How to write a 2 in 1 aware application

*Dynamically adapting your UI to 2 in 1 configuration changes*

**Introduction**

With the introduction of 2 in 1 devices, applications need to be able to toggle between “laptop mode” and “tablet mode” to provide the best possible user experience. Touch-optimized applications are much easier to use in “tablet mode” (without a mouse or keyboard), than applications originally written for use in “laptop mode” or “desktop mode” (with a mouse and keyboard). It is critical for applications to know when the device mode has changed and toggle between the two modes dynamically.

This paper describes a mechanism for detecting mode changes in a Windows* 8 or Windows* 8.1 Desktop application, and provides code examples from a sample application that was enhanced to provide this functionality.
Basic Concepts

Desktop Apps vs. Touch-Optimized Apps
Most people are familiar with Desktop mode applications. Windows* XP and Windows* 7 applications are the examples. These types of apps commonly use a mouse and a keyboard to input data, and often have very small icons to click on, menus that contain many items, sub-menus, etc. These items are usually too small and close together to be selected effectively using a touch interface.

Touch-optimized applications are developed with the touch interface in mind from the start. The icons are normally larger, and the number of small items are kept to a minimum. These optimizations to the user interface make using touch-based devices much easier. With these UI elements correctly sized, you should extend this attention to the usability of objects the application is handling. Therefore, graphic objects representing these items should also be adapted dynamically.

The original MTI application
The MTI (MultiTouchInterface) sample application was originally written as part of the Intel® Energy Checker SDK (see Additional Resources) to demonstrate (among many other things) how the ambient light sensors can be used to change the application interface.

At its core, the MTI sample application allows the user to draw and manipulate a Bézier curve. The user simply defines in order, the first anchor point, the first control point, the second anchor point, and finally the second and last control point.

Figure 1 shows an example of a Bézier curve. Note that the size and color of each graphic element are designed to allow quick recognition—even via a computer vision system if required—and easy manipulation using touch.

- Anchor points are square and tan.
- Control points are round and red.
- The segment joining an anchor point to its control point is blue.
- The Bézier curve is black.
Figure 1: Select control and anchor points to draw Bezier curve.

Figure 2, Figure 3, Figure 4, and Figure 5 show the key interactions the user can have with the Bézier curve. An extra touch to the screen allows redrawing a new curve.

Figure 2: A green vector shows the displacement of the control point.
Figure 3: A grey arc shows the rotation of the Bezier curve.

Figure 4: A green vector shows the change of the Bezier curve placement onscreen.
Support for Ambient Light Sensors (ALS) was added to the MTI sample application. Once the level of light is determined, the display dynamically changes to make it easier for the user to see and use the application in varying light situations. Microsoft recommends increasing the size of UI objects and color contrast as illumination increases.

MTI changed the interface in numerous stages, according to the light level. In a bright light situation, the MTI application changes the display to “high contrast” mode, increasing the size of the anchor and control points and fading the colors progressively to black and white. In a lower light situation, the application displays a more colorful (less contrasted) interface, with smaller anchor and control points.

Indeed, anyone who has used a device with an LCD screen, even with backlight, knows it may be difficult to read the screen on a sunny day. Figure 6 and Figure 7 show the issue clearly.
In our case, we decided to re-use the size change mechanism that we implemented for the ALS support. We are using only the two extremes of the display changes for the UI objects’ size that were introduced for the ALS support. We do this, simply by setting the UI objects’ size to the minimum when the system is in non-tablet mode, and to the maximum when it is in tablet mode (by convention, the unknown mode maps to the non-tablet mode).

**Modified MTI (aka: Bezier_MTI)**

Using the two extremes of the display shown above, the original MTI source code was modified to add new capabilities to toggle between the two contrast extremes based on a certain event. The event used to toggle between the two contrast extremes is the switch between tablet mode and laptop mode of a 2 in 1 device. Switches in the hardware signal the device configuration change to the software (Figure 8).

Upon starting the Bezier_MTI application, the initial status of the device is unknown (Figure 9). This is because the output of the API used to retrieve the configuration, is valid only when a switch notification has been received. At any other time, the output of the API is undefined.
Note that only the first notification is required since an application can memorize that it received a notification using a registry value. With this memorization mechanism, at next start, the application could detect its state using the API. If the application knows that it has received a notification in the past on this platform, then it can use the GetSystemMetrics function to detect its initial state. Such mechanism is not implemented in this sample.

![State machine diagram](image)

**Figure 9:** State machine.

When the mode of the device is changed, Windows sends a WM_SETTINGCHANGE message to the top level window only, with “ConvertibleSlateMode” in the LPARAM parameter. Bezier_MTl detects the configuration change notification from the OS via this message.

If LPARAM points to a string equal to “ConvertibleSlateMode”, then the app should call GetSystemMetrics(SM_CONVERTIBLESLATEMODE). A “0” returned means it is in tablet mode. A “1” returned means it is in non-tablet mode (Figure 10).

```c
... 
3 // Process system setting update. 
4 //-------------------------------
5 case WM_SETTINGCHANGE: 
6 ... 
```
As good practice, Bezier_MTI includes an override button to manually set the device mode. The button is displayed as a Question Mark (Figure 11) at application startup; then changes to a Phone icon (Figure 12) or a Desktop icon (Figure 13) depending on the device mode at the time. The user is able to touch the icon to manually override the detected display mode. The application display changes according to the mode selected/detected. Note that in this sample, the mode annunciator is conveniently used as a manual override button.
Figure 11: Device status unknown.

A phone icon is displayed in tablet mode.

Figure 12: Note the larger control points.
A desktop icon is displayed in non-tablet mode.

![A desktop icon is displayed in non-tablet mode.](image)

**Figure 13**: Note the smaller control points.

**How do I notify my application of a device configuration change?**

Most of the changes in this sample are graphics related. An adaptive UI should also change the nature and the number of the functions exposed to the user (this is not covered in this sample).

For the graphics, you should disassociate the graphics rendering code from the management code. Here, the drawing of the Bezier and other UI elements are separated from the geometry data computation.

In the graphics rendering code, you should avoid using static GDI objects. For example, the pens and brushes should be re-created each time a new drawing is performed, so the parameters can be adapted to the current status, or more generally to any sensor information. If no changes occur, there is no need to re-create the objects.

This way, as in the sample, the size of the UI elements adapt automatically to the device configuration readings. This not only impacts the color, but also the objects' size. Note that the system display's DPI (dots per inch) should be taken in account during the design of this feature. Indeed, small form factor devices have high DPI. This is not a new consideration, but it becomes more important as device display DPI is increasing.

In our case, we decided to re-use the size change mechanism that we implemented for the ALS support (Figure 14). We do this simply by setting the UI objects' size to the minimum when the system is in non-tablet mode and to the maximum when it is in tablet mode (by convention, the unknown mode maps to the non-tablet mode).

```c
1    ... 
2    ret = GetSystemMetrics(SM.ConvertibleSlateMode);
3    if(ret == 0) {
```
data._2_in_1_data.device_configuration = DEVICE_CONFIGURATION_TABLET;
//------------------------------------------------------------
shared_data.lux = MAX_LUX_VALUE;
shared_data.light_coefficient = NORMALIZE_LUX(shared_data.lux);
}
else {
    data._2_in_1_data.device_configuration = DEVICE_CONFIGURATION_NON_TABLET;
//---------------------------------------
    shared_data.lux = MIN_LUX_VALUE;
    shared_data.light_coefficient = NORMALIZE_LUX(shared_data.lux);
}

Figure 14: Code example for changing the UI.

The following code (Figure 15) shows how a set of macros makes this automatic. These macros are then used in the sample’s drawing functions.

#define MTI_SAMPLE_ADAPT_TO_LIGHT(v) ((v) + ((int)(shared_data.light_coefficient * (double)(v))))
#define MTI_SAMPLE_LIGHT_COLOR_COEFFICIENT (1.0 - shared_data.light_coefficient)
#define MTI_SAMPLE_ADAPT_RGB_TO_LIGHT(r, g, b) RGB((int)(MTI_SAMPLE_LIGHT_COLOR_COEFFICIENT * ((double)(r))),
    (int)(MTI_SAMPLE_LIGHT_COLOR_COEFFICIENT * ((double)(g))),
    (int)(MTI_SAMPLE_LIGHT_COLOR_COEFFICIENT * ((double)(b)))

Figure 15: Macro example.

Conclusion

Windows 8 and Windows 8.1 user interface allows developers to customize the user experience for 2 in 1 devices. The device usage mode change can be detected, and the application interface changed dynamically, resulting in a better user experience for the user.

About the Authors

Jamel Tayeb is the architect for the Intel® Energy Checker SDK. Jamel Tayeb is a software engineer in Intel’s Software and Services Group. He has held a variety of engineering, marketing and PR roles over his 10 years at Intel. Jamel has been worked with enterprise and telecommunications hardware and software companies in optimizing and porting applications for/to Intel platforms, including Itanium and Xeon processors. Most recently, Jamel has been involved with several energy-efficiency
Jamel earned a PhD in Computer Science from Université de Valenciennes, a Post-graduate diploma in Artificial Intelligence from Université Paris 8, and a Professional Journalist Diploma from CFPJ (Centre de formation et de perfectionnement des journalistes – Paris Ecole du Louvre).

Stevan Rogers has been with Intel for over 20 years. He specializes in systems configuration and lab management and develops marketing materials for mobile devices using Line Of Business applications.
**Notices**

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A “Mission Critical Application” is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL’S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS’ FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENCE IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: [http://www.intel.com/design/literature.htm](http://www.intel.com/design/literature.htm)

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others

Copyright© 2013 Intel Corporation. All rights reserved.

**Optimization Notice**


**Performance Notice**

For more complete information about performance and benchmark results, visit [www.intel.com/benchmarks](http://www.intel.com/benchmarks)