

## Implementing Specialized Data Capture Applications with InVision Development Tools (Part 2)

[This is the second 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 refining the application developed in Part 1 and introduces scan-enabling web pages.]

In Part 1 of this series, we developed a basic bar-coded data capture application for a handheld device that captures data to an Ms-Access inventory database for items received by a receiving clerk. In this paper, we will explore how to control barcode symbologies and filter the data received as a result of scanning a barcode.

### Barcodes

A bar code symbology is the encoding scheme used to express numerical and character data with a pattern of bars and spaces in a bar code. These bars and spaces are the elements of the symbology. This paper is only concerned with "**Linear Symbology**," which is one row of bars and spaces. The elements vary either by width as in the bar code on the left or by height as in the bar code on the right:



Some symbologies can represent and contain the full range of ASCII characters, such as Code 39 — Full ASCII, or numeric only characters such as The UPC/EAN/JAN Family. UPC stands for the Universal Product Code. This symbology was initially adopted for use by the U.S. grocery industry, although its use has spread into other retail market places as well. The Uniform Code Council (UCC) controls the allocation of UPC codes.

Examples of UPC barcodes follow:



UPC-A

UPC-E

The last digit of a UPC/EAN/JAN bar code message is always a "check digit." It is not part of the identification number, but an extra number used to guarantee the data integrity of the bar code. Its value is automatically computed from the other digits in the code and then it is printed as the last digit.

The first digit of a UPC-A code is called the "number system digit". Like the check digit, it is often stripped off by your bar code reader, and therefore does not enter

your computer or cash register. The first generation of UPC-A labeled products used a "0" almost exclusively as the number system digit. However, as more and more products use UPC bar codes, number system digits other than "0" are becoming more prevalent.

Upon reading the code, the bar code reader repeats the same check digit computation that was performed when the bar code was printed. If the newly computed check digit value does not agree with the one previously encoded into the bar code, a read error is detected, and the bar code reader does not transmit any part of the message. In the case of a good read, this digit may or may not be "stripped off" prior to transmission of the bar code to the host system. This depends on the configuration of your bar code reader.

The symbology structure of UPC-A is 12 digits (10 without the "number system" and "check" digits).

I generated the following barcode with a commercially available utility, designated a UPC-A symbology, and assigned a data value of "1234567890" (recall, numbers only):



**Figure 1 UPC-A barcode**

Before we go any further, your assignment will be to print this page, and use the application we developed in Part 1 of this series to scan the barcode above. Note that we have done nothing to configure the scanner on the handheld device, so all the scanner parameters should be set to their defaults.

Note the Item Number field after the barcode is scanned. It contains:

012345678905

This is more than we want if the Item Number is only 1234567890, so what happened? We need to configure the scanner to strip off the last check digit and disregard the number system preamble. Let's add some code to do just that. We'll add 3 lines to our Form Load event:

```
iScanIt1.ScannerEnabled = False  
iScanIt1.SetUPCAPParams False, 0  
iScanIt1.ScannerEnabled = True
```

Here we used the iScanIt control's Method SetUPCAPParams method to set the UPCA symbology parameters to strip off the check digit (False) and return no preamble (0).

*NOTE: If scanning is enabled, SetUPCAPParams method calls have no effect until scanning is first disabled, then re-enabled!!!*

If we make the changes and run the handheld application again, we get in the Item Number field:

1234567890

which is the UPC data for our Item.

## **Scan-Enabled Web Applications**

One of the most exciting features of **iScanIt™** is its ability to take advantage of allowing itself to be used in a web page. Using the standard OBJECT tag, we can set up all the parameters and actually get the control to render on a web page. The user then navigates with Pocket Explorer to the web page as usual and can scan a barcode directly into a field on a form.

In order to see how this might work, we'll take our scan-enabled inventory application and host it on a web page. Going one exciting step further, we will use Visual Studio 2003 and .NET to create the web application and host the form and its data entry fields.

### **Some Important Differences in this Version**

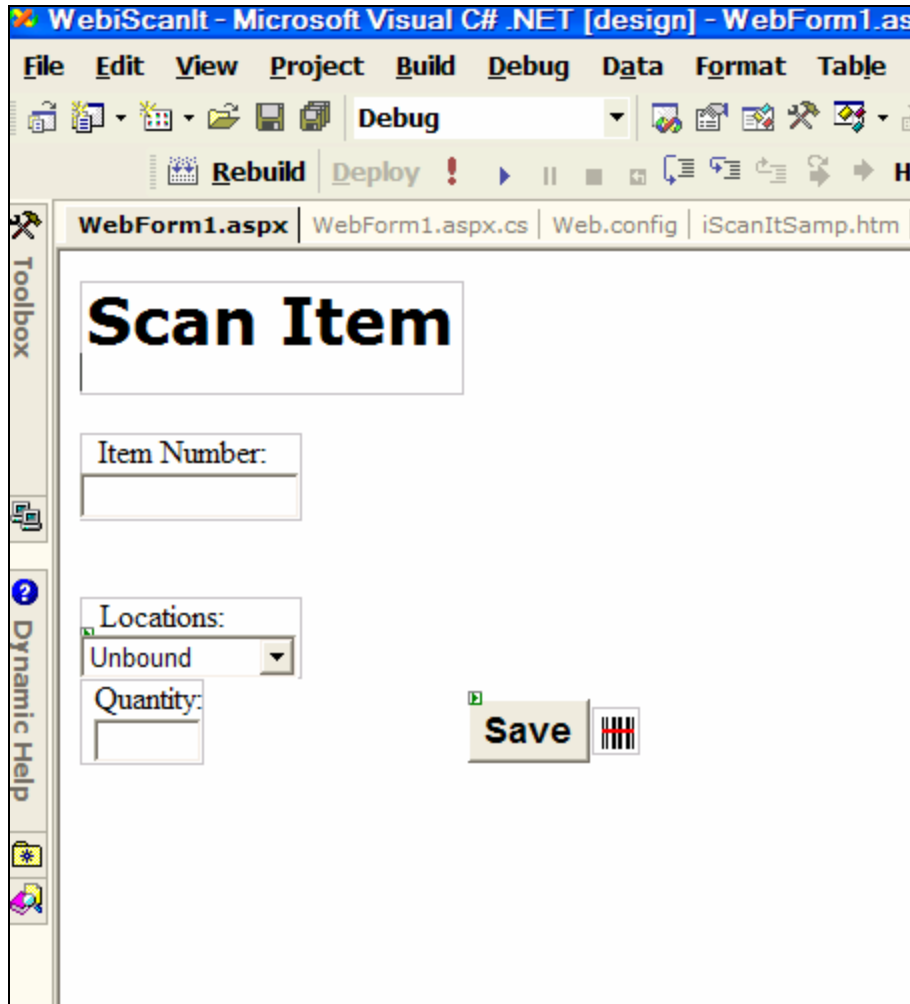
We have been taking advantage of a "disconnected," batch-style approach to our data capture solution. The user can be independent and disconnected from the "main" database while performing item scanning. This is because the reference data needed for lookup (the Locations list) is "downloaded" via ActiveSync synchronization with the desktop copy of the MS-Access database. Also, any data-entry transactions captured are "batched" in the local copy of the database in the handheld, and similarly "uploaded" during the same synchronization.

For the web application version to operate correctly, the handheld device must be able to connect to the web server at any time. So for this version, we need a wireless-capable handheld, say a Symbol 2837. Now, we no longer need to support a local copy of the database or a mechanism to synchronize local and desktop copies of the databases.

The transactions that represent each scanned item, its count, and its location update the desktop database in real-time as the user clicks the Save button on the handheld. If we applied the transactions to a master inventory table as they were inserted into the transactions table, we would have the basis for an on-line inventory system.

## Implementing the Inventory Scan-Enabled Web Application

We assume the reader has implemented at least one .NET ASP.NET web application with Visual Studio .NET. We are using a "Web Form" as our container for the iScanIt control. Here's the form as displayed in the Visual Studio IDE:



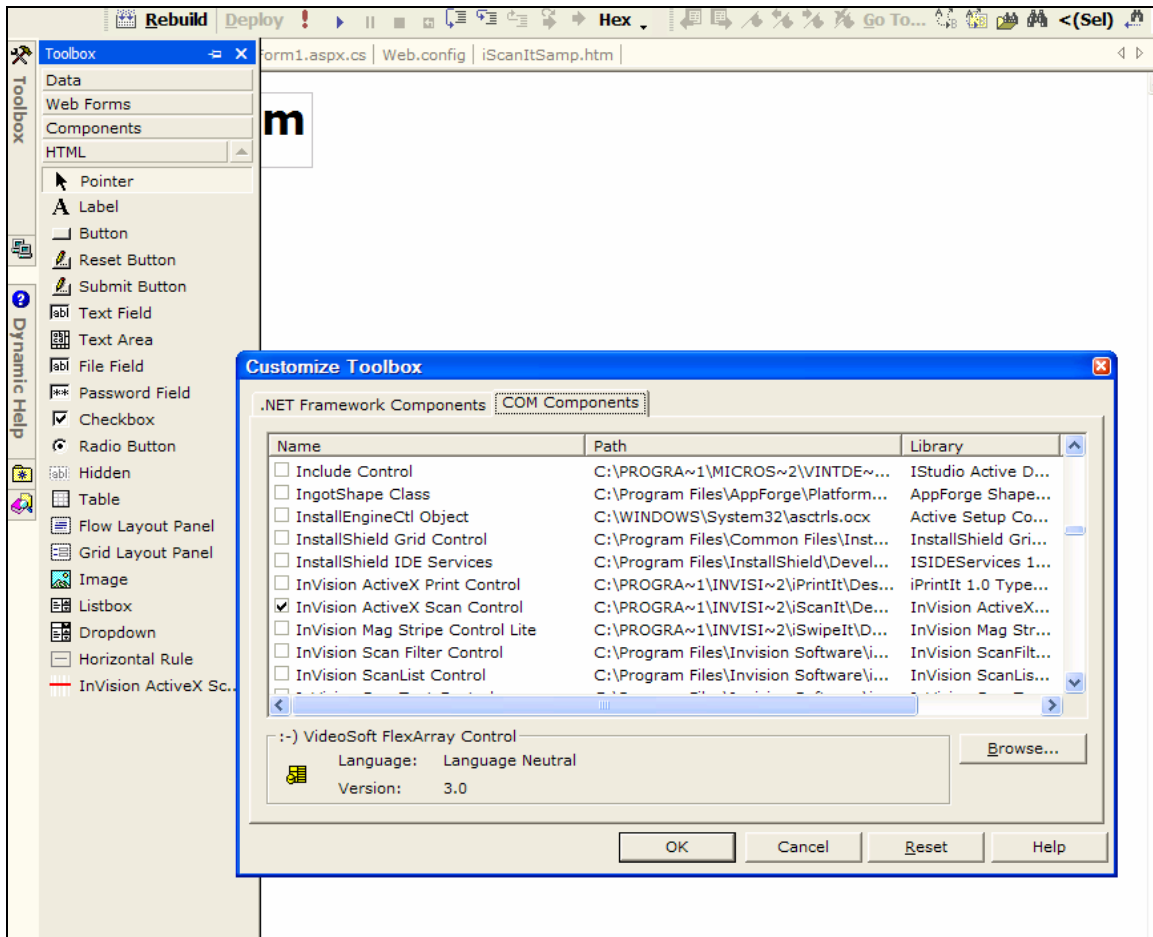
**Figure 2 Visual Studio .NET Inventory WebForm**

The following controls exist within our form tag:

- The Locations dropdown and the Save button are Server Controls. They execute on the server for database access reasons.
- The Item Number and Quantity controls are HTML controls and must execute on the client. The Quantity control will use some DHTML to respond to scanning events and update its quantity. The Item Number control will be updated with the scan data for each scan event as well.

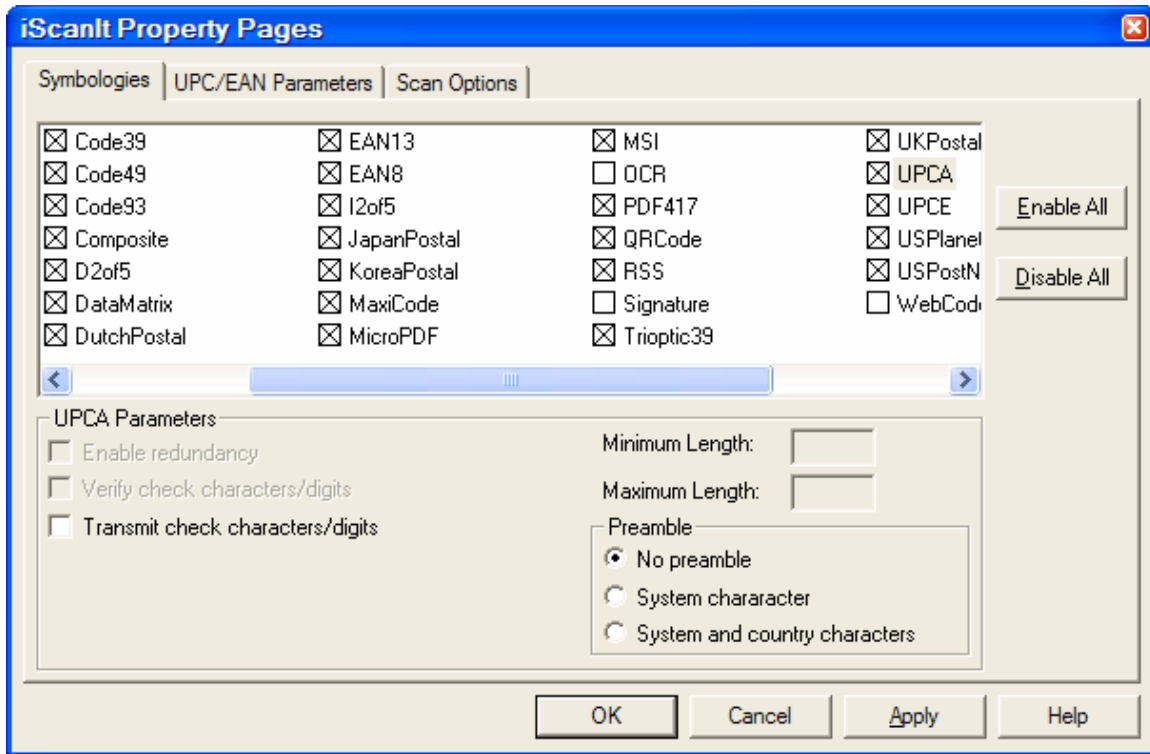
We assume that iScanIt component has been installed or downloaded on the handheld device.

How did we get the iScanIt Control on the web form? Simple...we just selected the HTML control group on the toolbox and added "InVision ActiveX Scan Control" from the Com Components tag:



**Figure 3 iScanIt in the VS Toolbox**

We then drag the control anywhere on the WebForm. Visual Studio, being a much more sophisticated environment than eVB, allows us to look at the iScanIt property pages if we select it and right-click and then select properties:



**Figure 4 iScanIt Property Pages**

Click on the UPCA checkbox and note the options selected. Remember the UPCA parameters we set in code in our eVB version? Let's set the same ones here (as above) in the VS IDE:

- Deselect "Transmit check characters/digits"
- Select the "No preamble" option

Click OK, and take a look at the HTML code generated by these selections, you will notice there is a fairly large HTML OBJECT tag with all the parameters we set encoded. We could also set these parameters at run time, when the page loads.

One thing left to do, make sure the "name" property of the control (set in the property window of the VS IDE) is set to "iScanIt", so we can refer to the control in the code.

## Database Access

Set up an ODBC DSN pointing to the MS-Access database called "Inventory", and then use the VS IDE Designer and drag an ODBC command object from the Data group. Point the corresponding ODBC connection object to the ODBC DSN, and set the CommandText property to "SELECT LocationIndex, LocationDescription FROM tblLocations".

Set the drop down list controls properties like so:

- DataTextField = LocationDescription
- DataValueField = LocationIndex

## Code-Behind Page Processing

When the page loads, in the code-behind we simply populate the dropdown list with the locations from the table and bind it to an ODBC Reader object. .NET generates the page to the browser with the list filled with the locations in our table.

```
private void Page_Load(object sender, System.EventArgs e)
{
    if (!IsPostBack)
    {
        // Fill the drop down list with our locations

        odbcConnection1.Open();
        OdbcDataReader myReader =
odbccmdLocations.ExecuteReader();
        drpLocations.DataSource = myReader;

        drpLocations.DataBind();

        myReader.Close();

        odbcConnection1.Close();
    }
}
```

When the user hits the Save button, the form data (item Number, the location selected and quantity) is posted back to the server. We then refer to the control data in the form and update the database:

```
private void Button1_Click(object sender, System.EventArgs e)
{
    // Update our Inventory Database
    odbcCommand1.CommandText = "INSERT INTO tblTransactions
(TransactionNumber, ItemUPC, LocationID, Quantity) VALUES(" +
    DateTime.Now.Ticks.ToString().Remove(0,12) + "','" +
    Request.Form["scnItemNumber"].ToString() + "','" +
    Convert.ToInt32(drpLocations.SelectedValue.ToString())
+ "','" +
    Request.Form["txtQuantity"].ToString() + ")";

    try
    {
        odbcConnection1.Open();
    }
}
```

```
        odbcCommand1.ExecuteNonQuery();
        odbcConnection1.Close();
    }
    catch (Exception ex)
    {
        Response.Write ("Exception: " + ex.Message);
    }
}
```

## DHTML Page Processing

We need to write a few lines of JavaScript to initialize the iScanIt control, to hook up its Scan\_Complete event to a function, and to handle our quantity field auto-increment. Here's the HTML and scripting for the entire page:

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireup="false" Inherits="WebiScanIt.WebForm1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <title>WebForm1</title>
    <meta content="False" name="vs_showGrid">
    <meta content="Microsoft Visual Studio .NET 7.1"
name="GENERATOR">
    <meta content="C#" name="CODE_LANGUAGE">
    <meta content="JavaScript" name="vs_defaultClientScript">
    <meta content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema">
    <SCRIPT language="javascript" id="clientEventHandlersJS">
  <!--

  var strOldItemNum;

  function iScanIt_ScanComplete()
  {

    scnItemNumber.value = iScanIt.ScanData

    if ((iScanIt.ScanData == strOldItemNum) && (txtQuantity.value
!= ""))
    {
      var Qty = parseInt(txtQuantity.value) + 1
      txtQuantity.value = Qty.toString();
    }
    else
    {
      txtQuantity.value = "1";
      strOldItemNum = iScanIt.ScanData;
    }
  }

  function Cleanup()
  {
```



```

name="iScanIt" VIEWASTEXT>
  <PARAM NAME="DeviceNumber" VALUE="1">
  <PARAM NAME="ScannerOpen" VALUE="-1">
  <PARAM NAME="ScannerEnabled" VALUE="0">
  <PARAM NAME="Timeout" VALUE="5000">
  <PARAM NAME="ScanWedgeMode" VALUE="0">
  <PARAM NAME="WedgeSuffix" VALUE="0">
  <PARAM NAME="ScanOptions"
VALUE="24010000000000008813000001000000D007000001000000C80000006E0A">
  <PARAM NAME="UpcEan"
VALUE="1400000001000000000001010001010201000105">
  <PARAM NAME="AusPostal" VALUE="0C000000020000000001">
  <PARAM NAME="Aztec"
VALUE="140000000300000001000000A60E0000000101">
  <PARAM NAME="CanPostal" VALUE="0C000000040000000001">
  <PARAM NAME="CodaBar"
VALUE="18000000050000000200000003C00000000010000000001">
  <PARAM NAME="CodaBlock"
VALUE="140000000600000000000000000800000001">
  <PARAM NAME="Code11"
VALUE="180000000700000004000000500000000001020101">
  <PARAM NAME="Code128"
VALUE="1800000008000000000000005000000000010101">
  <PARAM NAME="Code39"
VALUE="1C0000000900000000000000300000000001">
  <PARAM NAME="Code49"
VALUE="140000000A000000000000003C0000000001">
  <PARAM NAME="Code93"
VALUE="140000000B00000000000000500000000001">
  <PARAM NAME="Composite"
VALUE="140000000C00000001000000BE0A00000001">
  <PARAM NAME="D2of5"
VALUE="140000000D000000040000005000000000010001">
  <PARAM NAME="DataMatrix"
VALUE="140000000E00000001000000DC050000000101">
  <PARAM NAME="DutchPostal" VALUE="0C0000000F0000000001">
  <PARAM NAME="EAN13" VALUE="0C00000010000000000101">
  <PARAM NAME="EAN8" VALUE="0C00000011000000000101">
  <PARAM NAME="I2of5"
VALUE="180000001200000004000000500000000001">
  <PARAM NAME="JapanPostal" VALUE="0C000000130000000001">
  <PARAM NAME="KoreaPostal" VALUE="0C000000140000000001">
  <PARAM NAME="MaxiCode"
VALUE="140000001500000001000000960000000001">
  <PARAM NAME="MicroPDF"
VALUE="140000001600000001000000BE0A00000001">
  <PARAM NAME="MSI"
VALUE="18000000170000000400000030000000000100000001">
  <PARAM NAME="OCR" VALUE="1003000018000000000003">
  <PARAM NAME="PDF417"
VALUE="140000001900000001000000BE0A00000001">
  <PARAM NAME="QRCode"
VALUE="140000001A00000001000000AC0D00000001">
  <PARAM NAME="RSS"
VALUE="140000001B00000001000000500000000001">

```

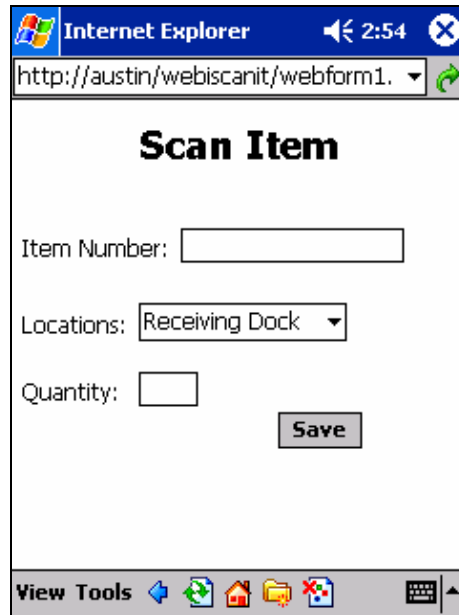
```

        <PARAM NAME="Trioptic"
VALUE="140000001D000000000000000000000001">
        <PARAM NAME="Signature" VALUE="100000001C000000">
        <PARAM NAME="UKPostal" VALUE="0C0000001E0000000001">
        <PARAM NAME="UPCA" VALUE="0C0000001F0000000001">
        <PARAM NAME="UPCE" VALUE="100000002000000000000101">
        <PARAM NAME="USPlanet" VALUE="0C00000021000000000101">
        <PARAM NAME="USPostnet" VALUE="0C00000022000000000101">
        <PARAM NAME="WebCode" VALUE="1400000024000000">
    </OBJECT>
</form>
</SCRIPT>
        iScanIt.ScannerEnabled = true;
</SCRIPT>
        <SCRIPT language="javascript" event="ScanComplete"
for="iScanIt">
            iScanIt_ScanComplete();
        </SCRIPT>
</body>
</HTML>

```

**Try it!**

After permissioning the web application and MDB file permissions for your particular environment, visit the web page. You should see the following:



**Figure 5 Webform Page in Pocket IE**

Scan a UPC-A barcode and watch it increment. Press the Save button and then inspect your database on the desktop.

**Next Installment**

That's it for now. In the next installment, we examine InVision's **iSignIt™** control, a control for capturing signatures!