

## Designing User Interfaces for Windows Mobile Handhelds using Visual Studio and the .NET Compact Framework (Part 1)

[This is the first of a series of white papers on designing successful mobile solutions for various industries. This white paper is part 1 of a 2-part paper that focuses on designing and implementing user interfaces.]

Being in the business of designing mobile solutions for a variety of industries, designers and developers at InVision Software are faced with a wide array of challenges. One of the more exciting and interesting challenges is designing the handheld application user interface (UI) from sketchy details or from little more than a concept. Most product designers have a hard enough time designing a *desktop* UI from a concept-only specification. Try and jam all that required screen real estate into a 3 x 2 inch screen at 240 x 320 pixels resolution, and you find out quickly just how experienced you are at UI design!

Most customers have some idea what their application should look like and how it should interact with their users. Microsoft certainly has published general design guidelines, but first-hand experience is the best teacher. Rapid prototyping and stepwise refinement are a designer's best allies at this stage. That's why we choose tools that allow a concept to be rapidly visualized and prototyped.

We find it useful to have the client provide at least some data fields that **MUST** be on the handheld input side. Most business-oriented mobile applications have the same generalized model: Get data from a server for lookup, capture data either independently from the server (batch), or attached by a wireless connection, then send data back to a server. There are many technical approaches to ferrying data back and forth, and this will be a topic for the next white paper. The point is that *limiting* the amount of data that must be presented visually at the handheld is a good idea. It is a worthwhile effort to give serious consideration to partitioning and caching data between the server and the handheld.

### Interview the Client

Let's assume the client has provided a rough draft of the data entry forms that exist at the handheld. What follows is a series of questions we routinely ask the client.

#### 1. What input method will the user be employing most of time?

We like to classify usage into two very large and general categories:

- "Rough-and-ready" usage
- "Refined" contemplative usage

One quickly finds that the expectations of the user on a fork lift are *quite* different from that of, say, a market survey user. Many times we have watched looks of amazement from clients when reacting to the tiny keyboard "input panel" on a Windows Mobile device. "You don't really expect our users to peck away at this thing, do you?" is a common response to seeing it for the first time.

InVision supplies or recommends Windows Mobile handhelds in the form of Symbol Portable Data Terminals (PDTs), which fortunately offer a choice of form factors, including “real” keyboards. We find the most important step is to ascertain how users will navigate and input data. Many users prefer a “fat finger” approach. That is, no stylus and a real keyboard. Of course, that choice requires a heftier form factor that some users will not tolerate. In addition, they would like to see:

- fatter controls with a wide spread between them
- selection lists
- large fonts, bold text

This kind of user needs as much automatic assist with data entry as possible.

Some clients present a dilemma: “We need to capture a lot of data, but our users can’t be fussing with the handheld, and it must be quick.” Consider, for example, an emergency room nurse, or a patient care data capture system. How does a designer hit the mark between a responsive, easy-to-use system and the amount of data that may be required for capture? The form factor of a handheld screams for light-duty data capture and uncluttered forms for visual presentation but the client is requiring a UI for the desktop.

## **2. Can the system use barcodes, RFID, IR, Bluetooth, etc. for data capture?**

There is no greater assist for data capture than data that presents itself with a simple swipe of the handheld over or near a data source like a barcode. We developed a wireless sales order entry form system that salespeople use to generate orders from scanned bar-coded products at the stores. This client had experience with text-input handhelds and wanted more scan-able data input. One of the forms we designed required typing in a customer number and performed a partial-key lookup to retrieve customer data as they were typing. As efficient as this was, they were more than willing to tag *every* display shelf unit they owned at each store location with barcodes that had that customer number, so they could scan it, instead of typing.

We have just finished development of a mobile patient safety system for hospitals. One of the forms used for data input requires three sets of four vitals to be taken. Currently, they are recorded with a keyboard, but the user must touch the input textbox for each vital to set the focus for the keyboard. The form is quite crowded, but this is what the design dictated. In the future, we will ask medical device manufacturers if they offer a Bluetooth interface to automatically capture the vitals we need.

## **3. How can the data (both reference and capture) be spread across the interface?**

This question leads straight to the heart of a successful design. After laying the groundwork with the previous questions, we at least have some sort of indication how data will be captured and what the client is likely to use as an input device most of the time.

Presentation of data on forms is one of those talents acquired over years of experience with different industries. One never gets it right the first time. Clients

should be prepared to review prototypes and revise original concepts over more than a few cycles.

## Maximize screen real estate

Let's take a simple data capture requirement, and use a combination of Microsoft's Visual Studio 2003, .NET compact framework, tabs on forms, and an InVision control called "Magnify Label." Magnify Label is a control that addresses an issue in the world of handheld form data displays: How can the full content of a data field be displayed if the field is not large enough to display the data? At face value, this does not seem to be a problem. We just make the field larger. However, screen real estate is at a premium, so we can't just display fields at the maximum size of their content all the time.

So, for reference fields (fields that display reference information from lookups or queries but do not receive input focus) that are too short for the data they contain, there are several possibilities.

If we use a textbox and allow scrolling we incur several penalties:

- Scrolling is tedious with the stylus and with the keyboard, because the "window" (the size of the textbox) is small
- .NET compact framework does not allow the border to be removed
- The Read-only property grays the background by default and you can't change this. On monochrome displays, this makes for black on gray text...a bad combination that is almost unreadable.


If we use a label:

- There is no scrolling property
- There is no border property
- There is no BackColor property

There is no ideal, and we have been faced with conflicts between the client's requirements and the .NET controls available. The solution was simple, but a clear need emerged for a control to "magnify" fields at the user's whim. We decided to inherit the base control, Control, and give it some properties that respond to a tap and magnify its contents. When tapped again, it returns to its original size. You can instantiate the control with or without a border, a magnification factor, and position and size parameters. The control is smart enough to magnify itself to fill the entire screen horizontally and will size vertically by the multiplication factor parameter. When magnified, if it grows too far vertically, it re-positions itself vertically within its container so no bottom content is clipped. Word wrap is supported in magnify mode, as well, but NOT in normal mode as too much content may be lost when spaces are used.

The idea of this simple but helpful control is to present just enough information to the user to give him an idea of what the content may be, and provide a means of fully rendering the content when he doesn't have a clue, or just wants to confirm what he is seeing. A more sophisticated version might display a small tree to display and navigate hierarchical information.

### Example: Form fields without magnification



Inventory 1:37

Location ID/Description: 123  
 Vinter's Warehouse at 135 Renison D

Item ID: 1

Vintage: 1997

Name: Beaulieu Clone #4 Cabernet Sa

Producer: Domaine De La Romanee  
 Conti

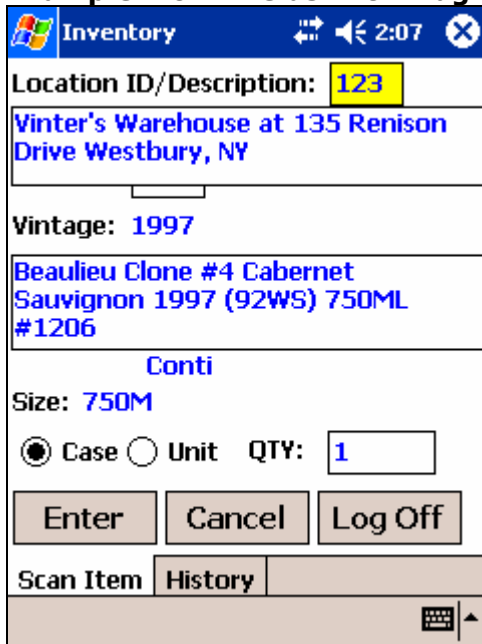
Size: 750M

Case  Unit QTY: 1

Enter Cancel Log Off

Scan Item History

### Example: Form fields with magnification



Inventory 2:07

Location ID/Description: 123  
 Vinter's Warehouse at 135 Renison  
 Drive Westbury, NY

Vintage: 1997

Beaulieu Clone #4 Cabernet  
 Sauvignon 1997 (92WS) 750ML  
 #1206

Conti

Size: 750M

Case  Unit QTY: 1

Enter Cancel Log Off

Scan Item History

The same control can be floated over a data grid so that when tapped it gives the appearance that the data grid's cell has been magnified.

## Next Installment

That's it for now. In the next installment we will cover handheld form design using tabs and other handheld UI issues such as

- Status displays: Does the user need to know the status or history of his data capture entries? Does the user need to refer to a to-do list for entries to be performed?
- Scan Focus: How to give feedback to the user when multiple scan fields are required.