Creating User Interfaces On Remote Devices

Using Bluetooth

Richard Hoptroff

Published (in edited form) in
Dr Dobb's Journal,Summer 2004

Creating User Interfaces On Remote Devices Using Bluetooth

The FlexiPanel Bluetooth Protocol is a remote user interface service for computers, electrical appliances and other machinery. A FlexiPanel server resides on the application and holds a user interface database that reflects the appliance's human-machine interface needs. A FlexiPanel client can connect at any time, read the database and displays the user interface. A user may then control the application from the client device. Using Bluetooth, the client can be up to 330 feet away, without need for line-of-sight communication. FlexiPanel clients have been implemented on a range of PDAs and cellphones and are freely available.

Like many higher-level protocols such as OBEX file exchange, FlexiPanel sits on top of the RFCOMM serial port emulation layer of the Bluetooth protocol stack (Figure 1). It is not part of the "official" Bluetooth standard. However, the standard is relatively open in that anyone is free to create FlexiPanel clients, and FlexiPanel server licenses are royalty-free.

FlexiPanel was first developed in 2002 as an IrDA infrared protocol for engine tuning in hobbyist racing cars. The engine was controlled by a microcontroller which needed to be fine-tuned with a number of preset values. Incorporating an onboard user interface (such as a LCD display and a keyboard) to enter the tuning

parameters was impractical: it would be too expensive, heavy, bulky and fragile. At the time, personal digital assistants (PDAs) with infrared capability were becoming become popular, so it was logical to use one for the user interface instead. The FlexiPanel Protocol was thus conceived as a way to let embedded systems create user interfaces on any suitable device that a user might have to hand.

The protocol's limitations, and its wider potential, soon became apparent. IrDA might be wireless, but in practice, the communicating devices must be held steady within a one foot (30cm) range of each other and within line of sight. Emerging radio technologies such as Wi-Fi, Bluetooth and ZigBee (a low power protocol) offered 330 foot (100m) range. In addition, they needed no exterior real estate, a tremendous product design advantage in terms of cost, æsthetics and reliability in a hostile environment.

We decided to migrate the technology to Bluetooth, on the grounds that it was widely implemented on remote devices and it suited the *ad hoc* nature of the connection between appliances and remote handhelds. In addition, Intel had just committed to incorporating Bluetooth into its Centrino 2 chipset. Wi-Fi and ZigBee remain viable transport layers, though at present they are not implemented widely enough on handheld devices.

At the same time, it became clear that the FlexiPanel Protocol's potential went far beyond engine tuning. It provides a Human-Machine Interface for *smart dust* (intelligent devices comprising a microcontroller and a few external components), thus far extending their range of application. It offers a user-interface to "headless" (*i.e.* no screen or keyboard) single board computers. This promises cost reductions for point-of-sale systems, computer operated machine tools and so on, while freeing the operator from the control panel location. In traditional Windows applications, it offers the possibility of a supplementary, roaming user interface. Though rarely vital to a Windows application, the roaming capability is a liberating addition to any application that might have anything to say to the user while away from his desk. *(How's my stock price? Do I have new mail? Is it going to rain?)*

As with all emerging technologies, early Bluetooth devices suffered from clunky operation and compatibility problems. This was particularly true of service discovery, *i.e.* finding out what Bluetooth devices are in range and the services they provide. Many 2002-era devices were limited to pre-connection over a virtual serial port, rather than providing service discovery APIs. In the last year, Microsoft's support for Bluetooth in Windows Sockets has made service discovery a one-click process that even Amazon boss Jeff Bezos would be proud of.

A User Interface Service

From the application's perspective, FlexiPanel is a graphical user interface service just like any other. It can create controls and subsequently change the properties of those controls. If a user modifies a control, the application is sent a notification message. The application doesn't need to know that the user interface is displayed remotely. Conversely, just like a web browser, the FlexiPanel client is generic and does not need to know anything about the server it is providing a user interface for. (Client software is generally free, too.)

The types of control that may be created are listed in Table 1. Under the hood, the FlexiPanel Protocol is based on just twelve basic types of message passed between client and server (Table 2).

The main differences between the FlexiPanel Protocol and regular user interface services are:

- The nature of the client's user interface may be unknown. The controls displayed will always be logically correct, but appearances may vary between different client devices. Compare the same slideshow controller user interface on a PDA (Figure 2) and a cellphone (Figure 3). If the client device can be anticipated in advance, certain additional preferences can be requested, such as a particular control

layout (Figure 2) or keyboard accelerators (Figure 3).

- The connection might be broken at any time, for example if the client's batteries fail or the client goes out of range. The appliance must enter a fail-safe state if connection is lost at a critical moment.

- FlexiPanel servers might be very small, low cost devices, such as *Parallax Inc*'s FlexiPanel peripheral for BASIC Stamps, based on a low-end 8-bit microcontroller (Figure 4). Consequently system requirements on the server side must be extremely lean and communication very succinct (unlike XML!). The remote client device takes over as many responsibilities as possible. For example, a server is not required to buffer any I/O, manipulate any floating point numbers or make any conversions between single-byte characters and Unicode.

Bluetooth is a multi-point communication protocol. It is possible for a FlexiPanel server to manage user interfaces on up to seven remote devices at once. This is only implemented on high-end systems such as Windows servers but it is useful for applications such as restaurant table service, where several wait-staff can connect to the same server at once. A quick-disconnect mode is also available, where the server closes the connection immediately after connection has been established and control panel information sent. This allows a server to send user

interfaces to a large number of clients connecting asynchronously, although the clients have no chance to send any messages back to the server.

The FlexiPanel protocol plays no role in authentication, encryption, error detection or power management. These are expected to be managed by other layers of the Bluetooth protocol.

Embedded Systems Example

The following example uses the FlexiPanel server C library to create a remote control panel for a data-logging embedded system. The library is intended to work with the lowest-level embedded controllers possible. Therefore no assumptions are made within the library about serial port buffering or multi-threading support.

The data logger records the value of a proximity sensor. It has no user interface of its own and relies on FlexiPanel to communicate with the outside world. The distance measured is displayed as a digital readout on a number control and as a historical log in a matrix control.

The data logger (Figure 5) is based on a 80186-based *FlashCore* embedded controller from *Tern Inc*. A *GP2D12* analog proximity sensor (*SHARP Electronics*) is connected to an A/D input and serves as the data source. A

BlueWAVE Bluetooth serial module from *Wireless Futures Ltd* is connected to a serial port to enable connection to FlexiPanel remote clients.

The data logger's job is simple and few lines of code are required to implement it (Listing 1). All calls to the FlexiPanel server library begin with the prefix *FBVS*. Like most FlexiPanel applications written for embedded controllers, it consists of fixed sections: *initialization, user interface definition, main program loop, client message processing* and an *ungraceful disconnect handler*.

Initialization. The Bluetooth stack and FlexiPanel server are initialized. This includes giving the user interface a name. This name appears in a client's user interface in the list of servers available for connection.

User Interface Definition. Sandwiched between calls to *FBVSStartControlList* and *FBVSPostControls*, each control is defined by a call to an *FBVSAdd…* function. Each time *FBVSPostControls* is called, the previous user interface definition is replaced; the user interface can thus be changed at any time. *FBVSSetOption* permits an application to request how a control is depicted on a specific remote device. In this listing, a point-plot chart is specified if the client is a Pocket PC or a Windows computer.

Main Program Loop. In the main program loop, the FlexiPanel library is polled for *messages from client devices*. Every second, the sensor is sampled and the

sampled value written to both the digital readout and historical log controls. In addition, every few seconds a *ping* test is made to check whether a client device is still in range. If a client was connected but contact was unexpectedly lost, control is passed to the *ungraceful disconnect handler*.

Client Message Processing. When client-related events occur, the FlexiPanel server library posts notification messages to the application (Table 3). The data logger uses the *FBVSGetNotifyCode* function to collect these messages. Most notifications are informational and useful only for providing a local indication of the connection status. The four messages which the developer should always consider are:

- *FBVSN_ClientConnected*. A remote device connected to the server. In this example, the server clears the matrix control of old data.

- *FBVSN_ClientData*. The remote device modified a control, *e.g.* the user pressed a button. In this particular application, neither of the controls is modifiable by the client and so nothing need be done. Usually, however, the application would be expected to respond to any user interaction at this point.

- *FBVSN_ClientDisconnected*. A remote device disconnected from the server. In this example, the message is ignored.

.

- *FBVSN_Abandon*. An error occurred. This message has only ever been witnessed during application device development due to a readily apparent programming error. In this example, as a precaution, control is passed to the *ungraceful disconnect handler* if this message is received.

Ungraceful Disconnect Handler. The application must provide for the possibility that the connection is lost unexpectedly. This might occur because the remote device has gone out of range or its batteries are worn out. In this example, no action is necessary. In applications controlling machinery, an emergency stop procedure should be implemented.

Windows .NET Example

The *OrderMaster* console application uses the FlexiPanel .NET library to create a remote control panel for a restaurant order-taking system. A Windows server computer with a printer is located near the kitchen and prints out the orders for the kitchen and the checks for the customers. It needs to be Bluetooth equipped, and so will require a USB Bluetooth adapter (ideally a class 1 adapter with a 330 foot range).

The wait-staff carry remote devices for taking orders. The application is targeted specifically for Pocket PC clients. The user interface has been designed using a *fat thumbs* approach, where the controls are large enough that no stylus is required.

In keeping with the FlexiPanel philosophy, however, any FlexiPanel client would be able to connect. A waiter who lost his Pocket PC could always use his cellphone to take the order. Indeed, it would be possible for customers to place orders and print out their checks themselves if they had an appropriate device.

The *OrderMaster* main screen is shown in Figure 6. At the top is a text control for displaying a summary of the current order. Below it are section controls which drill down to specific sections of the menu. At the bottom is a list control to select the table being served and buttons to order the check, clear the order and confirm the order.

The *Appetizers* sub-screen is shown in Figure 7. At the top is the section control which returns to the main screen. Below it are number controls for setting order quantities for individual items on the menu.

The code required to implement *OrderMaster* is shown in Listing 2. Since much of the code is repetitive and would in practice be replaced with loops working from a menu database, code for sub-screens other than the appetizers has not been implemented.

The FlexiPanel .NET namespace is called *RCapiM*. It consists of static function calls for management of the user interface service and classes for each of the twelve control types. Like the data logger, this application consists of the same

five sections: *initialization*, *user interface definition*, *main program loop*, *client message processing* and *ungraceful disconnect handler*.

Initialization. The *_tmain()* entry point in midway through Listing 2 begins by initializing the FlexiPanel library, setting up the Bluetooth port, giving the user interface a name and setting up a timer to ping the remote device regularly. Since a Pocket PC client is anticipated, several options are requested specifically for it. It should hide its usual navigation buttons and it should regularly ping the server (the computer, that is, not the waiter!)

User Interface Definition. The user interface is defined by creating controls and attaching them to an *ArrayList* called *RemoteForm*. The *RemoteForm* is then sent to the *FlexiPanel::PostControls* static function in order to display the control panel. If the application needs to respond when the user modifies a control, a delegate is added to the *OnClientModify* event for that control. Since the delegate will be called from a different thread to the main thread, the delegate contains static pointers to controls it needs to access. A delegate is also added to the static *OnClientNotify* event so that the order can be cleared if a *Client Disconnected* notification is received (refer to Table 3).

Main Program Loop. In the main program loop, nothing happens except for waiting for the instruction to quit. All further activities are in response to events.

Client Message Processing. The following delegates respond to events and are managed within the *EvtHandler* struct.

- *OnClientNotify*. If the client notifies it is disconnecting, the order is cleared.

- *OnButCheck*. If the Check button is pressed, the check is printed.

- *OnButClear*. If the Clear button is pressed, all order quantities are set to zero.

- *OnButConfirm*. If the Confirm button is pressed, the order is printed for the kitchen and then all order quantities are set to zero.

- *OnCtlModify*. If the Table list box is modified or one of the menu section controls is opened or closed, the status text box is updated.

- *OnPingTimer*. The client is pinged. If contact is lost, an *Ungraceful Disconnect Handler* procedure is followed.

Ungraceful Disconnect Handler. The application clears the order and waits for the client to reconnect.

Future Developments

Born almost by accident while trying to optimize engine performance, the FlexiPanel Protocol has evolved into a patented user interface service for a range of devices from tiny microcontrollers to .NET applications. Future development plans include:

- Embedding the protocol directly inside Bluetooth chipsets. This will lower the cost sufficiently that even basic electrical appliances such as light switches can create remote user interfaces.

- Provision of a local Bluetooth to HTTP bridge so that a web browser anywhere in the world might connect to a FlexiPanel server.

- Provision of a local Bluetooth to telecoms bridge so that a FlexiPanel server can accessed by anyone dialing in using a touch-tone phone.

Figures and Tables

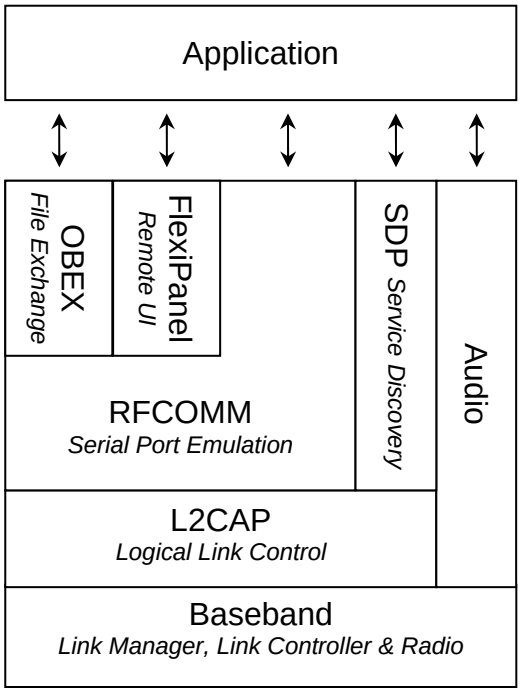

Figure 1 – FlexiPanel in the Bluetooth Protocol Stack

(Available as a Microsoft Word Drawing)

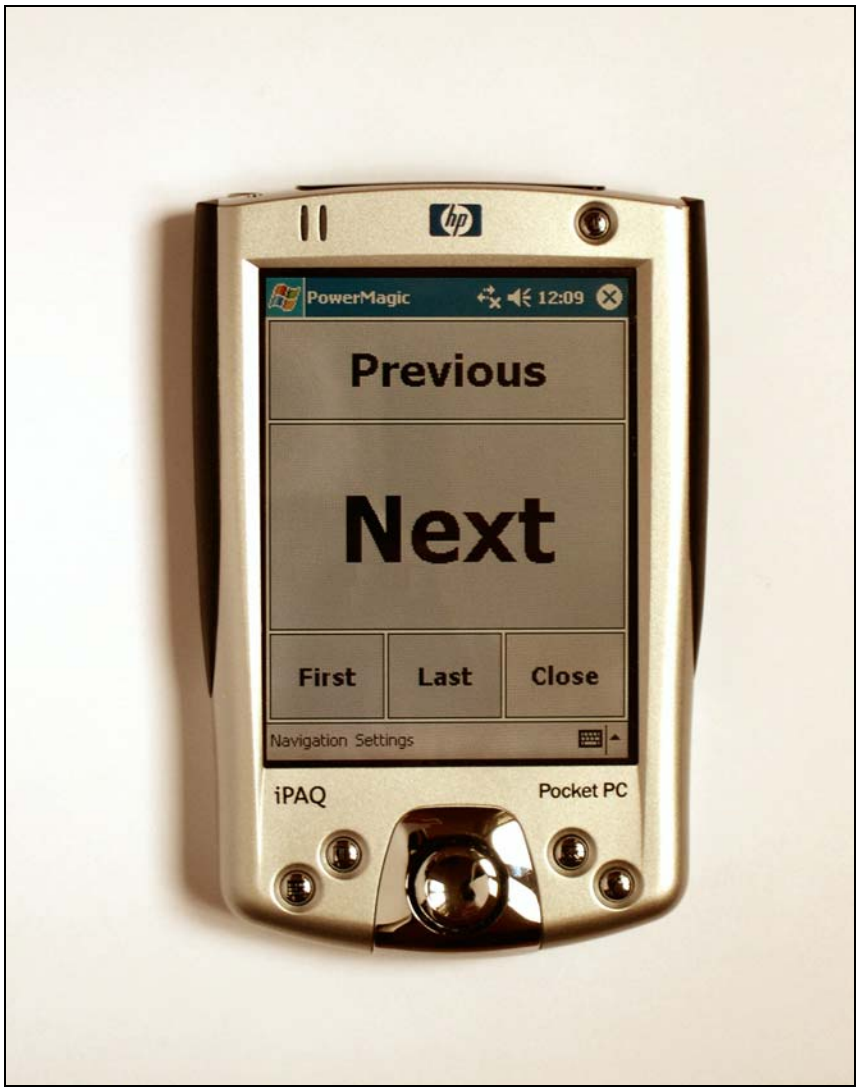

Figure 2 – Powerpoint presentation controller user interface on a Pocket PC

(Available as DDJPPCPmg.jpg 1620 x 2058 pixels)

Figure 3 – Powerpoint presentation controller user interface on a Smartphone

(Available as DDJPPCPmg.jpg *1704 x 2272 pixels)*

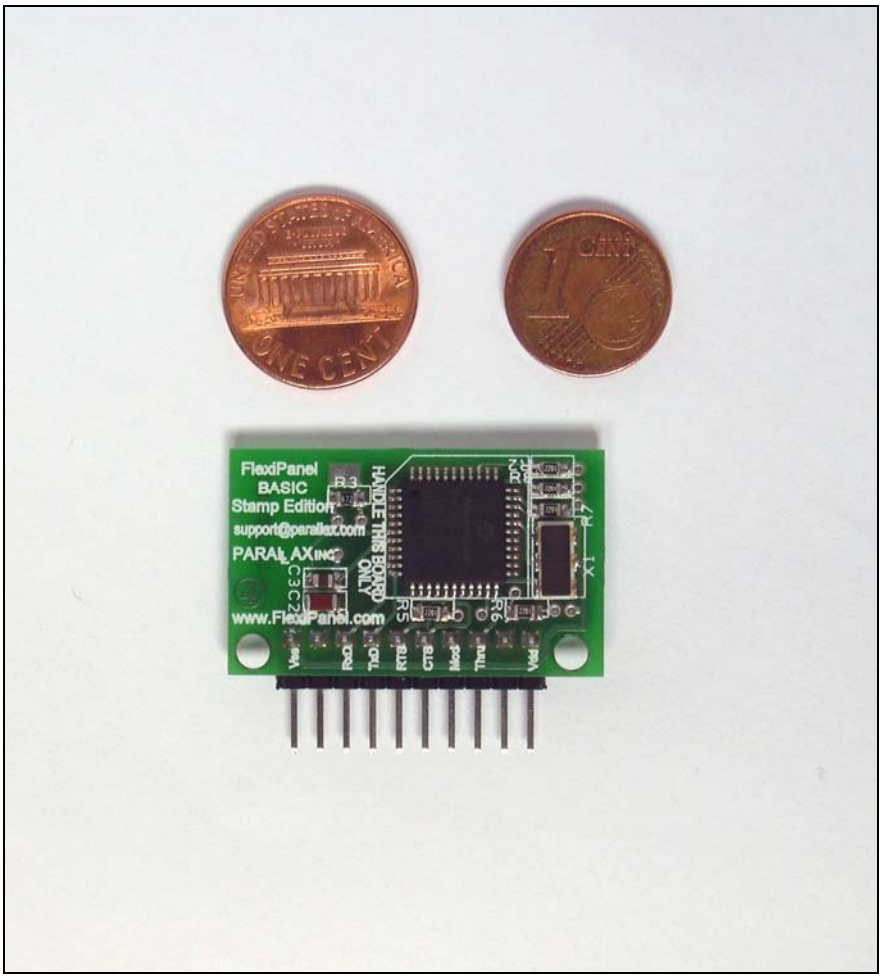

Figure 4 – FlexiPanel server implemented on an 8-bit microcontroller.

The Bluetooth radio is mounted on the reverse side of the board.

(Available as DDJFxPPIC.jpg *1363 x 1509 pixels)*

Figure 5 – Data logging embedded system using the FlexiPanel protocol to create user interfaces on Pocket PCs.

(Available unlabeled as DDJDataLog.jpg *2579 x 1651 pixels.*
Composite photo; one PDA in reality
pre-composite originals also available)

Figure 6 – Main order screen in *OrderMaster* application.

(Available as DDJOrdMst1.jpg, *1704 x 2272 pixels)*

Figure 7 – Appetizers sub-screen in *OrderMaster* application.

(Available as DDJOrdMst2.jpg, *1704 x 2272 pixels)*

Logical control	Example depiction on a remote client	Function / value
Button	Button	Single-press event
Latch	Check box Radio button	Binary value
Text	Static text Edit text	Character string
Number	Progress bar Slider	Integer or fixed-point value
Matrix	Table Column chart Line chart	2-D array of numeric values
Date Time	Date time picker	Seconds to years plus day of week
List	List box	1-of-*n* selection
Section	Popup menu	Arranges controls in a hierarchy
Password	Client-specific dialogs	Controls access to user interface
Message	Message box	Alerts user
Blob*	Client-specific dialogs	Exchanges binary data
Files*	Client-specific dialogs	Exchanges files

* Optional. A client device is not required it implement this control.

Table 1 – Controls provided by the FlexiPanel protocol.

(Footnote to be included with table.)

Message	Originator (Client / Server)	Purpose
Greetings	Either	Establishes a connection
Goodbye	Either	Closes a connection
New Control Panel	Server	Sends control descriptions to client
Control Modified	Either	Modifies a control's value
Ping	Either	Presence check request
Ping Reply	Either	Presence check confirmation
Ack	Either	Acknowledge receipt of message
New Server	Server	Server initializing
Props Update	Server	Modifies a control's properties
Files	Server	Downloads requested files
Profile Request	Client	Requests device-specific layout advice
Profile Reply	Server	Downloads device-specific layout advice

Table 2 – FlexiPanel Client-Server message types.

Notification	Meaning
No Notify	No notable events have happened
Client Connected	A client connected to the server
Client Data	The client modified a control
Client Disconnected	The client disconnected from the server
Got Pinged	Client pinged server and server pinged back
Got Ping Reply	Server pinged client and client pinged server back
Got Ack	Client acknowledged receipt of message
Got Profile Request	Client asked how to lay out controls and server replied

Table 3 – Notification messages that a FlexiPanel server can send to an application.

Listing 1 – Embedded data logger application

```
// Bluetooth module is connected to serial port SER1
extern COM ser1_com;
#define BTH_BAUD    12                    //  115,200 baud for SER1
#define BTH_INBUFF  1024                  //  SER1 Input buffer size
#define BTH_OUTBUFF  1024                 //  SER1 Output buffer size
unsigned char ser1_in_buf[BTH_INBUFF];    //  SER1 Input buffer
unsigned char ser1_out_buf[BTH_OUTBUFF];  //  SER1 Output buffer


// UI constants
#define CID_ATOD      1
#define CID_CHART_TY  2
#define NUM_CTRL      2
#define NUM_OPTION    2


// function prototypes
void ProcessFlexiPanelMessages(void);
void HaltProcesses(void);


void main(void)
{
    int16 loopcount;

    // initialize serial port (calls serial I/O library)
    s1_init( BTH_BAUD, ser1_in_buf, BTH_INBUFF,
             ser1_out_buf, BTH_OUTBUFF, &ser1_com );

    // initialize FlexiPanel library and give the UI a name
    FBVSInit( NULL, NULL, 1, &ser1_com, NUM_OPTION );
    FBVSSetDevNameAndCharSet( "Tern Demo" );

    // initiate control panel description
    FBVSStartControlList( NUM_CTRL );

        // numeric display of distance
        FBVSAddNumber( CID_ATOD, CTL_NUM_FIXEDPOINT, "Range", 0, 0, 0,
                 2, -2, "%% m", NULL);

        // graphical display of distance log
        FBVSAddMatrix(CID_CHART_TY, CTL_MTX_DATA_TY | CTL_MTX_Y_FIXEDPOINT |
                CTL_MTX_Y_2BYTE, "Data Log", 30, 0, NULL, 1, NULL, "Range",
                "Time", "%% m", "%HH%:%mm%:%ss%", 2, -2, 0, 0, NULL);

        // suggest how chart might be displayed
        FBVSSetOption( PPC_DEV_ID, CID_CHART_TY, PPC_ATT_STYLE,
                PPC_CST_MATRIX_POINTS );
        FBVSSetOption( WIN_DEV_ID, CID_CHART_TY, WIN_ATT_STYLE,
                WIN_CST_MATRIX_POINTS );

    // complete control panel description
    FBVSPostControls( );

    // start control panel service
    FBVSConnect();

    // main program loop
    loopcount = 0;
    while (1)
    {
        // ensure each loop takes around 10ms
        delay_ms(10);

        // discover whether client has sent any messages
        ProcessFlexiPanelMessages();
```

```
        // every 3 seconds, ping
        loopcount ++;
        if (loopcount == 300)
        {
            loopcount = 0;

            // ping
            if ( FBVSIsClientConnected() && FBVSIsPingSupported() && FBVSPing() )
            {
                // lost contact with remote device; continue cautiously
                HaltProcesses();
            }
        }

        // every second, log proximity sensor
        if (loopcount%100==0)
        {
            int16 range;
            DateTimeU dt;

            // read A/D (calls A/D library)
            range = fb_ad16( 0xc6 );

            // update numeric display
            FBVSSetNumberControlData( CID_ATOD, range );

            // update chart
            SetToCurrentTime( &dt );
            FBVSAddMatrixControlData( CID_CHART_TY, &range, &dt );

            // send updated time to client
            FBVSUpdateControlsOnClient( );
        }
    }
}


void ProcessFlexiPanelMessages(void)
{
    // check for message
    switch (FBVSGetNotifyCode())
    {
        // nothing has happened
        case FBVSN_NoNotify:
            break;

        // Client has connected; clear the contents of the matrix control
        case FBVSN_ClientConnected:
            FBVSSetMatrixControlData( CID_CHART_TY, 0, NULL, NULL );
            break;

        // Client has modified a control.  in this app, no controls are
        // modifiable by the client, so nothing to do
        case FBVSN_ClientData:
            break;

        // Client has disconnected
        case FBVSN_ClientDisconnected:
            break;

        // following messages are informational only and will be ignored
        case FBVSN_GotProfileRequest:
        case FBVSN_GotPinged:
        case FBVSN_GotPingReply:
        case FBVSN_GotAck:
        case FBVSN_IncompatibleVersion:
             break;

        case FBVSN_Abandon:
            // Error; generally only gets here during development
```

```
            HaltProcesses();
            Reset();
            break;

    }
}


void HaltProcesses(void)
{
   // In this function, anything controlled by the embedded
   // controller is put in a fail-safe state.

   // nothing being controlled in this app, so nothing to do
}
```

Listing 2 – *OrderMaster* application

```
// OrderMaster.cpp

#include "stdafx.h"

#using <mscorlib.dll>
#using <RCapiM.dll>
#include <tchar.h>
#include <math.h>
#include <stdlib.h>
#using <System.Drawing.dll>

// FlexiPanel constants
#include "HopCodes.h"
#include "PocketPCProfiles.h"

using namespace System;
using namespace System::Drawing;
using namespace System::Collections;
using namespace System::Threading;
using namespace RCapiM;

public __gc struct EvtHandler
{
public:
  // pointers to controls that event handler needs access to
  // order quantities other than appetizers omitted to save listing space
  static RemoteNumber* NumMixedSalad;
  static RemoteNumber* NumCaesarSalad;
  static RemoteNumber* NumChowder;
  static RemoteNumber* NumSeafood;
  static RemoteNumber* NumOysters;
  static RemoteNumber* NumSatay;

  static RemoteList* ListTable;
  static RemoteText* TextStatus;

  // set all order quantities to zero
  static void ClearOrder( void )
  {
    NumMixedSalad->SetVal( 0, false );
    NumCaesarSalad->SetVal( 0, false );
    NumChowder->SetVal( 0, false );
    NumSeafood->SetVal( 0, false );
    NumOysters->SetVal( 0, false );
    NumSatay->SetVal( 0, true );
    OnCtlModify( NULL );
  }

  // print out order for cook
  static void PrintOrder( )
  {
    // should print to printer rather than console...
    Console::WriteLine( S"Order table {0}",  __box(ListTable->GetSel()+1));
    if (NumMixedSalad->GetVal())
        Console::WriteLine( S"{0} Mixed Salad", __box(NumMixedSalad->GetVal()));
    if (NumCaesarSalad->GetVal())
        Console::WriteLine( S"{0} Caesar Salad", __box(NumCaesarSalad->GetVal()));
    if (NumChowder->GetVal())
        Console::WriteLine( S"{0} Clam Chowder", __box(NumChowder->GetVal()));
    if (NumSeafood->GetVal())
        Console::WriteLine( S"{0} Seafood Salad", __box(NumSeafood->GetVal()));
    if (NumOysters->GetVal())
        Console::WriteLine( S"{0} Oysters", __box(NumOysters->GetVal()));
    if (NumSatay->GetVal())
        Console::WriteLine( S"{0} Chicken Salad", __box(NumSatay->GetVal()));
  }
```

```
  // print check for customer - similar to PrintOrder()
  static void PrintCheck( )
  {
    // ...check would be printed here...
  }

  // something happend at the client
  static void OnClientNotify( NotifyCode iNotifyCode, int iChannel )
  {
    // if client disconnected, set all order quantities to zero
    if (iNotifyCode==ClientDisconnected)
    {
      ClearOrder();
    }
  };

  // Check button pressed
  static void OnButCheck( RemoteControl* rc )
  {
    PrintCheck( );
  };

  // Clear button pressed
  static void OnButClear( RemoteControl* rc )
  {
    ClearOrder();
  };

  // Confirm button pressed
  static void OnButConfirm( RemoteControl* rc )
  {
    PrintOrder( );
    ClearOrder();
  };

  // a control was modified so update status text
  static void OnCtlModify( RemoteControl* rc )
  {
    // calculate number of appetizers ordered
    int numAppetizer = NumMixedSalad->GetVal() + NumCaesarSalad->GetVal() +
           NumChowder->GetVal() + NumSeafood->GetVal() + NumOysters->GetVal() +
           NumSatay->GetVal();

    // update status text
    String* NewText = new String(S"");
    NewText = String::Format(
        S"Table {0}: {1} Appetizers, 4 Entrees, 0 Desserts, 6 Drinks",
        __box(ListTable->GetSel()+1), __box(numAppetizer));
    TextStatus->SetText( NewText, true );
  };

  // implement ping event
  static Timer* pingTimer;
  static void OnPingTimer(Object* stateInfo)
  {
    // ping client
    try
    {
      FlexiPanel::Ping(0);
    }
    catch (FxPPingFailException* e)
    {
      // ping failure, clear order
      // this is the "Ungraceful Disconnect Handler"
      ClearOrder();
    }
  };
};
```

```
// This is the entry point for this application
int _tmain(void)
{
  try
  {
    // Initialize FlexiPanel
    FlexiPanel::Init( NULL, NULL );

    // Set up device
    int chan[1];
    chan[0] = 4;  // Bluetooth port is COM4: on this computer
    FlexiPanel::SetChannels( chan, 1 );
    FlexiPanel::SetDevName( S"OrderMaster" );

    // set timer to ping client every five seconds
    TimerCallback* timerDelegate = new TimerCallback(0, &EvtHandler::OnPingTimer);
    EvtHandler::pingTimer = new Timer(timerDelegate, NULL, 5000, 5000);

    // client is pocket pc expected to be a Pocket PC.
    // suggest pinging every 5 seconds
    FlexiPanel::SetOption( PPC_DEV_ID, PPC_SETTING, PPC_PING_SECS, 5 );

    // remove nav controls at bottom of screen
    FlexiPanel::SetOption( PPC_DEV_ID, PPC_NAV_CLOSE, PPC_ATT_XPOS, -100 );
    FlexiPanel::SetOption( PPC_DEV_ID, PPC_NAV_FIRST, PPC_ATT_XPOS, -100 );
    FlexiPanel::SetOption( PPC_DEV_ID, PPC_NAV_NEXT, PPC_ATT_XPOS, -100 );
    FlexiPanel::SetOption( PPC_DEV_ID, PPC_NAV_PREV, PPC_ATT_XPOS, -100 );
    FlexiPanel::SetOption( PPC_DEV_ID, PPC_NAV_LAST, PPC_ATT_XPOS, -100 );

    // Create remote form
    ArrayList* RemoteForm = new ArrayList();

      // create status text control and add to remote form
      EvtHandler::TextStatus = new RemoteText( 0, 0, S"Status", S"Initializing...",
             128, NULL );
      RemoteForm->Add( EvtHandler::TextStatus );

      // since remote device is probably a Pocket PC, specify preferred layout.
      // This should be implemented for all controls; only shown for this control
      // in order to save listing space
      FlexiPanel::SetOption( PPC_DEV_ID, EvtHandler::TextStatus->GetID(),
            PPC_ATT_XPOS, 2 );
      FlexiPanel::SetOption( PPC_DEV_ID, EvtHandler::TextStatus->GetID(),
            PPC_ATT_YPOS, 2 );
      FlexiPanel::SetOption( PPC_DEV_ID, EvtHandler::TextStatus->GetID(),
            PPC_ATT_XSIZE, 237 );
      FlexiPanel::SetOption( PPC_DEV_ID, EvtHandler::TextStatus->GetID(),
            PPC_ATT_YSIZE, 40 );
      FlexiPanel::SetOption( PPC_DEV_ID, EvtHandler::TextStatus->GetID(),
            PPC_ATT_PAGE, 0 );
      FlexiPanel::SetOption( PPC_DEV_ID, EvtHandler::TextStatus->GetID(),
            PPC_ATT_FONTSIZE, 19 );

      // create appetizer group
      RemoteSection* SecAppetizers = new RemoteSection( 0, CTL_SCT_AUTOCLOSE,
            S"Appetizers", false, NULL );
      RemoteForm->Add( SecAppetizers );
      SecAppetizers->OnClientModify +=
             new ClientModify(0,&EvtHandler::OnCtlModify);

      // Mixed Salad number control
      EvtHandler::NumMixedSalad = new RemoteNumber( 0, CTL_NUM_MODIFIABLE,
            S"Mixed Salad", 0, 0, 0, 0, 0, S"%% Mixed Sld", NULL );
      RemoteForm->Add( EvtHandler::NumMixedSalad );

      // Caesar Salad number control
      EvtHandler::NumCaesarSalad = new RemoteNumber( 0, CTL_NUM_MODIFIABLE,
            S"Caesar Salad", 0, 0, 0, 0, 0, S"%% Caesar Sld", NULL );
      RemoteForm->Add( EvtHandler::NumCaesarSalad );
```

```
  // Clam Chowder number control
  EvtHandler::NumChowder = new RemoteNumber( 0, CTL_NUM_MODIFIABLE, S"Chowder",
        0, 0, 0, 0, 0, S"%% Chowder", NULL );
  RemoteForm->Add( EvtHandler::NumChowder );

  // Seafood Salad number control
  EvtHandler::NumSeafood = new RemoteNumber( 0, CTL_NUM_MODIFIABLE,
        S"Seafood Salad", 0, 0, 0, 0, 0, S"%% Seafood Sld", NULL );
  RemoteForm->Add( EvtHandler::NumSeafood );

  // Mixed Salad number control
  EvtHandler::NumOysters = new RemoteNumber( 0, CTL_NUM_MODIFIABLE, S"Oysters",
        0, 0, 0, 0, 0, S"%% Oysters", NULL );
  RemoteForm->Add( EvtHandler::NumOysters );

  // Chicken Satay number control
  EvtHandler::NumSatay = new RemoteNumber( 0, CTL_NUM_MODIFIABLE, S"Mixed Sal",
        0, 0, 0, 0, 0, S"%% Chkn Satay", NULL );
  RemoteForm->Add( EvtHandler::NumSatay );

RemoteSectionEnd* SecEndAppetizers = new RemoteSectionEnd();
RemoteForm->Add( SecEndAppetizers );

// create Meat dishes group - as appetizers, code omitted to save space;

// create Fish dishes group - as appetizers, code omitted to save space;

// create Veggie group - as appetizers, code omitted to save space;

// create side order group - as appetizers, code omitted to save space;

// create drinks group - as appetizers, code omitted to save space;

// create specials group - as appetizers, code omitted to save space;

// create desserts group - as appetizers, code omitted to save space;

// create table list control
String* sTables[] = { S"Table 1", S"Table 2", S"Table 3", S"Table 4",
        S"Table 5", S"Table 6",  S"Table 7", S"Table 8", S"Table 9",
        S"Table 10" };
EvtHandler::ListTable = new RemoteList( 0, 0, S"Table", sTables, 0, NULL );
RemoteForm->Add( EvtHandler::ListTable );
EvtHandler::ListTable->OnClientModify +=
        new ClientModify(0,&EvtHandler::OnCtlModify);

// create check button
RemoteButton* ButCheck = new RemoteButton( 0, 0, S"Check", NULL ) ;
RemoteForm->Add( ButCheck );
ButCheck->OnClientModify += new ClientModify(0,&EvtHandler::OnButCheck);

// create clear button
RemoteButton* ButClear = new RemoteButton( 0, 0, S"Clear", NULL ) ;
RemoteForm->Add( ButClear );
ButClear->OnClientModify += new ClientModify(0,&EvtHandler::OnButClear);

// create clear button
RemoteButton* ButConfirm = new RemoteButton( 0, 0, S"Confirm", NULL ) ;
RemoteForm->Add( ButConfirm );
ButConfirm->OnClientModify += new ClientModify(0,&EvtHandler::OnButConfirm);

// display control panel
FlexiPanel::PostControls( RemoteForm, 0 );

// subscribe to client message service to pick up disconnect message
FlexiPanel::OnClientNotify += new ClientNotify(0,&EvtHandler::OnClientNotify);

// activate server
FlexiPanel::Connect( false );
Console::WriteLine( S"OrderMaster initialized" );
```

```
    // update status text
    EvtHandler::OnCtlModify( NULL );

    // await command from console
    String* sLine( S"" );
    do
    {
      // process any commands other than quit here
      sLine = Console::ReadLine();
    }
    while (!sLine->StartsWith( S"q" ));

    Console::WriteLine( S"Disconnecting..." );

    // Disconnect
    FlexiPanel::Disconnect( );
  }

  catch (FxPEvalOverException* e)
  {
    Console::WriteLine( e->Message );
  }

  catch (RCapiM::FxPSystemErrorException* e)
  {
    Console::WriteLine( e->Message );

    // this error is most likely during connection if the COM port is incorrect
    Console::WriteLine( S"Are you sure the correct COM port was specified?"  );
  }

  __finally
  {
    // return timer resource
    if (EvtHandler::pingTimer) EvtHandler::pingTimer->Dispose();

    // Free library
    FlexiPanel::Quit( );
  }

  return 0;
}
```

About the author

Richard is a development engineer at FlexiPanel Ltd and co-author of *Data Mining and Business Intelligence: A Guide to Productivity*. He may be contacted at *rhoptroff@flexipanel.com*.