

Implementing Specialized Data Capture Applications with InVision Development Tools (Part 1)

[This is the first of a series of white papers on implementing applications with special requirements for data capture such as barcodes, signatures, and magnetic stripes. This white paper is focuses on implementing scan-enabled applications.]

“There Are No Problems, Just Opportunities”

So, your boss just called you and told you that your company’s manufacturing business has increased 10-fold due to successful advertising. He needs an application to scan items as they are received at the warehouse receiving dock and update the company’s asset database...pronto. Worse, there are hundreds of items on the receiving platform and the warehouse guys are threatening to strike because of the workload. Your one ally is the receiving clerk, who has been logging in items by hand until now.

Being a seasoned IT professional, how do you handle the situation? Well, you could jump in and volunteer to pull all-nighters with the receiving clerk for the rest of your career, or you could finally modify the inventory application for that scanning device you bought a year ago.

Scan-Enabling a Pocket PC Application

Let’s assume your company has a simple MS-Access database. Your challenge (opportunity) is to get the item data from the scanning device and update that database. We’ll start with an embedded VB application and InVision’s **iScanIt™** ActiveX control. iScanIt is an ActiveX Control specifically designed to control bar code scanning on data collection devices. iScanIt is fully scalable, and is available for use on most Windows-based platforms. This custom control provides bar code scanning capabilities for inventory control, warehouse management, point of sale, identification and any other data collection needs.

Assuming you have purchased a copy of iScanIt and installed and registered the control on your development desktop, let’s fire up eVB. We’re going to use a Symbol 2800 device with the Pocket PC O/S for this project. We’ll make this very simple and create a form with three fields:

- Item Number – this will receive the Item barcode, or allow the user to enter the barcode manually
- Location – this will be a simple pick list of locations downloaded from the database
- Quantity – this will be a text field for manual entry, but will additionally increment every time the user scans an item with the same barcode as the last item scanned.

Creating the Scan Form

First, we must make eVB aware of our controls by populating its toolbox with some of the controls from the InVision iScanIt install. We’re going to use two controls:

- iScanIt, the InVision Scan control itself

- ScanText, a simple “scan-aware” component. Scan-aware controls are similar in some respects to their standard eVB counterparts and contain additional properties and events that further enhance data capture such as filtering the scanned data and telling us whether a match was found.

In Figure 1, we have already added the ScanText and iScanIt controls to the toolbox. We got to this point by right-clicking on the toolbox, choosing **Components...** to display the eVB Components dialog box and then selecting the **InVision ScanText Type Library** and **InVision ActiveX Scan Control** controls. Notice the small filter and barcode icons in the toolbox.

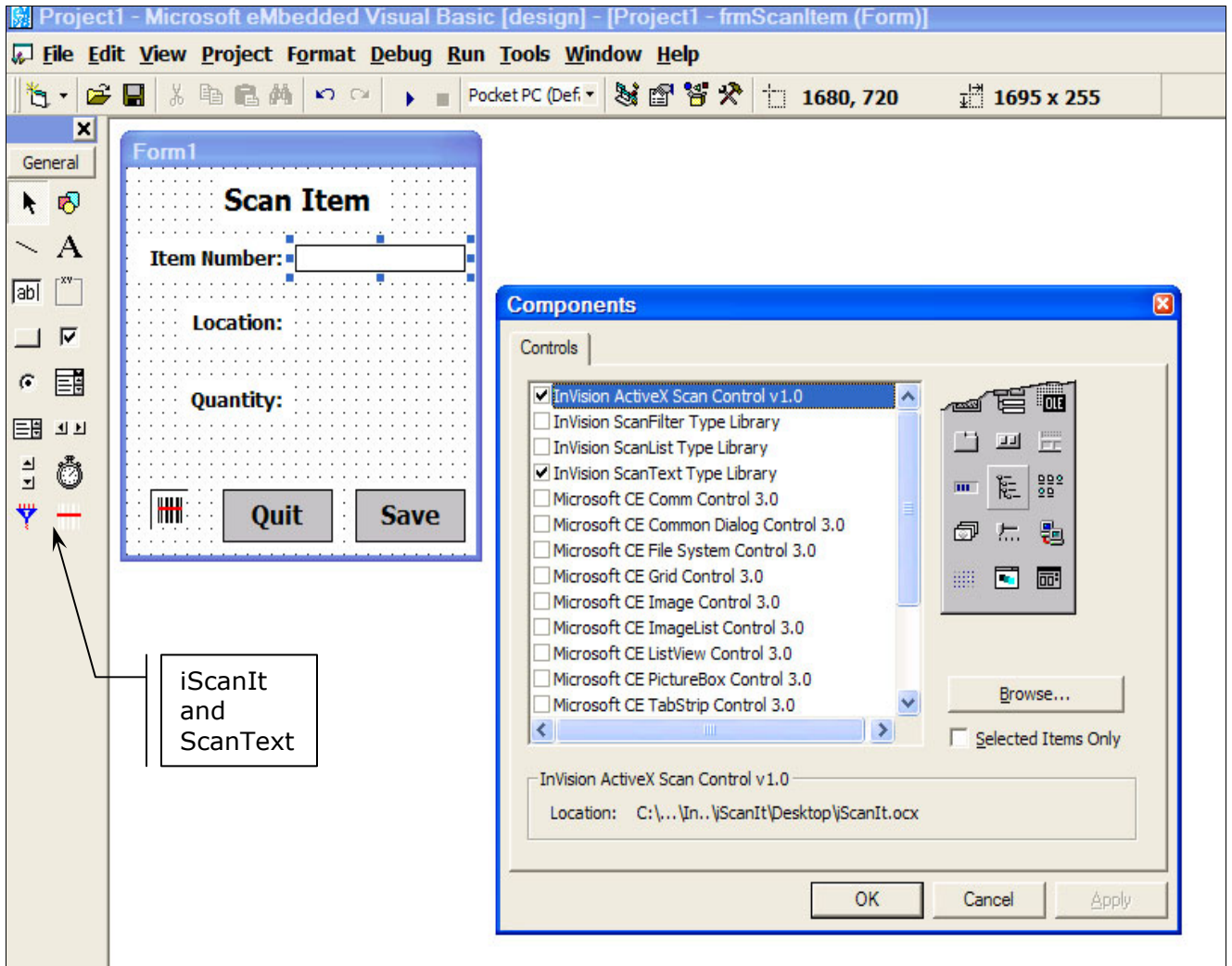


Figure 1: Adding the iScanIt and ScanText Controls

Now we'll click on each icon in the toolbox and then click on the Scan Item form to place the controls on the form. Don't worry about the iScanIt control; although it looks like it's visible on the form, it is intrinsically an "invisible" control. We will just drag it out the way of any other control so we can click on it and view its design time properties.

Additionally, we will drag the ScanText control alongside the "Item Number" label.

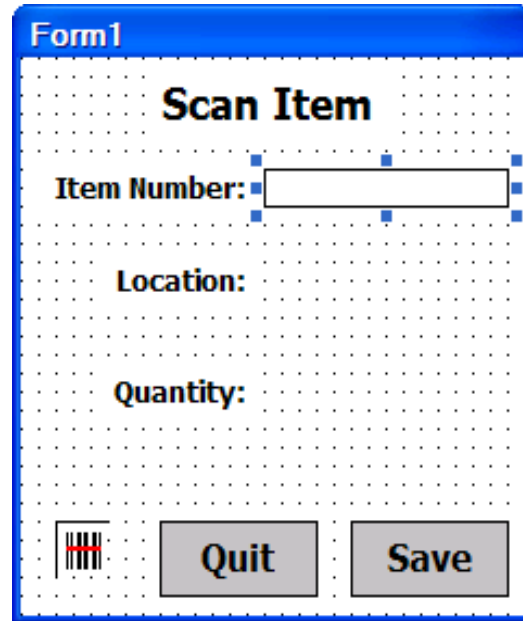


Figure 2: Scan Item with ScanText

We have to take note of some properties of the iScanIt control and change to suit:

- iScanIt:
 - ScannerEnabled, Scanner Open - If both ScannerOpen and ScannerEnabled are set to True at design time, the scanner device will be opened and enabled automatically when the form is loaded. This is the simplest set of options.
- ScanText:
 - Text: "ScanText" - we'll change that to blank
 - BorderStyle - we'll change to True
 - ScanControl - This tells the ScanText control which control we are using to handle scanning. In this case it is the iScanIt control. This is not a design-time property, so we'll have to set this in the form's load event:

```
Private Sub Form_Load()
    ScanTextCtl1.ScanControl = iScanIt1
End Sub
```

Before we look at some of the more interesting scan-aware properties, let's get some immediate feedback: In other words, see if the scanner works! Before we can do anything, we have to load the ScanText control onto the handheld. The eVB IDE can help us with this.

Ensure the handheld has an ActiveSync connection with the desktop, and Select Tools...Remote Tools...Control Manager... from the main menu. This displays the Control Manager Window in Figure 3.

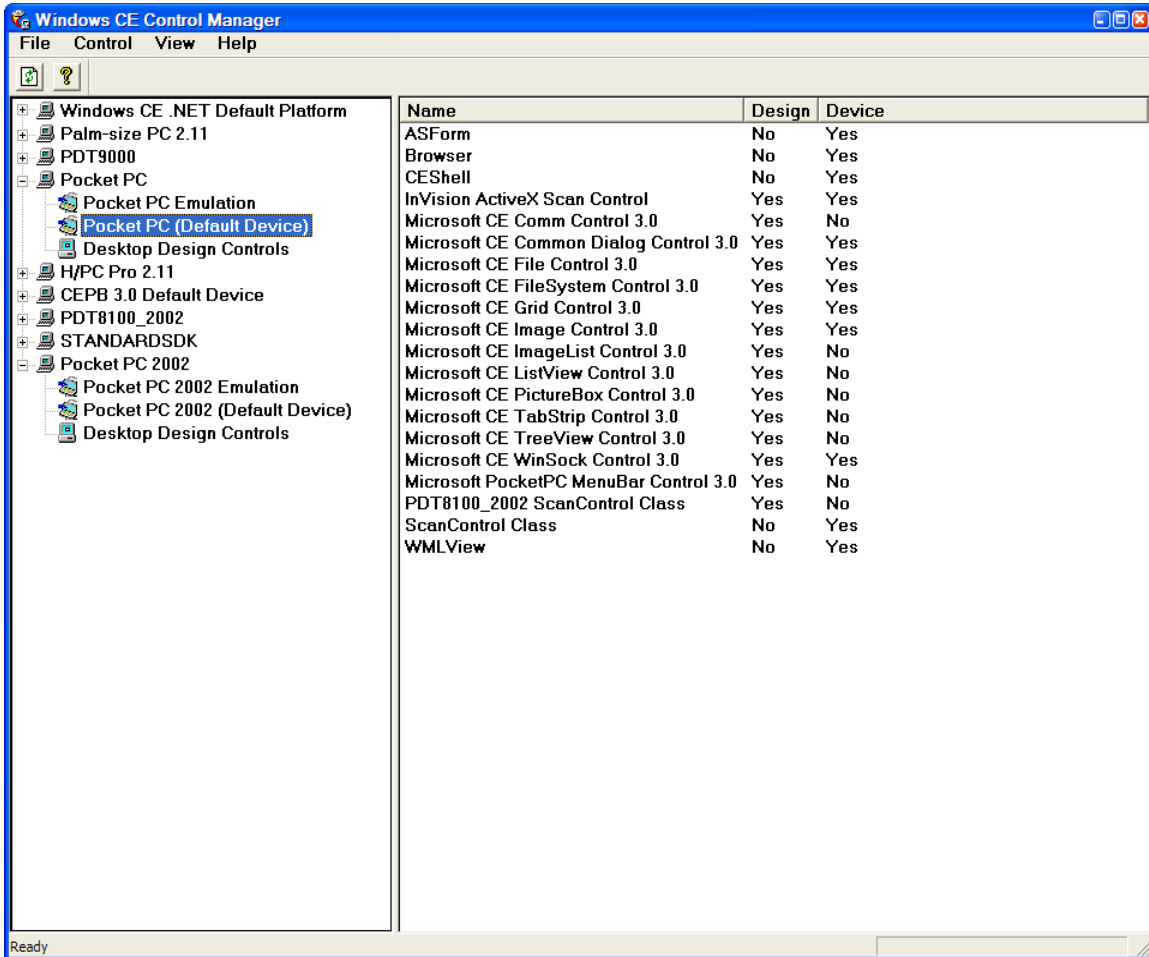


Figure 3: Control Manager

We'll select the O/S we're using in the left pane: Pocket PC. This displays a list in the right pane that tells us what controls are installed and accessible at design time vs. which ones are installed on the handheld device. Notice that the InVision controls are nowhere to be found on this list...we have to add them.

Right-click anywhere in the right pane and select "Add New Control...". Navigate to C:\Program Files\InVision Software\iScanIt\Device and select the folder that matches the processor of your handheld device. For this sample, we are developing for an ARMPPC, or Armstrong Pocket PC-based handheld, so we will navigate to C:\Program Files\InVision Software\iScanIt\Device\ArmPPC. Select the ScanText.ocx and iScanIt.ocx files and click Open. This automatically installs and registers the controls on the handheld.

We are now ready to tell eVB to run our application! Choose Run...Execute. eVB should load download and run our application, and the form should display. Press the scan trigger on the handheld and scan a barcode with one of the symbologies that are enabled by default, say Code 39, Full ASCII. You should see the content of the barcode appear in the ScanText control.

Voila! We are just a few steps away from a really useful application and your ascent to White Knighthood of the Receiving Clerks!

Database Access

It's a good thing Microsoft chose to support ActiveSync synchronization between MS-Access and the CEDB format for its Pocket PC architecture, because it's one of the compelling reasons to adopt it for small, simple environments like our sample.

Using MS-Access, create a database called Inventory.mdb and create two tables:

- "tblInventoryTransactions"
 - TransactionNumber (number, primary key)
 - ItemUPC field (text, size 50)
 - Location field (text, size 50)
 - Quantity field (number)
- "tblLocations"
 - LocationID field (number, primary key)
 - LocationDescription field (text, size 50)

Open the tblLocations table and type in some locations and assign each a unique LocationID number. Real simple; no database gurus need apply. Obviously, the transactions we receive need to be applied to a real Master Inventory database, but that is beyond the scope of this article.

Synchronize the Database with the Handheld

Using ActiveSync with the device connected, click the Options Icon and check Pocket Access in the list on the Sync Options tab (Figure 4).

Now click the Settings button and then click Add.

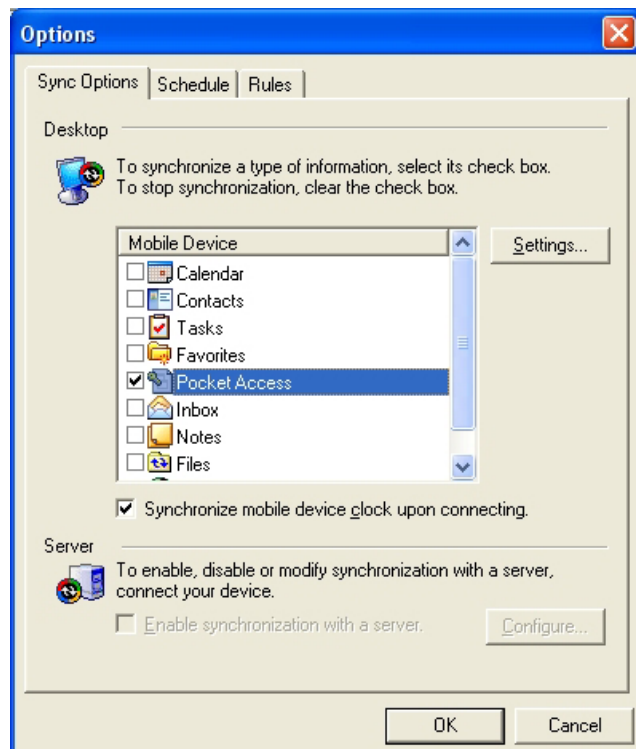


Figure 4: Sync Options

Select the Inventory.mdb database, and in the next dialog box (Figure 5), be sure to enter the complete path of where the Pocket PC version (Inventory.CDB) of the Inventory.mdb should reside. For simplicity, enter the path of the .vb executable you created. You can find this in eVB Project...Properties, General tab under the Remote Settings, Remote Path designations. For our project it is "\\Program Files\WhitePaper\Project1.vb", so "\\Program Files\WhitePaper\" is where you want the Inventory.CDB database to reside.

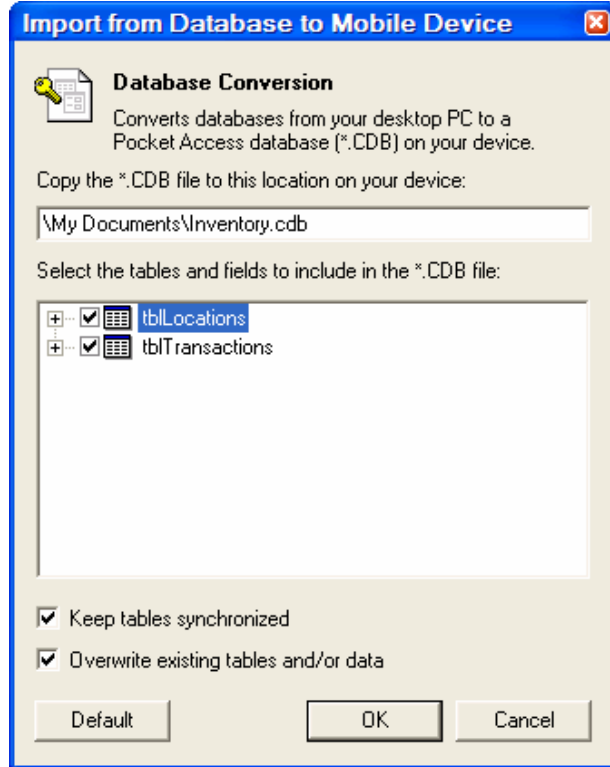


Figure 5: ActiveSync Import Database Dialog

When complete, you should end up with a Database Synchronization dialog that looks like Figure 6.

The database should synchronize immediately.

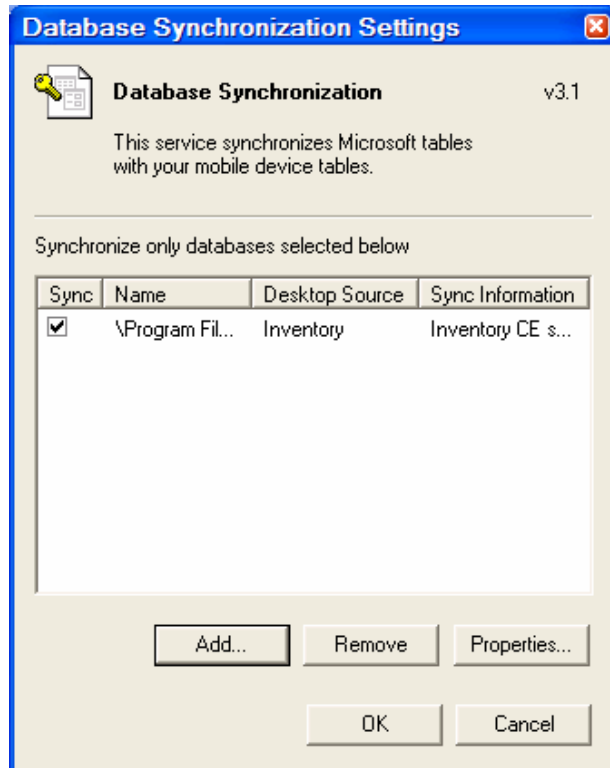


Figure 6: Database Synchronization Settings

Finish the Application – Add Database Code

Now we'll go back to eVB and add a couple of fields we will need on the Inventory Form:

- A dropdown list named "drpLocations" to display the locations for selection
- A textbox named "txtQuantity" to enter the quantity

The form should now look like Figure 7.

Let's add the reference that ScanText requires so that it can receive scans from iScanIt. We also need to populate our Locations dropdown list with the choices from the database. We'll do that in the main form's load event:

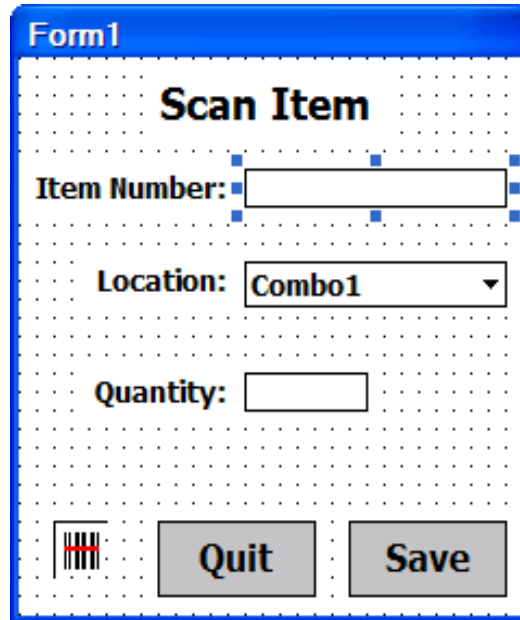


Figure 7: Completed Form

```
Private Sub Form_Load()
    scnItemNum.ScanControl = iScanIt1 ←-----Reference to
iScanIt
    scnItemNum.Text = ""
    strActiveConnection = "\Program
Files\WhitePaper\Inventory.cdb"
    'Connect to database
    Set connData = CreateObject("ADOCE.Connection.3.0")
    connData.ConnectionString = "provider=cedb;data
source=" & strActiveConnection
    connData.Open

    Dim rsData As ADOCE.Recordset

    On Error Resume Next
    'retrieve records from database
    Set rsData = CreateObject("ADOCE.Recordset.3.0")
    rsData.Open "SELECT * FROM tblLocations ",
strActiveConnection
    If Not rsData.BOF And Not rsData.EOF Then
        'Add to list
        rsData.MoveFirst

        Do While Not rsData.EOF
            drpLocation.AddItem
rsData.Fields("LocationDescription")
            drpLocation.ItemData(drpLocation.NewIndex) =
rsData.Fields("LocationIndex")
            rsData.MoveNext

        Loop
    End If
End Sub
```

```
End If

Set rsData = Nothing
rsData.Close

End Sub
```

Also, we must add the code that inserts the scanned item's barcode, it's locationID, and the quantity found. We'll do that in the Save button's click event:

```
Private Sub cmdSave_Click()
Dim lKey As Long
Dim currentNow As Variant ' will hold a date

currentNow = Now ' fixes the date and time at a point in
time
lKey = CLng(CStr(CInt(Month(currentNow)) & _
CStr(CInt(Day(currentNow)) & _
CStr(CInt(Hour(currentNow)) & _
CStr(CInt(Minute(currentNow)) & _
CStr(CInt(Second(currentNow))))))
connData.Execute ("INSERT INTO tblTransactions
(TransactionNumber, ItemUPC, LocationID, Quantity) VALUES("
& lKey & ",'" & scnItemNum.Text & "'," &
drpLocation.ItemData(drpLocation.ListIndex) & "," &
txtQuantity.Text & ") ")

End Sub
```

Some Nice Additions

We will need some global definitions to handle database, etc.:

```
Dim connData As ADOCE.Connection
Dim strActiveConnection As String
Dim strOldItemNum As String
```

A nice touch to add is the ability to recognize the last barcode scanned and automatically increase the quantity by 1. We'll have to handle the "Scan_Complete" event from the iScanIt control to do that. Just double-click on the iScanIt control to generate the skeleton code for that default event, and fill it like so:

```
Private Sub iScanIt1_ScanComplete(ByVal lStatus As Long,
ByVal strData As String, ByVal lLength As Long, ByVal
strAIMCode As String, ByVal lSymbology As Long)
If (strData = strOldItemNum And
IsNumeric(txtQuantity.Text)) Then
txtQuantity.Text = CStr(CInt(txtQuantity.Text) + 1)
Else
strOldItemNum = strData
txtQuantity.Text = "1"
End If
End Sub
```

And finally, we close the database connection and disable and close the scanner. If this is not done, the application will never properly close, and the device may have to be warm-booted:

```
Private Sub cmdQuit_Click()  
    connData.Close  
    iScanIt1.ScannerOpen = False  
    iScanIt1.ScannerEnabled = False  
  
    App.End  
End Sub
```

Now GO PLAY!

Fire up the application either with the RUN...EXECUTE technique from eVB or just click on the .vb executable. Scan a barcode, select a location. When you scan a new item, the quantity should start at 1 and increment for every barcode that matches the last one. When you are done with an Item(s), click the SAVE button. Go on to the next item and do the same thing.

When done, use ActiveSync to synchronize the device. Then open the Inventory database on the desktop and open the tblTransactions. You should see the transaction records reflecting the items scanned, their locations, and the quantities found.

Next Installment

In the next part of this series, we will use iScanIt's symbology selection functions and filtering to deal with those pesky barcode check digits! In addition, we will introduce a very unique feature: Web-enabled scanning.