How to simulate pointer events

Overview

The wiki article Simulating Pointer Events introduced a way to simulate pointer events within an application. However, sometimes a program may need to send pointer events to other applications. This article describes how to simulate pointer events by `RWsSession::SimulateRawEvent()`, and explains why it is impossible to send pointer event to a background application by `RWsSession::SendEventToWindowGroup()`.

Solution

The `RWsSession::SimulateRawEvent()` can be used to simulate pointer events, i.e. send pointer events to the foreground application. Assume that the phone is in portrait mode, then the following code can simulate a touch operation on the "Options" CBA button area. If the foreground application has the "Options" CBA button then its options menu will be popped up.

```
TPoint p(60,600); // within the "Options" CBA button area
TRawEvent event;
event.Set(TRawEvent::EButton1Down, p.iX, p.iY);
iCoeEnv->WsSession().SimulateRawEvent(event);  // SwEvent
User::After(1000);
event.Set(TRawEvent::EButton1Up, p.iX, p.iY);
iCoeEnv->WsSession().SimulateRawEvent(event);  // SwEvent
```

Source Code

Full example:

HelloWorld(SimulatePointerEvent).zip

How to use it on emulator:

1. Build the application for winscw and then start the emulator
2. Start the application and then press the "Multi-task" key (also called the Menu key),

How to use it on target:

1. Sign the HelloWorld_gcce.sis with a certificate that can grant SwEvent capability
2. Install the signed installation package on the phone and then start it.
3. Press the "Multi-task" key

You can see the app sends itself to background and then the foreground application (the Main menu) pops up its options menu.

M.I.P.

Many developers tried to send pointer events to a selected application by `RWsSession::SendEventToWindowGroup()`, but always the target application crashed with KERN-EXEC 3 panic.

To know why it is a Mission Impossible to send pointer events in this way, we shall first understand the convention between CONE and Window Server:

1. When a window-owning control (`CCoeControl`) is constructed the address of the control is used as the handle of the window.
2. When there is a pointer event the Window Server decides which window shall handle it, and then assigns the window handle to the event.iHandle, and then sends the event to the app that owns the window.
3. When the app CONE received the event, it casts the event.iHandle to a CCoeControl pointer and then call pointer->HandlePointerEventL(event);

So if the code looks like this:

```
TWsEvent event;
event.SetType(EEventPointer);
event.SetHandle(0); // "what the handle is? Anyway clear it..."
TPointerEvent& pointerEvent = *(event.Pointer());
pointerEvent.iType = TPointerEvent::EButton1Down;
pointerEvent.iParentPosition = TPoint(0,0);
pointerEvent.iPosition = TPoint(10,10);
pointerEvent.iModifiers = 0;
ws.SendEventToWindowGroup(...); // // send the event to app1
```

When the app1 receives the event the bad thing will happen:

```
// CONE code
…
CCoeControl* ctrl = reinterpret_cast(event.Handle());
ctrl->HandlePointerEventL(*(event.Pointer())); // KERN-EXEC 3 because ctrl equals NULL
```

The Window Server is in charge of all the windows so it knows what the iHandle should be, but you don't know, so "Window Server's score:100. Yours:0!"

PS: `RWindowGroup::Child()` can return the client handle but we can not take it for granted that it is a `CCoeControl` pointer.

Reference

The CONE source code included in ER5 SDK (log-in required).