Utilising the API

The UPDD API allows user mode applications to interface directly with the driver and/or the pointer devices handled by the UPDD pointer device driver. It is assumed that the reader is familiar with the various functions and parameters of the driver since that information is not duplicated here.

The UPDD API is supported on all platforms supported by the driver and is used by the driver's own utility programs, such as the UPDD Console, test program, calibration tool, gestures, TUIO server etc.

The files needed to build and link the API, i.e. the header in all cases and the .lib file in Windows are located in the API sub folder of the UPDD install folder. The run time files (dynamic library in the table below) are held in appropriate locations for the operating system.

OS	**Header**	**Import Library**	**Dy**
Windows	updapi.h	updapi.lib	up
Mac OS/x	updapi.h	n/a	libu
Linux	updapi.h	n/a	up

* *UPDD version 6 software is mostly implemented in 32 bit code that executes in compatibility mode on 64 bit systems.*

Only the parts that must be 64 bit such as some installer extensions are in 64 bit code. This approach works for all current Windows desktop systems. Therefore the shipping API referenced above is 32 bit code. For any UPDD 64bit client applications that require 64 bit api files they are available here.

The header file defines and documents all available API functions, macros and constants and this should be the main point of reference during API development.

Following the standard driver install the API files can be found in the following locations:

OS	Header	Import	Dynamic
Windows	c:\program files (x86)\UPDD\api	c:\program files (x86)\UPDD\api	c:\program files (x86)\UPDD

Mac OS X	/Library/Application Support/api	n/a	/usr/local/lib
Linux	/opt/updd/api/	n/a	/opt/updd/api/

upddapi.dll / .so / .dylib is tightly linked to a specific version of the driver. You ***must*** use the version installed with the driver.

One simple way to achieve this under Windows is to launch your application with the current working directory set to the UPDD API folder. This avoids making a copy of the file and thus avoids the risk of using an incorrect copy. Under Linux / Mac creating a symbolic link to the installed lib is recommended.

UPDD clients applications are guaranteed to work with the same or newer versions of the driver but cannot be guaranteed to work with older versions. The version 6 API is not compatible with the version 5 API.

Using the above files you may wish to write a simple application to see the UPDD API interface in action.

To aid your development we offer a number of example programs, the simplistic being a console program example, available here with source code example.

Notes

TBApiOpen will fail if the API version is incompatible with the calling client application. The TBApiGetLastError can be used to get additional information regarding any API error.

The comms layer used by the API has a compatibility check, basically the client and server set a "schema" level and comms can only be initiated if these are equal, so if the software changes so the API is no longer compatible this level is incremented.

As a general rule, the API is tightly bound to a version of the driver and the version number of the API must match that of the driver.

In practice this is not completely true; there is some leeway and it depends on the nature of any API changes. The backward compatibility is at the interface level; meaning that a program built against a certain version of the API should work with newer versions of the API without recompilation.

Utilising the API

Using callbacks

Using the API, an application can register callbacks to receive notification of a wide range of events internal to the driver. To receive notification of these events applications should register interest via TBApiRegisterDataCallback defining the relevant event settings (data-type constants).

The nature of the registered callback determines the type of data returned in a PointerEvent structure (although not all events are accompanied by additional data).

Driver connection considerations

An application that utilises the API should terminate if it receives a callback event of type _EventConfiguration with CONFIG_EVENT_UNLOAD. This is to allow the uninstaller to delete files that would otherwise be locked.

Additionally the program might react to the callback events of type _EventConfiguration with CONFIG_EVENT_CONNECT / CONFIG_EVENT_DISCONNECT. These indicate situations where the driver becomes available or unavailable. A program might for example avoid making API calls or otherwise alter its behavior in the case that the driver is not connected.

Data Structures

PointerEvent Structure

This structure (struct _PointerEvent) can be found in the header file tbapi.h. It is used to hold pointer (and other) data returned from the driver to a registered callback routine.

Qt application considerations

The UPDD API uses QT and there are certain considerations when using a Qt client application with UPDD API. Given the API also uses Qt and must by definition share a process space with the client app there are a few limitations.

1. TBApiOpen must be called after QCoreApplication (or a derived object such as QApplication) is created.

2. The client program must, if possible, be built against the same major Qt version as the API. Currently 5.x.x
3. The client must use the Qt binaries from the same Qt version. It is recommended that the client app use the Qt binaries installed by the driver.

MacOS addendum

Alternatively to 3 above, if using a QT UPDD API client application with Mac OS you can patch a private copy of the api to link with your qt installation:

1. Take a private copy of libupddapi.1.0.0.dylib and place it in a location where your application will reference this in preference to the installed copy.
2. Execute the following commands:

   install_name_tool -change @rpath/QtWidgets.framework/Versions/5/ QtWidgets **<path to your qt version>**/clang_64/lib/QtWidgets.framework/Versions/5/QtWidgets libupddapi.1.0.0.dylib

   install_name_tool -change @rpath/QtCore.framework/Versions/5/ QtCore **<path to your qt version>**/clang_64/lib/ QtCore.framework/Versions/5/ QtCore libupddapi.1.0.0.dylib

   install_name_tool -change @rpath/QtSql.framework/Versions/5/ QtSql **<path to your qt version>**/clang_64/lib/ QtSql.framework/Versions/5/ QtSql libupddapi.1.0.0.dylib

   install_name_tool -change @rpath/QtGui.framework/Versions/5/ QtGui **<path to your qt version>**/clang_64/lib/ QtGui.framework/Versions/5/ QtGui libupddapi.1.0.0.dylib

   install_name_tool -change @rpath/QtXml.framework/Versions/5/ QtXml **<path to your qt version>**/clang_64/lib/ QtXml.framework/Versions/5/ QtXml libupddapi.1.0.0.dylib

   install_name_tool -change @rpath/QtNetwork.framework/Versions/5/ QtNetwork **<path to your qt version>**/clang_64/lib/ QtNetwork.framework/Versions/5/ QtNetwork libupddapi.1.0.0.dylib

   In the commands above libupddapi.1.0.0.dylib is the private copy of the api

   e.g.

   ```
   UpddLibWrapper:: ()
   {
   ```

QLibrary lib {"/Users/gary/Desktop/UPDDTest/upddapi.1.0.0"};

copy libupddapi.1.0.0.dylib to the above location

install_name_tool -change @rpath/QtWidgets.framework/Versions/5/QtWidgets /**Users/gary/Qt_unified/5.11.2**/clang_64/lib/QtWidgets.framework/Versions/5/QtWidgets libupddapi.1.0.0.dylib

etc

Examples

Example programs and their source code can be found here.