CustomUI Sample

Collapse All

This sample is a simple demonstration of the UI subsystem implemented by DXUT. DXUT provides the common code and infrastructure found in most Direct3D applications. One area of DXUT is user interface support. DXUT contains commonly used control objects, such as buttons and check boxes, that both windowed and full screen Direct3D applications can use to implement their user interfaces.

Path

Source	*SDK root*\Samples\C++\Direct3D\CustomUI
Executable	*SDK root*\Samples\C++\Direct3D\Bin*x86 or x64*\CustomUI.exe

How the Sample Works

This sample demonstrates how an application that uses DXUT can take advantage of the framework's user interface support. DXUT supports the following control types:

Control type	Class that implements it
Statics	CDXUTStatic
Buttons	CDXUTButton
Radio buttons	CDXUTRadioButton
Check boxes	CDXUTCheckBox
Combo boxes	CDXUTComboBox
Sliders	CDXUTSlider
Edit boxes	CDXUTEditBox
Edit boxes with IME capability	CDXUTIMEEditBox

All of the DXUT controls work similarly to Windows controls. They are implemented entirely from scratch and do not use HWND or any Windows control underneath. They are also rendered with Direct3D, and therefore can be used by both windowed and full screen Direct3D applications.

The CDXUTIMEEditBox is an edit box with IME (Input Method Editor) functionality. When entering text with CDXUTIMEEditBox, the user can switch input language to one of the East Asian languages supported (Japanese, Korean, Simplified Chinese, and Traditional Chinese). When typing, CDXUTIMEEditBox renders IME windows with Direct3D, and it also supresses the default IME windows so that they will not corrupt the application window in full screen mode.

This sample starts by defining two CDXUTDialog objects: g_HUD and g_SampleUI. A CDXUTDialog serves as a container that encapsulates one or more controls. It sits between the application and the controls, so that the application can pass messages and rendering calls to CDXUTDialog, and CDXUTDialog will ensure that all of its controls receive the messages and get rendered properly. The controls themselves are designed to be used with a CDXUTDialog, not stand-alone. Applications are not limited to one CDXUTDialog object. They can use more at once, as this sample does.

The first thing that the sample must do to initialize the CDXUTDialog objects is set the event callback function with the SetCallback method. As the user interacts with the UI controls, the controls can notify the sample about important events, so that it can respond accordingly. At this time the sample also initializes additional fonts that it will need. This is done with the SetFont method.

The next step that the sample has to do is create the controls it needs. This is achieved by calling the

CDXUTDialog's AddXXX methods, such as AddButton. In the sample, g_HUD contains 4 buttons that provide standard functionalities that Direct3D samples have: view whitepaper, toggle between full screen and windowed mode, toggle between HAL and REF, and change 3D device settings. g_SampleUI contains the rest of the controls, which are sample-specific. After a control is added, its appearance can be modified. This is demonstrated in the sample as it modifies the appearance of the static controls as well as the combo box and IME-capable edit box.

Next, in OnResetDevice, are a number of calls to the CDXUTDialog's and controls' SetSize and SetLocation methods. This is because when the user resizes the application window, some dialogs and controls need to be resized and repositioned as well. It is worth noting that the CDXUTDialog's location is relative to the application window, while a control's location is relative to the CDXUTDialog it belongs.

To render the user interface, the sample calls the OnRender method of the two CDXUTDialog objects in its OnFrameRender function. CDXUTDialog will render the controls it contains in the correct order.

The final step required to make the UI work is passing messages to the controls. This lets them receive keyboard and mouse events from the user. In MsgProc, the sample calls CDXUTDialog's MsgProc method to pass down the messages. It is very important that if CDXUTDialog's MsgProc returns true indicating that the message has been handled, the sample sets *pbNoFurtherProcessing to true and returns immediately. Failure to do so will cause a message to be processed more than once, and this can cause undesirable behavior for certain Input Method Editor (IME) messages, such as double windows.

Event Callback

Certain events happen as the user interacts with the UI controls, such as checking or unchecking a check box, selecting a new item in a combo box, selecting a new item in a radio button group, etc. The application may be interested in such events, and it can be notified with the CDXUTDialog event callback mechanism. In the sample, a callback function, OnGUIEvent, is defined and passed as the parameter to the CDXUTDialog::SetCallback method. As a result, this function will get called whenever the user performs an interesting action with a UI control. The actions that can trigger a callback are:

- Clicking a button (CDXUTButton)
- Selecting a new item in a combo box (CDXUTComboBox)
- Selecting a new item in a radio button group (CDXUTRadioButton)
- Changing the check state of a check box (CDXUTCheckBox)
- Changing the value of a slider (CDXUTSlider)
- Pressing Enter in an edit box (CDXUTEditBox and CDXUTIMEEditBox)
- In the callback, the type of event, the ID of the control that fired the event, and the control object are available to the application. The sample responds to the callback by retrieving content of the firing control and displaying it on screen.

Further Reading

For more information please refer to the technical articles entitled Installing and Using Input Method Editors and Using an Input Method Editor in a Game.