is opened the driver checks for validity and if this check fails it references the backup files until a valid setting file is found. This was implemented to overcome corrupted files due to power outage corrupting open files in very specific system configurations.

The backup files are deleted once they are greater than 3 days old.

Setting file layout

The settings held in the tbupdd.ini file are placed in different sections relative to their meaning as follows:

Click on the branch to see settings held in the section and their meaning.

Global driver settings	[UPDD]	High level driver and system configuration settings
General driver settings	[UPDD\parameters]	General driver and system settings
Configured Device Settings	[UPDD\parameters\N]	Specific device settings, one for each device configured in the driver.
Device sub-section and nodes		Sub sections related to the device
Calibration	[UPDD\parameters\N\calibration styles\N]	Calibration style specific settings. One branch for each calibration style associated with the touch device
macros	[UPDD\parameters\N\macros\N] (N= 1,2,3 and 4)	UPDD macro string defined for when the controller starts (0), Controller stops (1), driver loads (2), driver unloads (3)
Toolbars	[UPDD\parameters\N\toolbars\N]	Toolbar settings. One branch for each toolbar defined on the touch device.
Default device settings	[UPDD\parameters\controller\tsNNN]	Default device settings for each touch device supported by the driver package. When a device is configured in the driver (via PnP or manually added in the case of serial or ps/2) the driver creates a new device section [UPDD\parameters\N] from these default settings. It then references the [Extra] settings branch to see if there are any settings defined here for a given device that should overwrite the default settings.
[Extra]		Extra (overwrite) device settings used when a device is configured in the driver.

<eof>	End of file marker. If missing, the driver discards the setting file as corrupt and uses the backup file

Updating the settings

The are numerous ways to update the driver settings:

- Customised setting
  These can be defined in a custom settings file that is used during software install to override default settings.
- UPDD Console
  UPDD Console, where available, allows for standard general settings to be updated via the GUI.
- UPDD API
  UPDD Application Program Interface (API) interface defines both specific function calls to update a defined setting and generic update settings function calls.
- Command line
  Utilise the command line interface to update device settings. Prior to version 4.1.10 the command line interface was available via the TBcalib program. From 4.1.10 onwards there is a separate command line interface program called TButils.
- UPDD Specialised dialogs
  Some specific setting sets, such as UPDD TUIO Server or UPDD Gesture definitions, have their own dialogs to update related settings.
- Manual Update
  Using a file editor to manually update settings.

Example Setting updates

This section highlights a number of useful setting that control some of the more common function and features that are not exposed via the UPDD Console.

Some UPDD functions have specific UPDD API calls but other functions are simply controlled by the value of a setting in the settings file. If, as a programmer using the UPDD API, there is no obvious API call to implement a desired function it may be possible to invoke the function simply by changing the file setting and applying the change using call TBApiApply or TBApiApplyNoReload as appropriate. Other examples, when using TButils to update the settings, can be found here.

Touch packet filters

In some circumstances it may be desirable to not process the initial touch packets but to filter them out either based on packet count or time threshold.

In a few cases dealing with some very old touch controllers we noticed that the first few data packets did not carry the correct x and y co-ordinate as the controller took a few packets to 'zero' in on the point of touch. In this instance we were able to set the TouchDownFilter setting to discard the number of inaccurate packets.

In another case we had a customer using a projected capacitive controller in an environment where the ambient electrical noise was causing the touch controller to generate the occasional random touch when the touch screen was not in use. We implemented a touch filter based on touch duration so that the driver will ignore any touches less than specified touch threshold. In the examples below the driver would filter out the first 3 data packets and ignore touches less than 100ms in duration:

Command Line	API
tbcalib [Device=N] /setting:touchdownfilter=3 tbutils [Device N] setting touchdownfilter 3	TBApiSetSettingDWORD(passedDeviceNumber,_T("touchdownfilter "),3);
tbcalib [Device=N] /setting:touchdowntimefilter=100 tbutils [Device N] setting touchdowntimefilter 100	TBApiSetSettingDWORD(passedDeviceNumber,_T("touchdowntimefilter "),100);

Touch clicks only

Some applications or system users do not want to generate motion when the touch device is in use. Touch pointer motion is determined by the controller setting Motion being set being disabled (0) or enabled (1)

Command Line	API
tbcalib [Device=N] /setting:motion=1 tbutils [Device N] setting motion 1	TBApiSetSettingDWORD(passedDeviceNumber,_T("Motion"),1);
Tbcalib [Device=N] /setting:motion=0 tbutils [Device N] setting dw motion 0	TBApiSetSettingDWORD(passedDeviceNumber,_T("Motion"),0);

Show system tray

To enable or disable the system tray icons set the general setting 'Show systray' to 0 (disable) or 1 (enable).

Command Line	API
tbcalib Device=0 "/setting:Show Systray=1" tbutils Device 0 setting "Show Systray" 1	TBApiSetSettingDWORD(0,("Show systray"),1);
tbcalib Device=0 "/setting:Show Systray=0" tbutils nodevice "setting dw "Show Systray" 0	TBApiSetSettingDWORD(0,("Show systray "),0);

Enable or Disable a device

To enable or disable a device set the controller setting 'Enabled' to 0 (disable) or 1 (enable), passing the Device Handle of the device.

Command Line	API
Tbcalib [Device=N] /enable	TBApiSetSettingDWORD(passedDeviceNumber,_T("Enabled"),1);
Tbutils [Device N] enable	
Tbcalib [Device=N] /disable	TBApiSetSettingDWORD(passedDeviceNumber,_T("Enabled"),0);
Tbutils [Device N] disable	

Set UPDD language

UPDD implements its own language system and translation files are supplied for various languages and new ones are easily implemented via the language tool. This means that the UPDD can display its own language within its dialogues irrespective of the system's locale (assuming code pages are available to display the characters).

To enable a specific language use the TBApiSetSettingSZ call to set the bundle registry key 'Language' to a language value; i.e. FR = French, EN = English, JP = Japanese, ES = Spanish, DE = German, IT = Italian etc.

Settings API calls

There are a number of generic API function calls to set and retrieve entries in the settings file. See the individual calls for more information.

API Call	**Description**
Global driver settings	[UPDD]
TBApiGetGlobalSettingDWORD	Get global DWORD setting
TBApiSetGlobalSettingDWORD	Set global DWORD setting
TBApiGetGlobalSettingSZ	Get global String setting
TBApiSetGlobalSettingSZ	Set global String setting
General driver settings	[UPDD\parameters] (device = 0)
Configured Device Settings	[UPDD\parameters\N] (device = 1)
TBApiGetSettingDWORD	Get DWORD setting
TBApiSetSettingDWORD	Set DWORD setting
TBApiGetSettingSZ	Get String setting
TBApiSetSettingSZ	Set String setting

There are also extended versions of the above that allow a named sub-section and node to be accessed:

Device sub-section and nodes	[UPDD\parameters\N\[sub-section]
TBApiGetSettingSZEx	Extended get String setting
TBApiSetSettingSZEx	Extended set String setting
TBApiGetSettingDWORDEx	Extended get DWORD setting
TBApiSetSettingDWORDEx	Extended set DWORD setting
Default device settings	[UPDD\parameters\controller\tsNNN]
TBApiGetControllerDWORD	Get DWORD setting
TBApiGetControllerSZ	Get String setting

Some of these API's perform special processing that may be extended in future releases. For example, for TBApiSetSettingSZEx, if the aName argument is set to "**Calibration Style**" and aSubtree equals "", the indicated style (in argument aSZ) is activated. e.g. for calibration styles - use aSubTree = "" to emulate the "non-extended" versions

The following example shows the settings that make up a typical UPDD sub-section:

TBUPDD.ini file
[UPDD]\Parameters\1\Number Of Calibration Styles - item count
[UPDD]\Parameters\1\Calibration Style - active item name
[UPDD]\Parameters\1\Calibration Styles\0\Calibration Style - item name
[UPDD]\Parameters\1\Calibration Styles\0\... - other item data

These calls work exactly the same in all OS environments.

Ad hoc settings

Some pointer devices have certain characteristics or functions that are not utilised by UPDD but may require settings to be held by UPDD that relate to these characteristics. These settings, which are not used by UPDD, are stored in the settings file as device-specific configuration details to allow retrieval via the API if required.

Application programs may need to retrieve these settings. For example, a program using a digitizer tablet may need to lookup device characteristics, such as the device width in inches. In this case UPDD will be shipped with preset values in:

[UPDD\Parameters\n\Ad Hoc Settings]

where **n** is the device number. This entry holds a semi-colon separated list of device specific settings in the following format:
VALUE NAME;VALUE TYPE;SETTING[crlf]VALUE NAME;VALUE TYPE;SETTING [crlf]…
Where:

VALUE NAME	Name of the ad hoc setting
VALUE TYPE	Type is either an '**S**' indicating the value is held as a string or '**H**' for hexadecimal ([crlf] is a carriage return/linefeed combination used as a separator).
SETTING	The actually setting

Using the VALUE NAME and VALUE TYPE it is possible to enumerate through the actual registry settings (if they are unknown) by calling either TBApiGetSettingSZ (for string values) or TBApiGetSettingDWORD (hex values) for each VALUE NAME. Note that the SETTING shown may match the actual registry entry but it is safer to access the latter as this may have been changed (under application control) from the factory settings.

However, to make life easier, an API call is available which will enumerate through any ad hoc settings, - see TBApiEnumAdhocValues for more details.

Example of ad hoc settings:
Registry value "Ad Hoc Settings" could contain:

```
AHOrigin_X;S;Lower Left[crlf]AHOrigin_Y;S;Lower Left[crlf]AHCTX_ORG_X;X;14[crlf]AHCTX_ORG_Y;X;14
```

The actual setting values are:

```
AHOrigin_X = Lower Left     (string)
AHOrigin_Y = Lower Left      (string)
AHCTX_ORG_X = 14            (hex)
AHCTX_ORG_Y = 14             (hex)
```

Note: If ad hoc settings have been defined for a device, the settings can be viewed using the UPDDDEMO demonstration program or directly in the tbupdd.ini file. Conventionally, all ad hoc settings are prefixed '**AH**'.

Contact

For further information or technical assistance please email the technical support team at technical@touch-base.com.