Enhancing Expo Displayfile Guide

Overview

This guide will show basic techniques to change the look and feel of Monarch Migrated application Pages, to make them more appealing.

During the description of the steps to apply the different techniques, the public repository in GitHub will be used:

Reference public GitHub repository:

`https://github.com/ASNA/SunFarm`

The repository was created with a fresh new Monarch Migration. As enhancements were being applied, the required changes were committed, such that by using git History tooling, the changes may be seen in real code.

Assumptions

Familiarity with the ASNA ***Customer Sample IBMi program*** is required. This program has been provided by ASNA as sample RPG/CL code in several products, with the names: M4CUST, M5CUST and/or IronMonger.

Scope

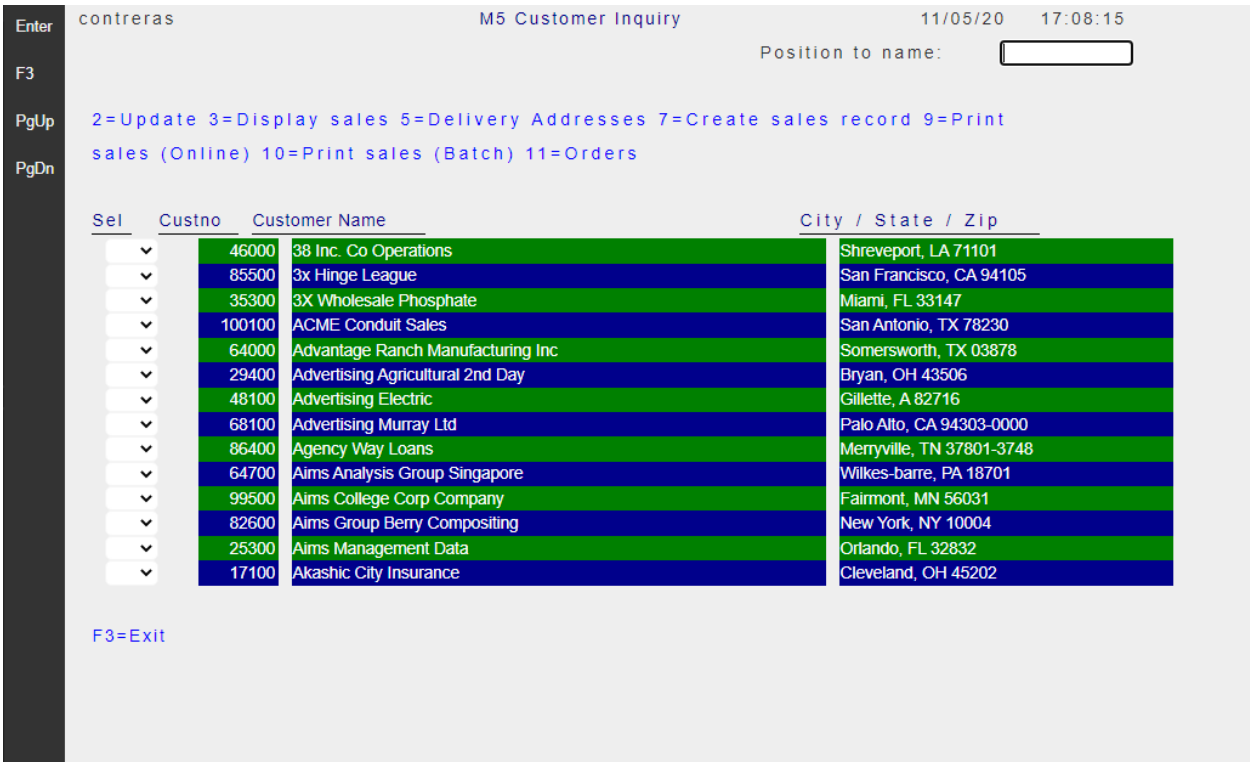
This Guide shows how to Enhance the following Pages:
1. Customer Inquiry (Initial Page)
2. Customer Maintenance (Option “2”)

In addition, the Site has been branded to show the fictitious company “SunFarm” with its logo on all pages.

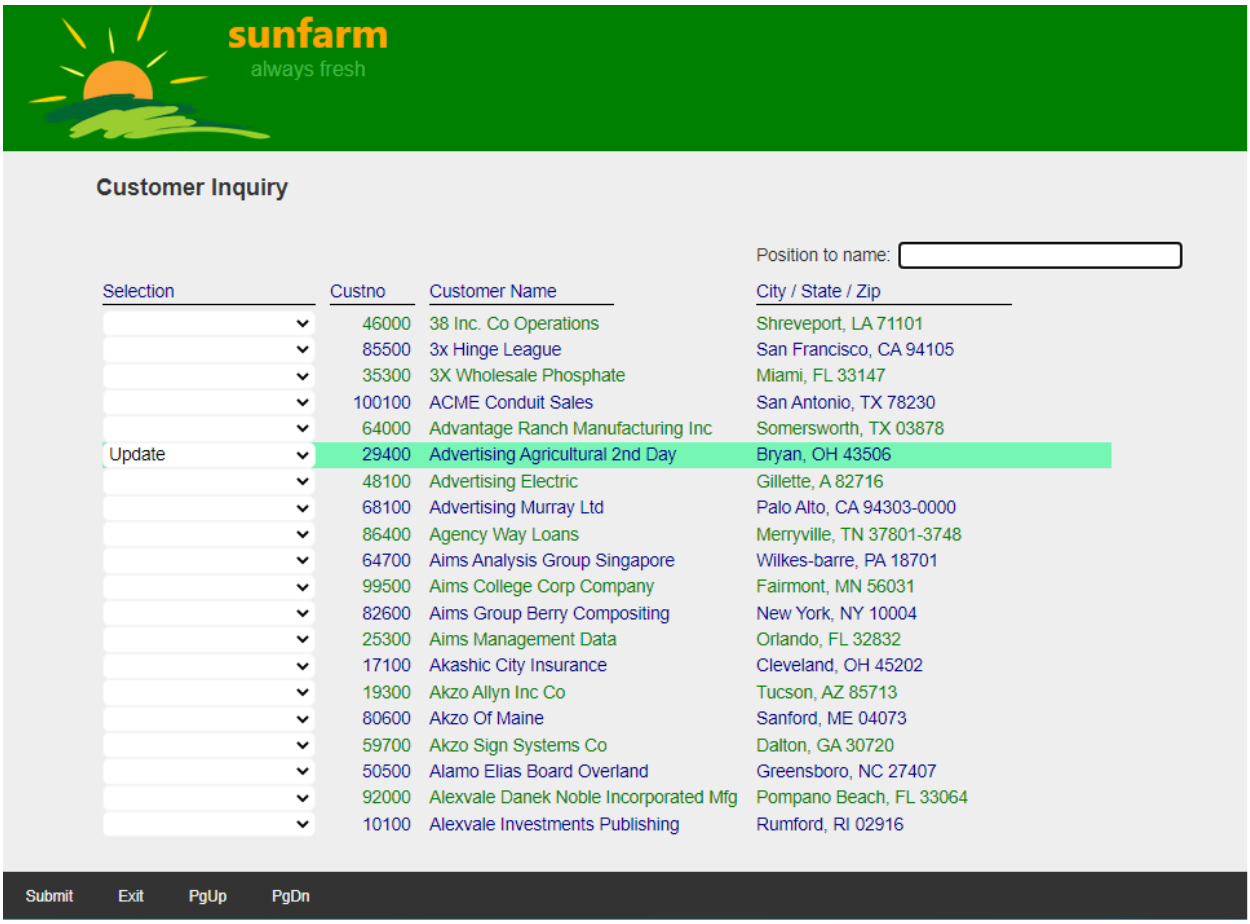
What this Guide *is not*

1. This Guide is not an introduction to ASNA Core Foundation
2. This is not a Tutorial.
3. Cloning the repository is not enough. There are ASNA references (or source Projects) that need to be installed.

Specifically, the goal is to transform: Customer Inquiry Page:



To this modern look:



Customer Maintenance Page:

Enter
F3
Prompt
New
Delete
Cancel

contreras M5 Customer Maintenance 11/05/20 17:52:40

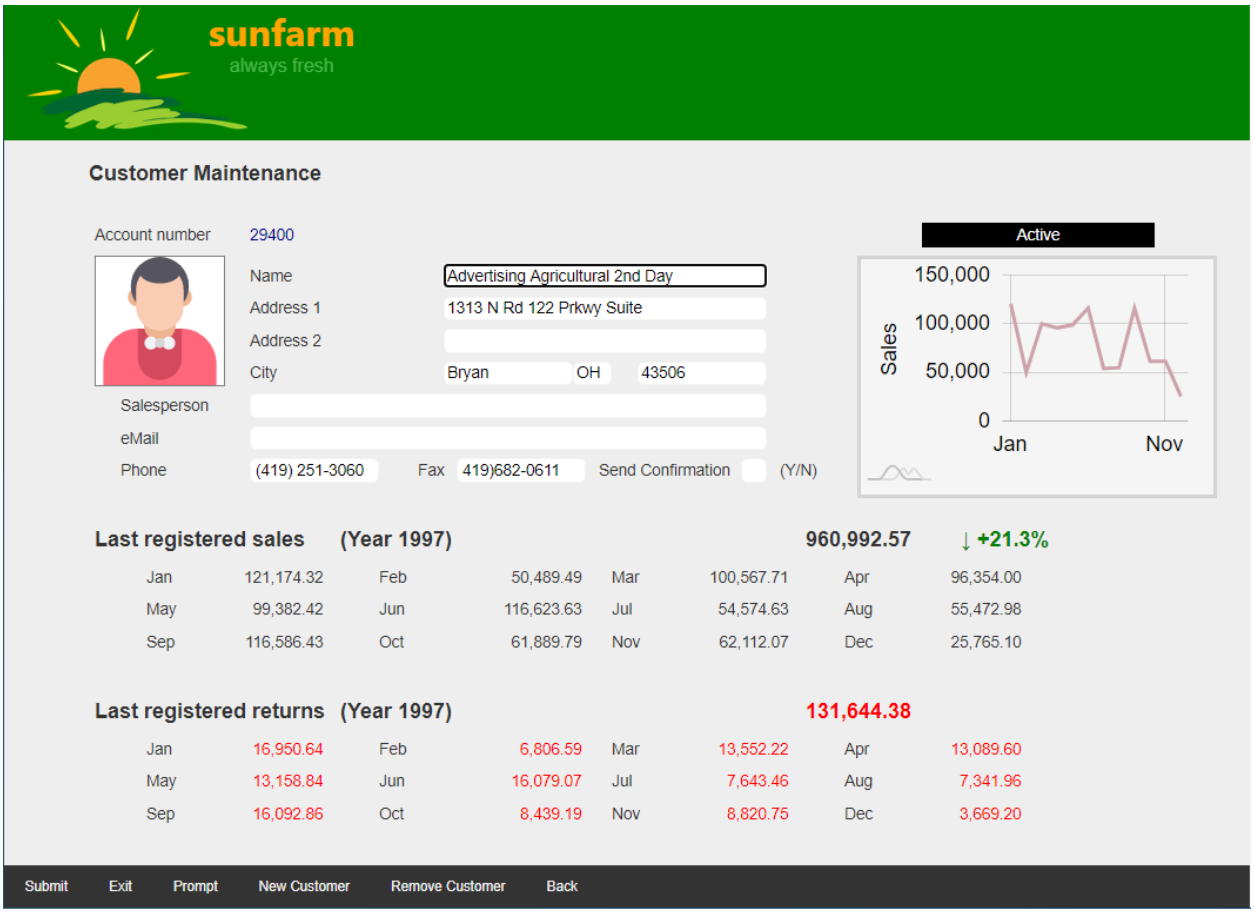
46000 Akzo Allyn Inc Co

Name: Akzo Allyn Inc Co
Address: Bldg S. Belt 1x Bld 2226
City: Tucson
State/Zip: AZ (F4) 85713

Fax: 602)251-3271
Phone: (602) 258-7898
Status: (F4)
Contact Name:
Contact eMail:
Send Confirmation: (Y/N)

F4=Prompt F6=New customer F11=Remove customer F12=Cancel

To this page where more useful information is displayed



Data Files used

The IBMi data files were also migrated to a Microsoft SQL instance.

Microsoft SQL Server Management Studio has been used to show file definitions and query results.

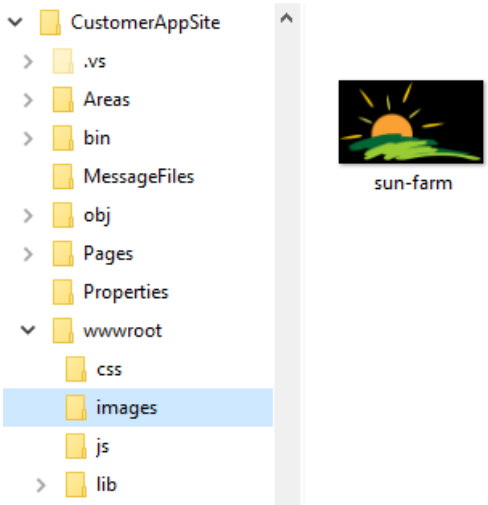
The DataGate database in $\SunFarm\CustomerAppLogic\MyJob.cs named ***NancySQL*** points to a Microsoft SQL instance where the Data Files have been migrated.

Adding branding: SunFarm Logo as a heading on all pages.

The heading is produced by and image, text added with proper CSS and a solid background.

The file used is a PNG raster image as shown, with transparency. The name of the file is “sun-farm.png”

The place where static resources are placed is under the `wwwroot` folder:



Out of the box, the folder ***wwwroot/css*** already has a CSS file named ***site.css***.
It is empty.

Let’s add some styles that will allow us to position and define the look of our logo elements

```
#logo-banner {
    height: auto;
    background-color: green;
}

#logo {
    padding-left: 1em;
}

#logo-text {
    display: inline-block;
    vertical-align: top;
    margin-left: -41px;
}

#logo-title {
    color: orange;
    font-size: xx-large;
    font-family: system-ui;
    font-weight: bolder;
}

#logo-subtitle {
    text-align: center;
    color: lightgreen;
    font-size: medium;
    opacity: 0.5;
}
```
[1]

The markup that we need is standard HTML which we will place in the file:

`$SunFarm\CustomerAppSite\Areas\CustomerAppViews\Pages\_ViewStart.cshtml.cshtml`

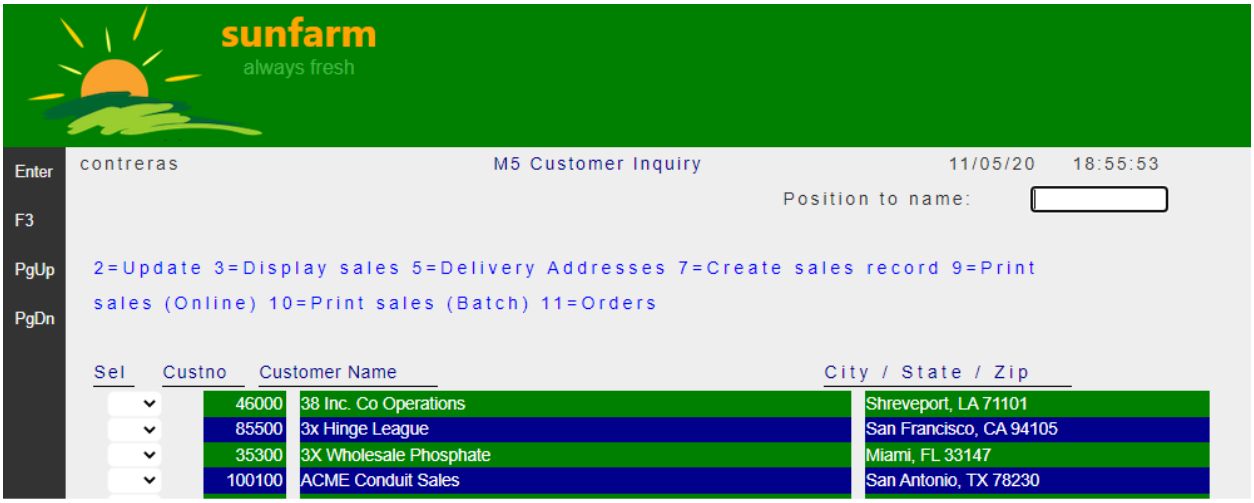
This file shared with all the Pages in the View file structure.

```
@{
    Layout = "_Layout";
}

<div id="logo-banner">
    <img id="logo" src="~/images/sun-farm.png" />
    <div id="logo-text">
        <div id="logo-title">sunfarm</div>
        <div id="logo-subtitle">always fresh</div>
    </div>
</div>
```

The Application will show the branded heading on top of all pages, as shown below:

---

[1] Commit: “Logo resources”.



[2]

Navigation menu (formerly DDS File Active Function Keys)

The two pages we are enhancing, are formed by records described by `CUSTDSP` Displayfile.

All Display files are described using Razor Page syntax, where ExpoTags runtime support augments with DDS-like tag-helpers.

The Expo tag-helper that represents the *"DDS File Active Function Keys"* is named ***DdsFunctionKeys*** (tag-helper).

The default markup for Monarch Migrated Displayfiles shows the ***DdsFunctionKey*** tag-helper without any attributes (at the top of each Displayfile, inside the ***DdsFile*** tag-helper):

```
<form id="MonarchForm" method="post">
    <DdsFile DisplayPageModel="Model">

        <DdsFunctionKeys />

        <main role="main" class="display-element-uninitialized">
            .
            .
            .
```

[2] Commit: "Branding all pages"

Let's change the ***Location*** attribute to the value "HorizontalBottom":

```
<DdsFunctionKeys Location="HorizontalBottom" />
```

The Navigation Menu looks much better at the bottom of the Page:

Instead of using aid key, you may want to use the mover vernacular form function key, something like: "Exit" equivalent to the **F3** aid key being pressed at the keyboard.

Note: experienced users may still use the ***Fxx*** key, if there is a physical keyboard that produces such key-code.

[3]

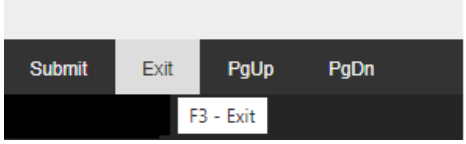
[3] Commit: "Navigation menu customization"

Re-labeling the Menu Options

Expo Tag-helpers that represent options at the DDS File (*DdsFile*) level and DDS Records (*DdsRecord* & *DdsSubfileControl*) have, have an attribute called ***KeyNames***, where we can override the default active key labels.

```
KeyNames="ENTER 'Submit'; F3 'Exit'; PageUp '◄ Page'; PageDown 'Next ►';"
```

Where the value in single-quotes re-labels the text for the key name that precedes it. Instead of “ENTER” we show ‘Submit’; instead of “F3” we show ‘Exit’, etc.



Note: The labels for PgUp and PgDn should have been replaced. This bug has been reported to R&D.

Color effects that worked on *green-screen* are displeasing on a modern *blue-screen* Browser Pages

DDS on the IBMi usually renders on a terminal with dark background. Text screen attributes such as REVERSE-IMAGE produced a subtle lighter green background which some considered pleasant on the eye to highlight text, such as record on a subfile.

REVERSE-IMAGE on a typically white/pale background Browser Page would produce a very heavy block on text when rendered.

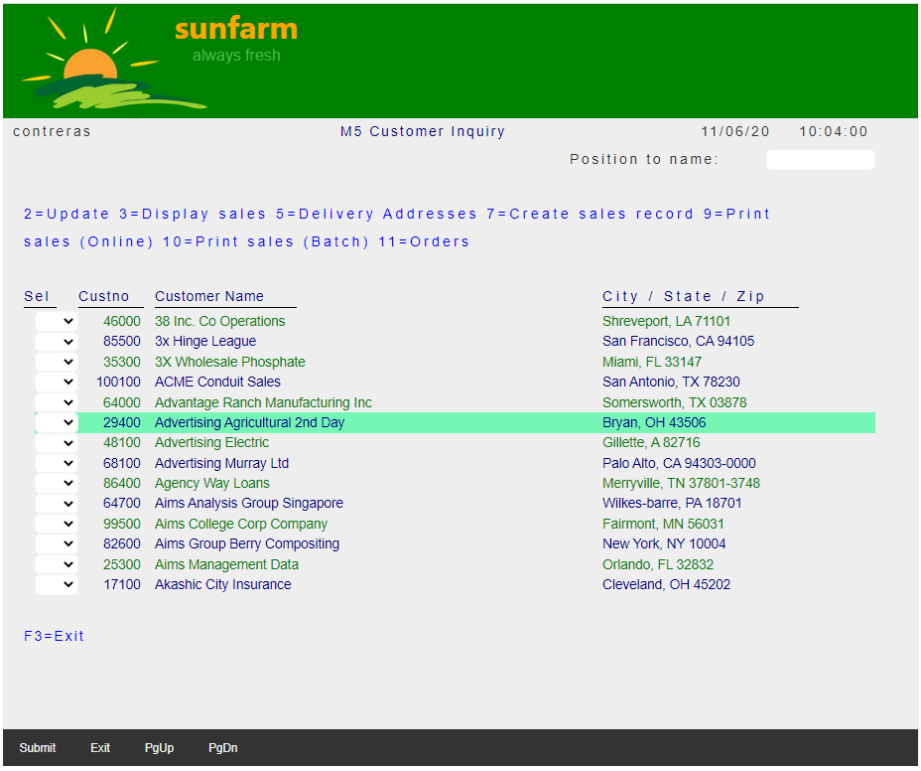
As you can see on the off-the-shelf migration for “M5 Customer Inquiry” Page, the subfile records are rendered with a color that is displeasing to the eye.

The reverse-image display attribute DDS keyword **DSPATR(RI)** gets translated to **InvertFontColors=“*True"** tag-helper attribute.

Removing such attribute produces a much pleasant output.

$\SunFarm\CustomerAppSite\Areas\CustomerAppViews\Pages\CUSTDSPF.cshtml
Lines 51, 52 and 53:[4]

```
<div Row="8" RowSpan="@SFLC_SubfilePage * @SFLC_SubfileRowsPerRecord">
    @for (int rrn=0, row = 8; rrn < Model.SFLC.SFL1.Count; rrn++, row +=
@SFLC_SubfileRowsPerRecord)
    {
        <DdsSubfileRecord RecordNumber="rrn" For="SFLC.SFL1">
            <div IsGridRow>
                <DdsCharField Col="2" For="SFLC.SFL1[rrn].SFCOLOR" VisibleCondition="*False"
VirtualRowCol="@row,2" tabIndex=1 />
                <DdsDecField Col="4" For="SFLC.SFL1[rrn].SFSEL" VirtualRowCol="@row,4"
EditCode="Z" ValuesText="'0','2','3','5','7','9','10','11'" tabIndex=2 />
                <DdsDecField Col="7+1" For="SFLC.SFL1[rrn].SFCUSTNO" VirtualRowCol="@row,7"
Color="Green : !61 , DarkBlue : 61" InvertFontColors="*True" EditCode="Z" Comment="CUSTOMER
NUMBER" />
                <DdsCharField Col="14+1" For="SFLC.SFL1[rrn].SFNAME1" VirtualRowCol="@row,14"
Color="Green : !61 , DarkBlue : 61" InvertFontColors="*True" />
                <DdsCharField Col="55+1" For="SFLC.SFL1[rrn].SFCSZ" VirtualRowCol="@row,55"
Color="Green : !61 , DarkBlue : 61" InvertFontColors="*True" Comment="CITY-STATE-ZIP" />
            </div>
        </DdsSubfileRecord>
    }
</div>
```



[4] Commit: “Remove Reverse-image on Subfile records”

Removing Redundant green-screen typical Information

Terminal program Displays usually included information about system Date and Time. Often the username was also included in most pages.

Modern devices that provide Web Browsers have already a place for this information to be retrieved.

Other green-screen constants that provided assistance to the user, such as which aid keys were active, are now part of the “tooltip” information that is already part of the Navigation Menu.

Lastly, constants that listed available options, in this case for selection of records in the subfile, can be placed in a better place.

The next commit will:
1. Remove User, Date and Time fields.
2. Remove F3=Exit constant.
3. Move the constant “2=Update 3=Display sales …” to the “SFLC.SFL1[rrn].SFSEL" field as Value-Text (or pull down selection option labels)

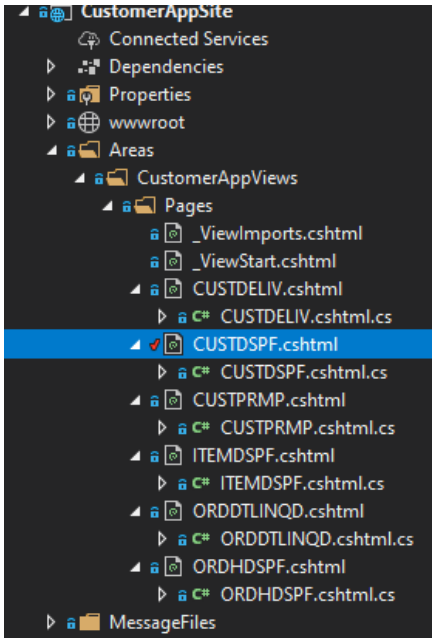
```
$\SunFarm\CustomerAppSite\Areas\CustomerAppViews\Pages\CUSTDSPF.cshtml

<div Row="1">
    <DdsConstant Col="2" Text="*USER" />
    <DdsConstant Col="31+1" Text="M5 Customer Inquiry" Color="DarkBlue" />
    <DdsConstant Col="64+1" Text="*DATE" />
    <DdsConstant Col="73+1" Text="*TIME" />
</div>


<div Row="4">
    <DdsConstant Col="3" Text="2=Update   3=Display sales    5=Delivery Addresses    7=Create
sales record     9=Print" Color="Blue" />
</div>
<div Row="5">
    <DdsConstant Col="3" Text="sales (Online)  10=Print sales (Batch)    11=Orders"
Color="Blue" />
</div>

<DdsRecord For="KEYS" KeyNames="ENTER 'Submit'; ">
    <div Row="23">
        <DdsConstant Col="3" Text="F3=Exit" Color="Blue" />
    </div>
</DdsRecord>
```

The Subfile field for the options, deserves further explanation.

We have two pieces of information:
1. The option code (char value), that is: <nothing>, 2, 5, 7, 9, 10 and 11.
2. The corresponding labels, such as: “Update” “Display sales” “Delivery Addresses” etc.

Each ASP Razor Page is defined by two files: the Markup file (extension *.cshtml*) and the corresponding Model file (extension *.cshtml.cs*).

For convenience, Visual Studio Solution Explorer, shows the Model file under the Markup file in the Website file structure:



Visual Studio intellisense, allows to jump back and forth, between symbols defined in the Markup and the Model. For example, positioning the cursor in the markup on top of DdsDecField For="SFLC.SFL1[rrn].SFSEL" and pressing F12, will take you to the Model's definition for the SFLC (Subfile record Controller) 's field SFLSEL (in blue):

```
public class SFL1_Model : SubfileRecordModel
{
    [Char(1, Protect = "*True")]
    public string SFCOLOR { get; set; }

    [Values(typeof(Decimal),"00","02","03","05","07","09","10","11")]
    [Dec(2, 0)]
    public decimal SFSEL { get; set; }

    [Dec(6, 0)]
    public decimal SFCUSTNO { get; private set; } // CUSTOMER NUMBER

    [Char(40)]
    public string SFNAME1 { get; private set; }

    [Char(25)]
    public string SFCSZ { get; private set; } // CITY-STATE-ZIP
}
```

Note that, in addition to C# decimal type, the field SFLSEL is decorated with Dec and Values attributes.

Dec attribute further defines the decimal as one with fixed precision and decimal positions. *Values* attributes define the valid values for the field. The position of the list of valid values is important, since it will be matched with the *ValuesText* tag-helper attribute.

Back on the Markup file: `…\Areas\CustomerAppViews\Pages\CUSTDSPF.cshtml`

```
<DdsDecField Col="4"
  For="SFLC.SFL1[rrn].SFSEL"
  VirtualRowCol="@row,4"
  EditCode="Z"
  ~~ValuesText="'0','2','3','5','7','9','10','11'"~~
  tabIndex=2
/>
```

Let's change ValuesText as follows:

```
<DdsDecField Col="4"
  For="SFLC.SFL1[rrn].SFSEL"
  VirtualRowCol="@row,4"
  EditCode="Z"
  ValuesText="' ','Update','Display sales','Delivery Addresses','Create sales
record','Printsales (Online)','Print sales (Batch)','Orders'"
  tabIndex=2
/>
```

[5]

Now we need to push the rest of the fields to the right, to make it look nicer.

There are several ways to accomplish this. What we will use, is the trial-and-error technique.

---

[5] Commit: "Subfile selection options as pull-down options"

To avoid having to recalculate the column positions for the fields in the subfile, we can add a constant numeric value to each of the fields to the right: “SFLC.SFL1[rrn].SFCUSTNO”, “SFLC.SFL1[rrn].SFNAME1" and “SFLC.SFL1[rrn].SFCSZ”.

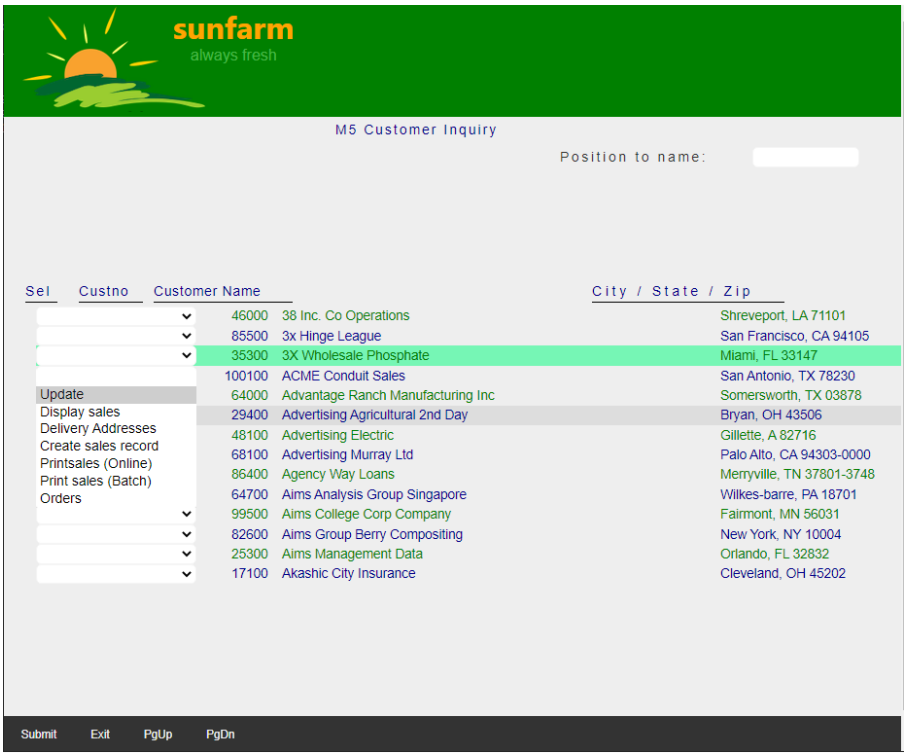
Let’s add *10+* … no, *15+* … no *12+*

The markup would look like this (for clarity I eliminated part of each line, replaced by …):

```
<DdsSubfileRecord RecordNumber="rrn" For="SFLC.SFL1">
    <div IsGridRow>
        <DdsCharField Col="2" For="SFLC.SFL1[rrn].SFCOLOR" ...
        <DdsDecField Col="4" For="SFLC.SFL1[rrn].SFSEL" ...
        <DdsDecField Col="12+7+1" For="SFLC.SFL1[rrn].SFCUSTNO" ... Comment="CUSTOMER NUMBER" />
        <DdsCharField Col="12+14+1" For="SFLC.SFL1[rrn].SFNAME1" ...  />
        <DdsCharField Col="12+55+1" For="SFLC.SFL1[rrn].SFCSZ" ... Comment="CITY-STATE-ZIP" />
    </div>
</DdsSubfileRecord>
```

As you change the markup, you may refresh the running page in the Browser, until it looks how you want it.

Note that the Col attribute tag-helper, already came with an expression. Cocoon Displayfile Migration Agent took advantage of the expression to indicate the ***original*** column value (from DDS and adjustments it had to do to prevent overlaps — due to HTML borders and padding—



Before we continue, let’s get rid of the vertical gap, between Position to name: (Row=2) and the headings of the Subfile (Row=7).

We accomplish this by changing ~~Row=“7”~~ to Row=“3” and ~~Row=“8”~~ to Row=“4”

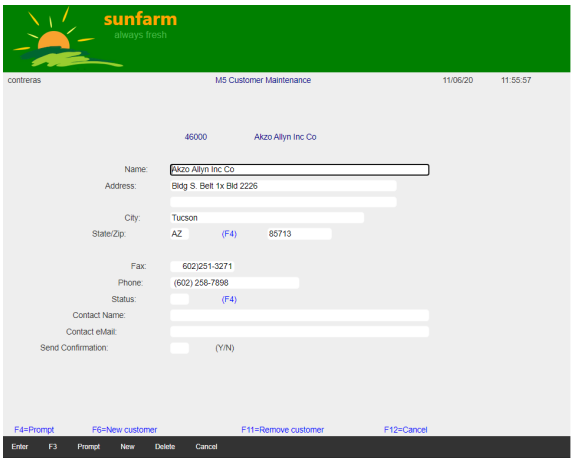
6

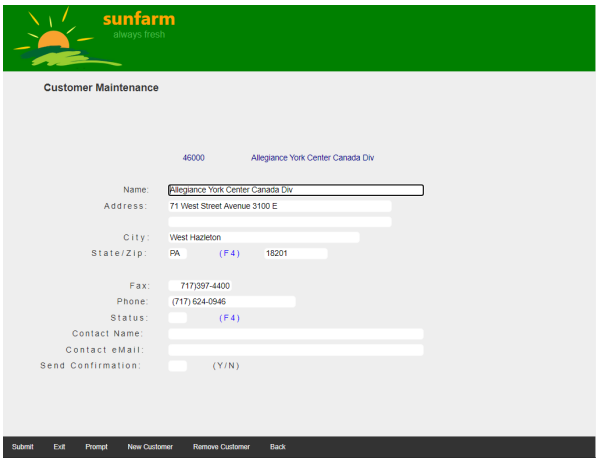
Stretching constant label's Text

If you had not noticed, the text on the constants is rendered with more white space between characters. Particularly noticeable is the subfile column heading "City / State / Zip".

ASNA Expo DdsConstant tag-helper has an attribute called ***StretchConstantText*** which defaults to ***true***. That attribute can be overridden at the Record level.

That attribute exists to match better the constant text alignment that green-screen developers used, particularly when splitting words in multiple DDS constants, or when right justifying (manually) text on the screen.





The left image shows Stretching Off and the right Stretching On. Notice how the right page improves the label right-alignment.

Most likely, what the green-screen designer intended for this page was to night-align the constants such that the ":" aligned close to the edit boxes.

Let's assume we want to reproduce the original designer intentions. We can make it very close to the intended alignment by:

1. Turning off the text-stretch feature.
2. Make sure the column span matches the original design.
3. Using a CSS style to align the text.

To turn off the text-stretch feature, all we have to do is add the attribute StretchConstantText=false to the record.

```
<DdsRecord For="CUSTREC" StretchConstantText=false KeyNames="ENTER 'Submit'; …
```

---

[6] Commit: "Push subfile up"

Matching the column span is achieved by adding ColSpan attribute with the special value ="-1" indicating that we want exactly the length of the constant text to be used.

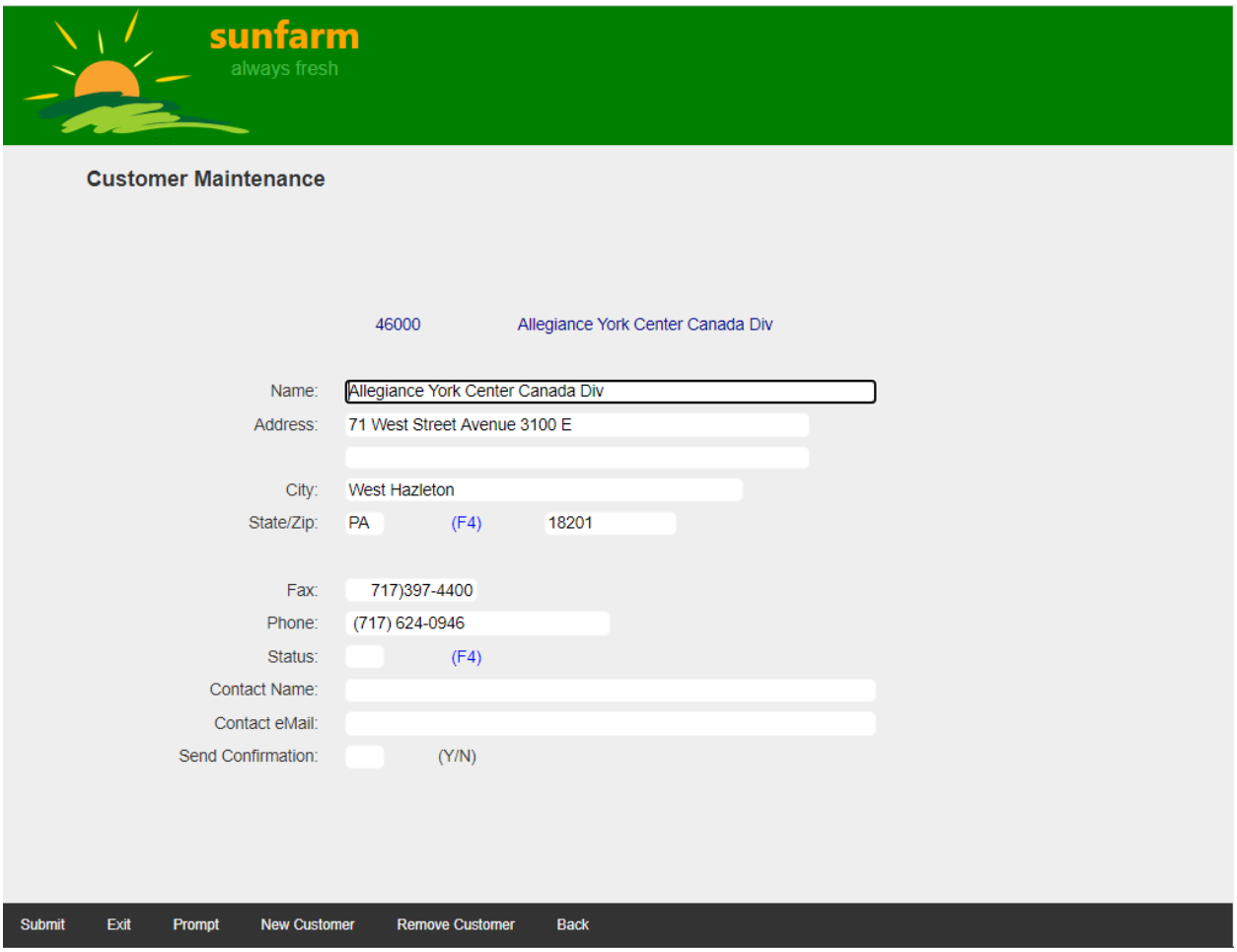
Lastly, we add a CSS class attribute to all the DdsConstant tag-helpers in the record set to "right-aligned-constant":

```
<div Row="7">
    <DdsConstant Col="20" ColSpan="-1" class="right-aligned-constant" Text="Name:" />
    .
    .
    .
</div>
```

Where the style right-aligned-constant is the following simple CSS added to the $ \SunFarm\CustomerAppSite\wwwroot\css\site.css file:

```
.right-aligned-constant {
    text-align: right;
}
```

The resulting screen is now:[7]

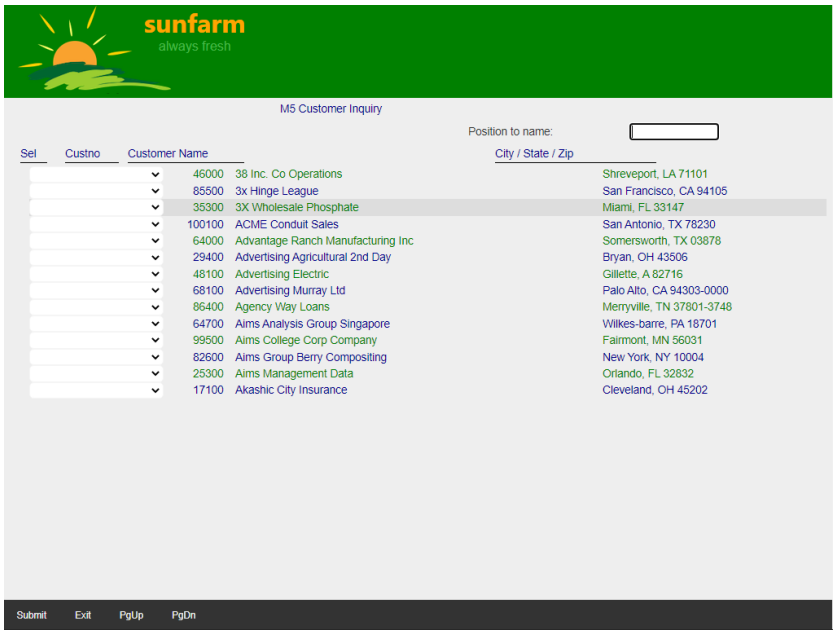


[7] Commit: "Right-aligned Original design"

For the Customer Inquiry (first page) we will reset the constant stretching effect, by adding StretchConstantText attribute:

```
<DdsSubfileControl For="SFLC"
  StretchConstantText=false
  KeyNames="ENTER 'Submit'; F3 'Exit'; PageUp '◄ Page'; PageDown 'Next ►';"
  SubfilePage="@SFLC_SubfilePage"
  CueCurrentRecord=true ClickSetsCurrentRecord=true
>
```

Almost complete:



Page Title with standard HTML/CSS

We can add HTML/CSS when Grid alignment serves no purpose. For example, we want the Title of the page to show aligned to the Page's left and using standard CSS styles.

Add the following CSS style to file

```
$\SunFarm\CustomerAppSite\wwwroot\css\site.css
```

```
#page-title {
    font-size: large;
    padding-left: 4.0em;
    padding-top: 1em;
    font-weight: bold;
}
```

And replace the following row / col constant definition in the CUSTDSP.chtml markup:

~~ ~~
~~<DdsConstant Col="31+1" Text="M5 Customer Inquiry" Color="DarkBlue" />~~
~~ ~~

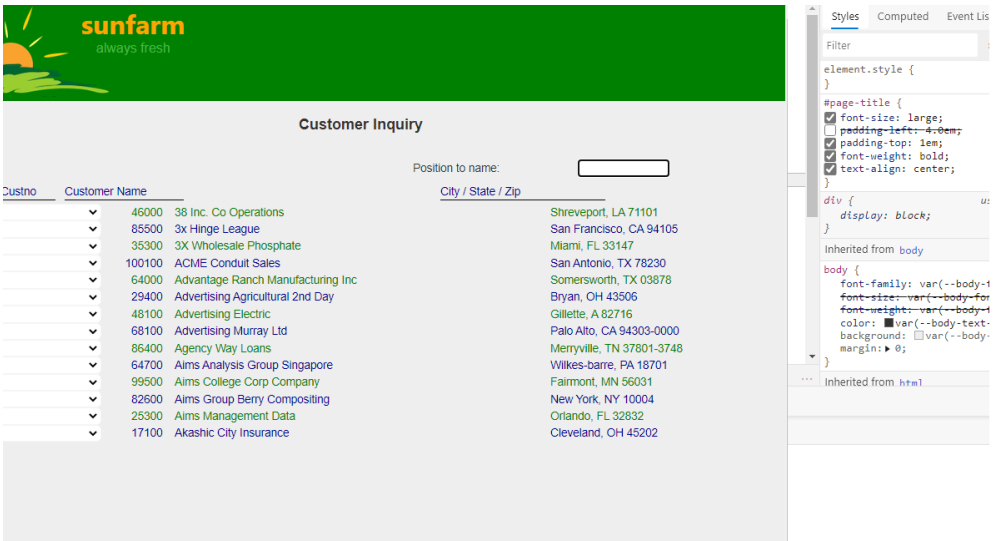
With this standard markup:

```
<div id="page-title">Customer Inquiry</div>
```

[8]

This may be particularly important when centering titles on a Page.

You may even user the Browser's developer tools to experiment with different styles:



[8] Commit: "Replacing Page Title"

Grid Column Span Adjustment

Field's starting positions are very accurately identified on the page based on the original DDS row, col positions. But ending column positions are harder.

The default Font for Expo Displayfiles is of type ***variable-pitch***, meaning that the width of characters varies according to the Font's designer's stroke used. Typically the letter "i" uses a lot less character width that an upper case "M".

Green-screen page designers used a Font that is of type fixed-pitch, meaning that the width of ALL characters is the same.

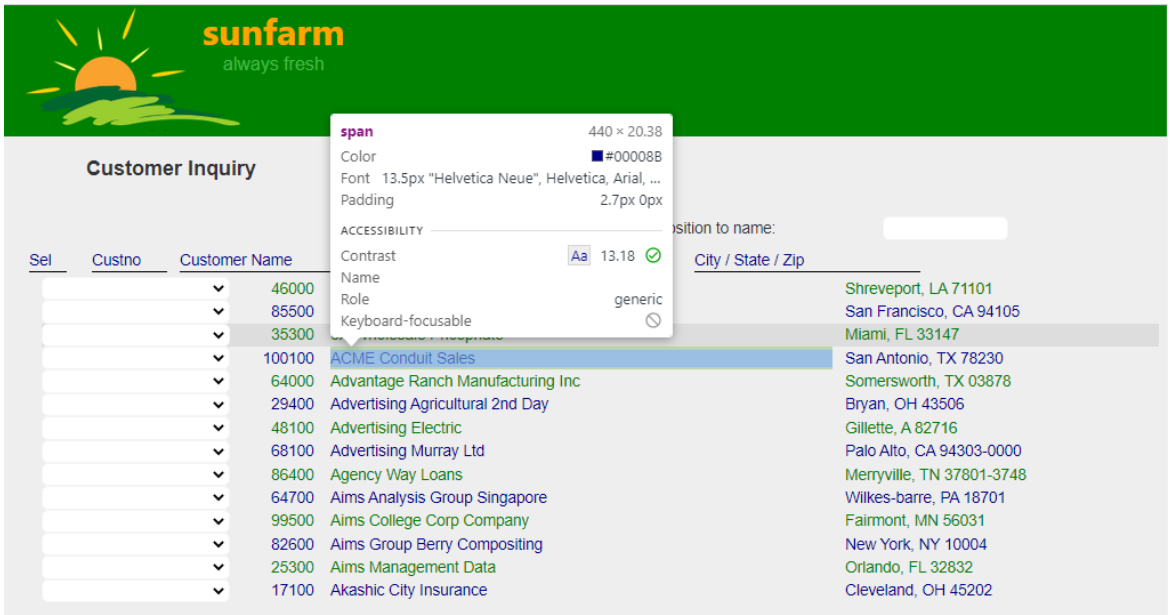
A green-screen label starting at column 5 with the constant "THIS CONSTANT" (thirteen characters) is guaranteed to end at column position 17. That is, the end-column position can be precisely computed by starting-position + field-length.

This no longer works on Browser fonts (even with those so-called *Monospaced*).

Monarch Cocoon Displayfile for Core Agent will use the length of the field of constant to compete the Grid column span (ending position), but with a fudge-factor to account for the use of Web fonts.

We can adjust that calculation on the field level.

Consider the column for SFNAME1 Subfile on the Customer Inquiry Page:

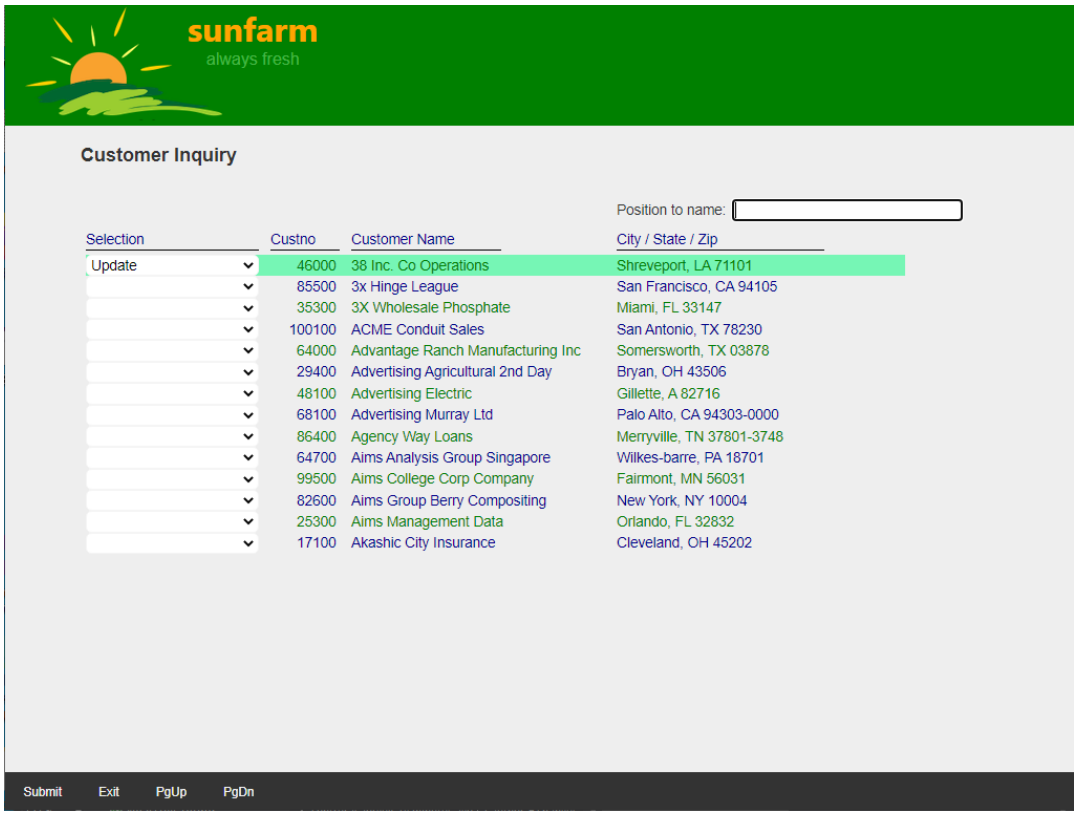


Notice how the field is defined as 40 characters, but most of the record's field value will likely not use the full 40.

We can override the column-span calculation by adding the tag-helper attribute ColSpan:

```
<DdsCharField Col="12+14+1" ColSpan="30" For="SFLC.SFL1[rrn].SFNAME1" VirtualRowCol="@row,14"
Color="Green : !61 , DarkBlue : 61" />
```

This way we can reduce Grid column positions and complete adjusting our elements on the Page to achieve the following look:
[9]



When Changes on Razor Pages is not enough

There comes a time when to continue enhancing pages, it is necessary to change the Business Logic.

Since we have been able to compress the screen while aligning elements and eliminating redundant items, we are left with unused real-estate.

The Business Logic is writing fourteen records at a time, now we can fit at least twenty (a nice rounding number) or even more if the majority of users may have higher resolution devices.

Let's assume we want to increase the record count from fourteen to twenty.

---

[9] Commit: "Fourteen records subfile"

Displaying Twenty records per page in the Subfile

Two changes are needed:
1. Write twenty records to the subfile in the Business Rules.
2. Expand the record count to show in the Razor Page's Subfile Controller

We will use blue color for the code that needs to be added. Red strikeout for code that needs to be removed.

To change the Business Logic, we need to change the following class (program):

`$\SunFarm\CustomerAppLogic\CUSTINQ.cs`

```
public partial class Custinq : ASNA.QSys.HostServices.Program
{
    const int SFLC_SubfilePage = 20;

    protected Indicator _INLR;
.
.
.
    void LoadSfl() // Line 519
    {
        _IN[61] = '0'; //Start with green.
        _IN[90] = '1'; //Clear the subfile.
        CUSTDSPF.Write("SFLC", _IN.Array);
        _IN[76] = '0'; //Display records.
        _IN[90] = '0';
        sflrrn = 0;
        _IN[77] = CUSTOMERL2.ReadNext(true) ? '0' : '1';
        //-----------------------------------------------------------
        while (!(bool)_IN[77] && (sflrrn < 14 SFLC_SubfilePage))
        {
.
.
.
        //*********************************************************************
        //  Read Backwards for a PageDown
        //*********************************************************************
        void ReadBack() // Line 558
        {
            _IN[76] = '0';
            _IN[77] = '0';
            X = 0;
            CUSTDSPF.ChainByRRN("SFL1", 1, _IN.Array); //Get the top name and
            CMNAME = SFNAME1; // number.
            CMCUSTNO = (decimal)SFCUSTNO;
            CUSTOMERL2.Chain(true, CMNAME, CMCUSTNO);
            _IN[76] = CUSTOMERL2.ReadPrevious(true) ? '0' : '1';
            while (!(bool)_IN[76] && (X < 14 SFLC_SubfilePage))
            {
                /* EOF or full s/f. */
                X += 1;
                _IN[76] = CUSTOMERL2.ReadPrevious(true) ? '0' : '1';
            }
            if ((bool)_IN[76])
                //Any records found?
                CUSTOMERL2.Seek(SeekMode.SetLL, new string(char.MinValue, 40));
        }
```

There are two places in CUSTINQ where the hard-coded value 14 is used, representing the records to write to the subfile.

We define a constant, and use it instead so we can adjust that number to twenty.
To expand the number of records that need to be displayed in the Subfile controller, we need to affect two files:

1. Model File: $\SunFarm\CustomerAppSite\Areas\CustomerAppViews\Pages\CUSTDSPF.cshtml.cs
2. Markup File: $\SunFarm\CustomerAppSite\Areas\CustomerAppViews\Pages\CUSTDSPF.cshtml

In the Model file, we need to update the Subfile Controller's ***Size*** attribute:

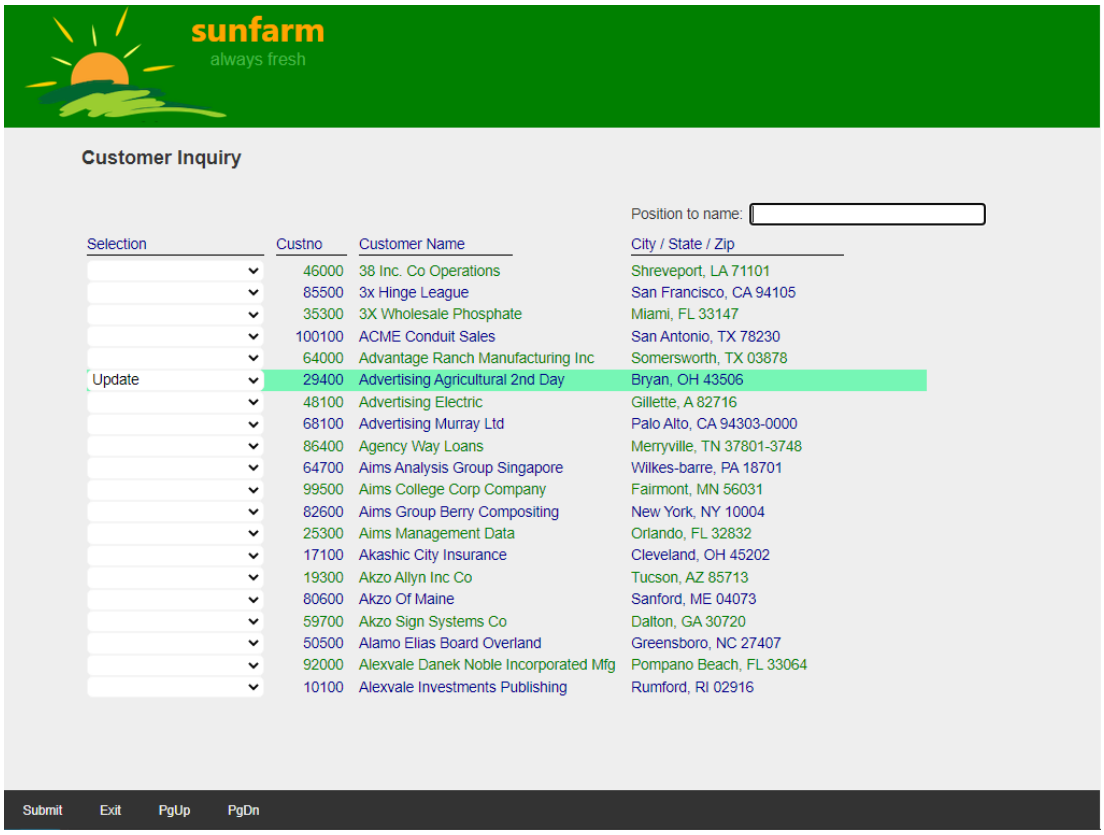
```
[
    SubfileControl(ClearRecords : "90",
        FunctionKeys = "PageUp 51:!76;PageDown 50:!77",
        DisplayFields = "!90",
        DisplayRecords = "!90",
        Size = 14 20,
        IsExpandable = false,
        EraseFormats = "CUSTREC SALESREC"
    )
]
public class SFLC_Model : SubfileControlModel
```

In the Markup file, we just need to change the value of the variable that controls the page size:

```
  @{
      int SFLC_SubfilePage = 14 20;
      int SFLC_SubfileRowsPerRecord = 1;
  }
  <DdsSubfileControl For="SFLC" StretchConstantText=false KeyNames ...
```

Build the Business Logic Project (CustomerAppLogic) …
Build the Website (CustomerAppSite) …

If all goes well, twenty records per page should show in the Subfile:
[10]



[10] Commit: “Customer Inquiry Twenty records per page”

Enhancing Customer Maintenance Page

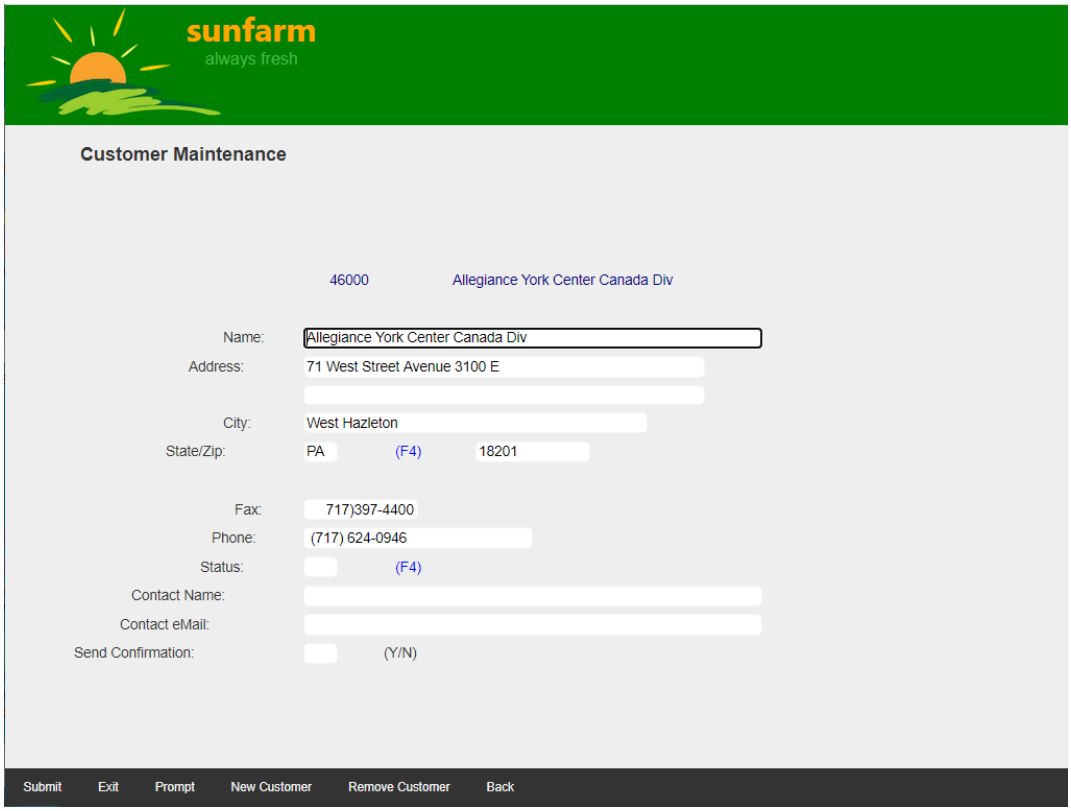
Preparation

Let's apply the same techniques like que did for the Customer Inquiry Page.

Same Markup file, different Record.

```
$\SunFarm\CustomerAppSite\Areas\CustomerAppViews\Pages\CUSTDSPF.cshtml
<DdsRecord For="CUSTREC"
```

1. Remove redundant Program Name, Date and Time
2. Improve KeyNames (Navigation Menu): 'Submit' 'Exit' 'Prompt' 'New Customer' 'Remove Customer' and 'Back'.
3. Reset StretchConstantText to false (record level)
4. Replace Page Title by standard HTML and CSS
[11]



[11] Commit: "Customer Maintenance Preparation"

When fields and constants have different styles, the colon constant separator adds unnecessary visual clutter

The default Expo colors scheme uses a white box with no borders for the field and transparent background color for the text constants. That is enough visual separation for the two elements, making the use of the colon unnecessary.

When numbers are used for non-numeric data, left alignment looks best

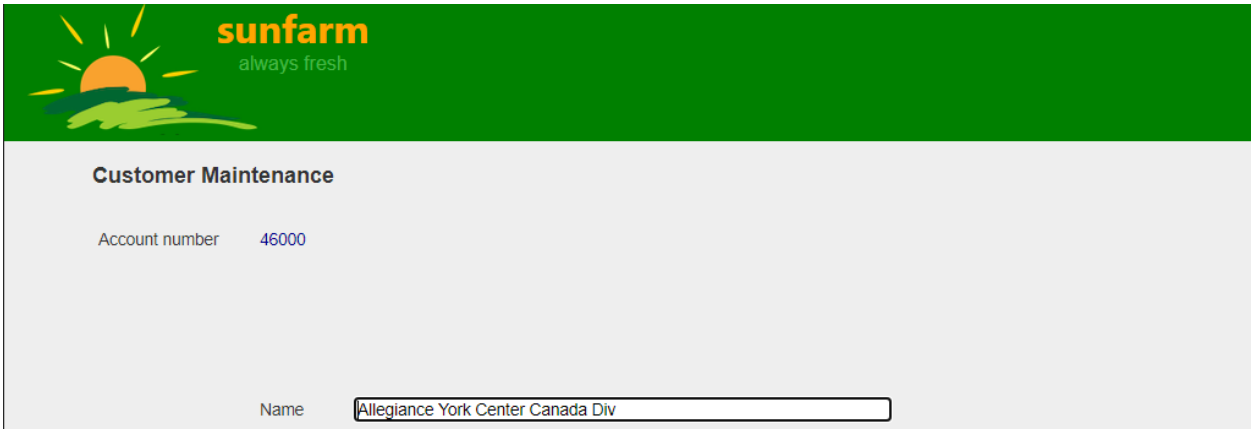
The read-only field **CUSTREC.SFCUSTNO** - customer number is rendered as a right align numeric field, but in this case it represents a code, which looks better left-aligned.

Just like we did for right-aligned-constant before let's define a CSS style for left-aligned-field

```
$\SunFarm\CustomerAppSite\wwwroot\css\site.css

.left-aligned-field {
    text-align: left;
}
```

Change the markup to show the Customer number data on row 2, aligned accordingly:
[12]

```
<div Row="2">
    <DdsConstant Col="8" Text="Account number" />
    <DdsDecField class="left-aligned-field" Col="20" For="CUSTREC.SFCUSTNO"
         VirtualRowCol="5,27" Color="DarkBlue" EditCode="Z" Comment="CUSTOMER NUMBER" />
</div>
```

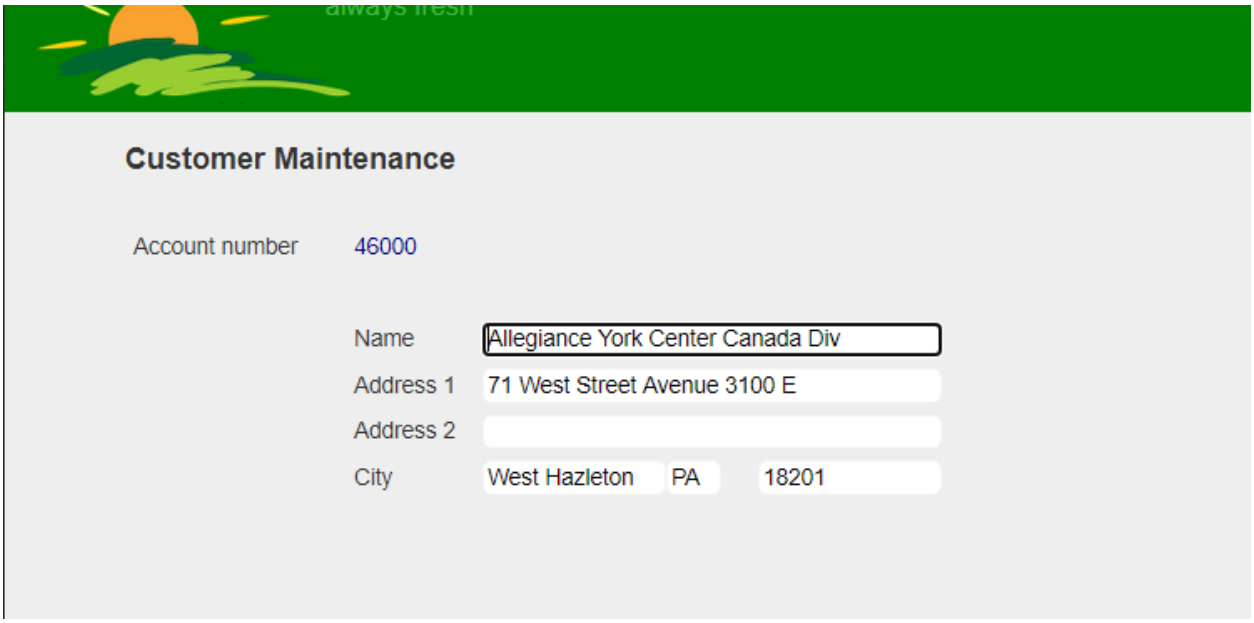


[12] Commit: "Left aligned decimal field"

Input-capable fields with equal display width

The input box on Web pages distinctively shows a box with a particular background. Making fields of equal display width produces pleasing results. Expanding (or contracting) the display width using the ColSpan attribute does not violate the field length definition. The data entry will either stop at the limit (even when there is more visual space), or scroll horizontally when the visual width is smaller than the length of the data field.

Let's make CUSTREC.SFNAME, CUSTREC.SFADDR1 and CUSTREC.SFADDR2 column spans the same: 25 positions. On the next row, we can fit City, State and Zip code information. Also move the Rows for this data section to Rows: 4, 5, 6 and 7
[13]



Note how we are leaving an area on the left to add a photo for the Customer contact.

The new Markup is:

```
<div Row="4">
   <DdsConstant Col="20" Text="Name" />
   <DdsCharField Col="27" ColSpan="25" For="CUSTREC.SFNAME" Lower="true" … PositionCursor="40"/>
</div>

<div Row="5">
   <DdsConstant Col="20" Text="Address 1" />
   <DdsCharField Col="27" ColSpan="25" For="CUSTREC.SFADDR1" Lower="true" … PositionCursor="41"/>
</div>

<div Row="6">
   <DdsConstant Col="20" Text="Address 2" />
   <DdsCharField Col="27" ColSpan="25" For="CUSTREC.SFADDR2" Lower="true" … />
</div>
```

[13] Commit: "Input capable field block"

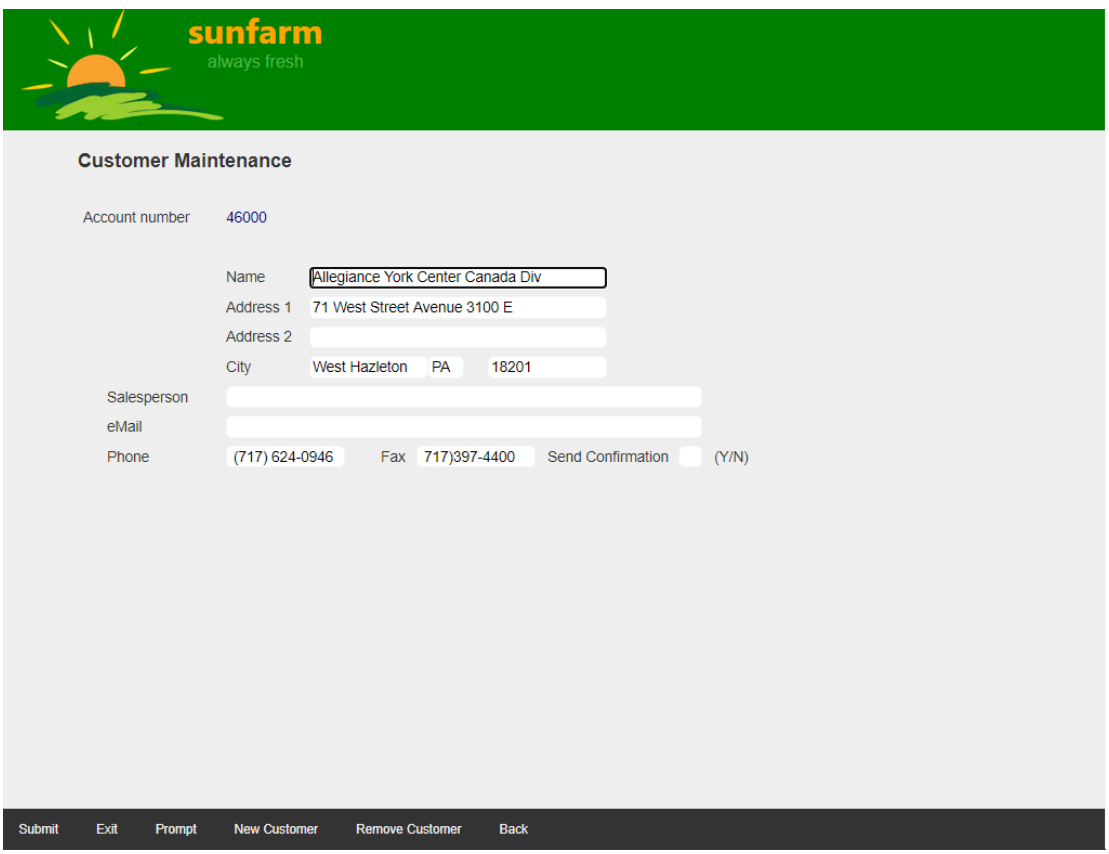
```
<div Row="7">
    <DdsConstant Col="20" Text="City" />
    <DdsCharField Col="27" ColSpan="10" For="CUSTREC.SFCITY" Lower="true" … PositionCursor="42"/>
    <DdsCharField Col="37" For="CUSTREC.SFSTATE" … PositionCursor="43" tabIndex=6 />
    <DdsCharField Col="42" For="CUSTREC.SFPOSTCODE" VirtualRowCol="11,37" tabIndex=7 />
</div>
```

Notice also that we have given CUSTREC.SFCITY field a ColSpan of 10 - when the definition of the field is Char(30)-most US cities will fit in the width provided, but if they don't the field will horizontally scroll to allow for the 30 positions max field length.

(Since we removed the "F4" constant, we should add a push button to indicate that State is *promptable* -adding such element is outside the scope of this guide-).

Move the rest of fields up and around the Customer Photo area

What used to be Rows 13, 14, 15, 16, 17 and 18 is now Rows 8, 9 and 10
[14]



Note that Y/N fields are old fashion. We will replace later for a checkbox.

---

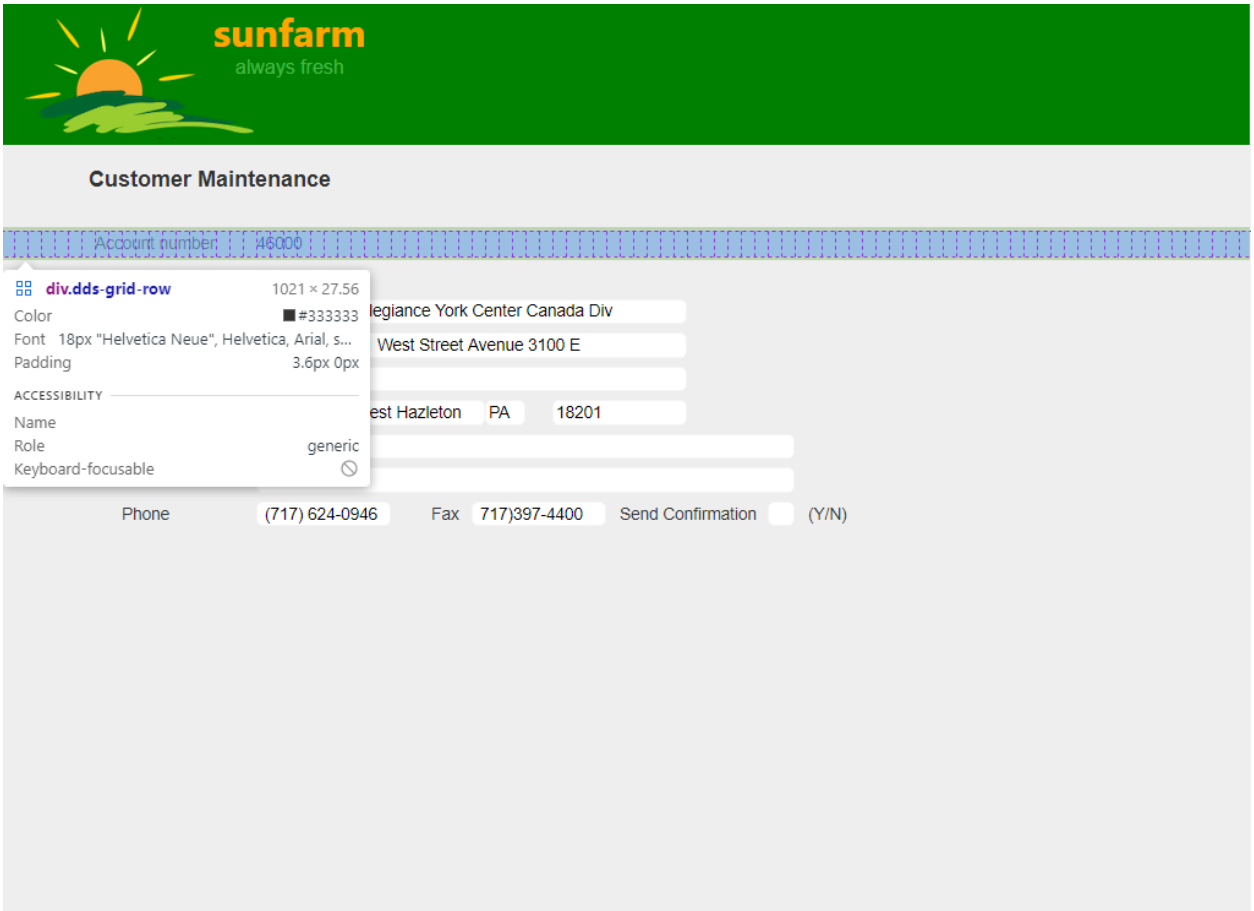
[14] Commit: "Rest of fields up and around Customer photo area"

Adding an Image to an Expo Displayfile Page

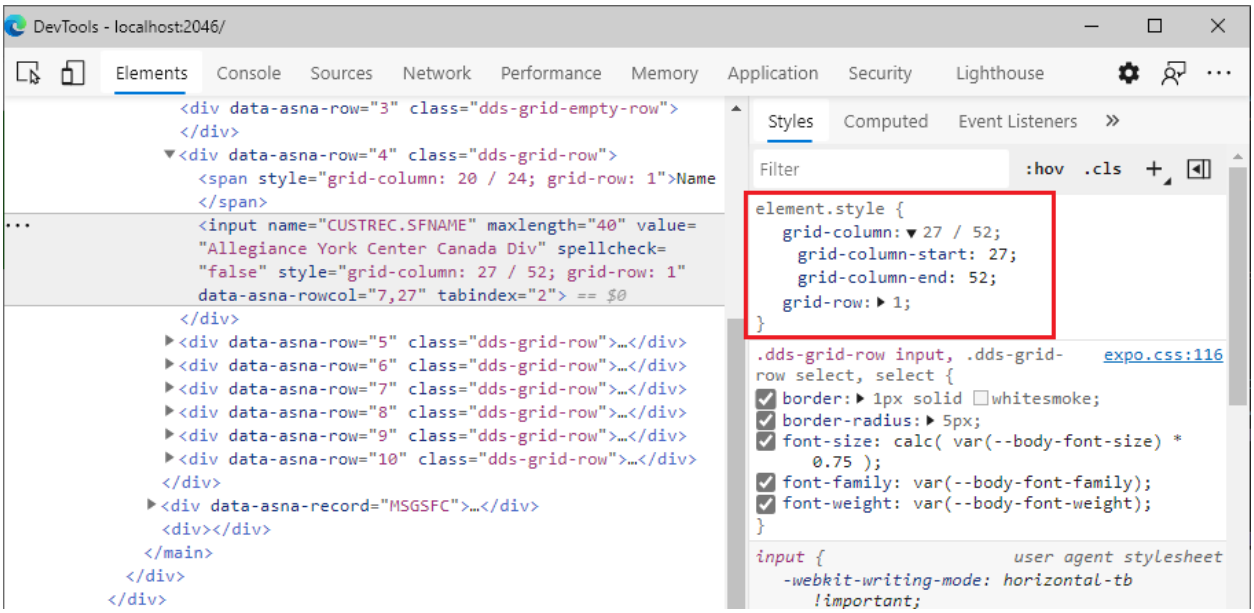
Expo Display pages are Microsoft Razor Pages where standard HTML elements may be used.

You may have noticed that we have been using **Row** attributes on a standard **div** element to indicate that the **div** container becomes a one-row **grid-display** element where we can position elements converting Col and ColSpan attributes to **grid-column** styles.

You may see this anatomy with the help of the Browser Developer tools:



And if you inspect one particular field, you will see how the grid starting column and ending column are translated to standard CSS styles.



The vertical position flows freely

Each **div** that represents a green-screen row gets rendered with a **data-asna-row** custom attribute that has the value of the **Row** attribute in the markup. This attribute has two purposes:

1. Annotate in the rendered HTML the intended row in the markup.
2. When there are row gaps (skipping the continuously ascending numbering in the markup), after the page loads completely JavaScript will run to insert rows with the CSS style **dds-grid-empty-row** which basically uses vertical spacing to fill the gap.

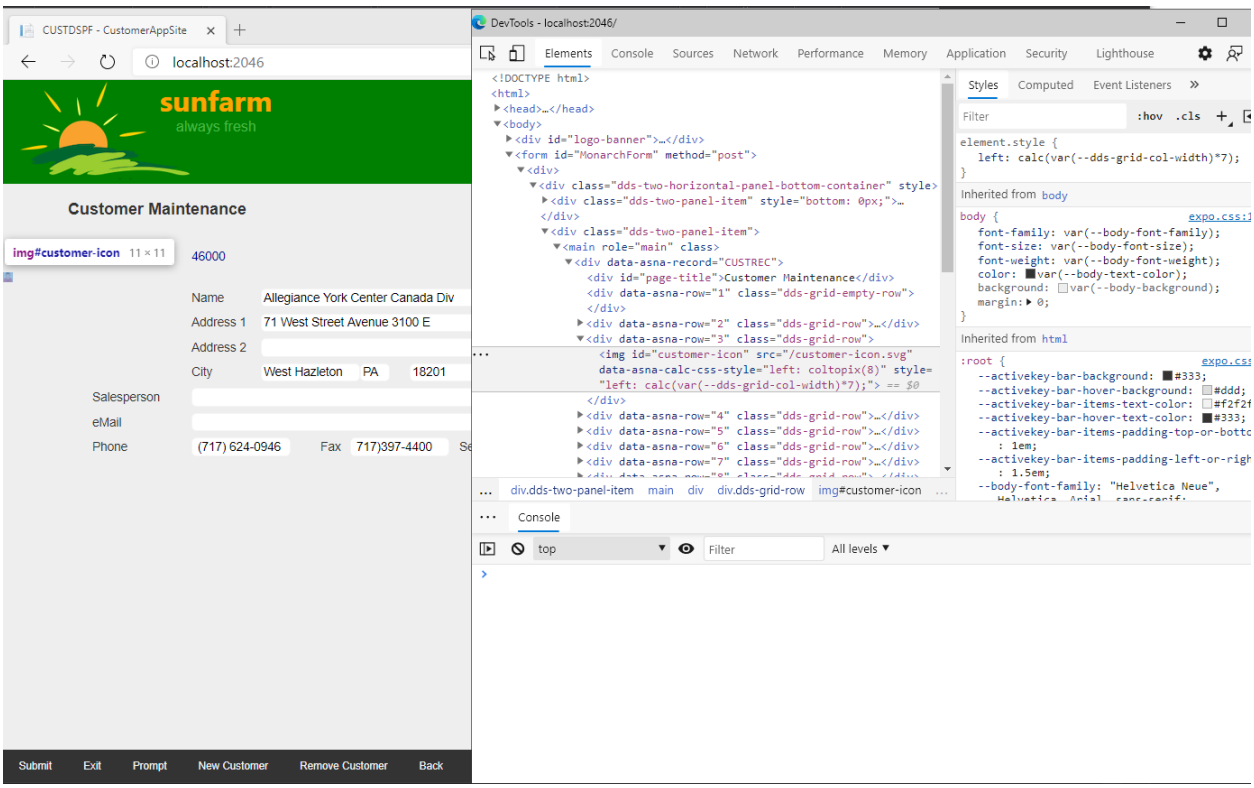
Adding other HTML elements outside of this **div Row** containers is not only perfectly compatible, but encouraged. These standard elements will just push all elements down (allowing free vertical flow).

For convenience, it is also advisable to add standard elements ***inside*** **div Row** containers. Doing so, will make sure elements are positioned at “Row” vertical boundaries and starting at the top-left position specified by a new ExpoCol attribute (similar to the **Col** attribute on *DdsCharField*, *DdsConstant* and *DdsDecField* tag-helpers you have been using so far).

Let’s add a Customer Photo placeholder on this page.

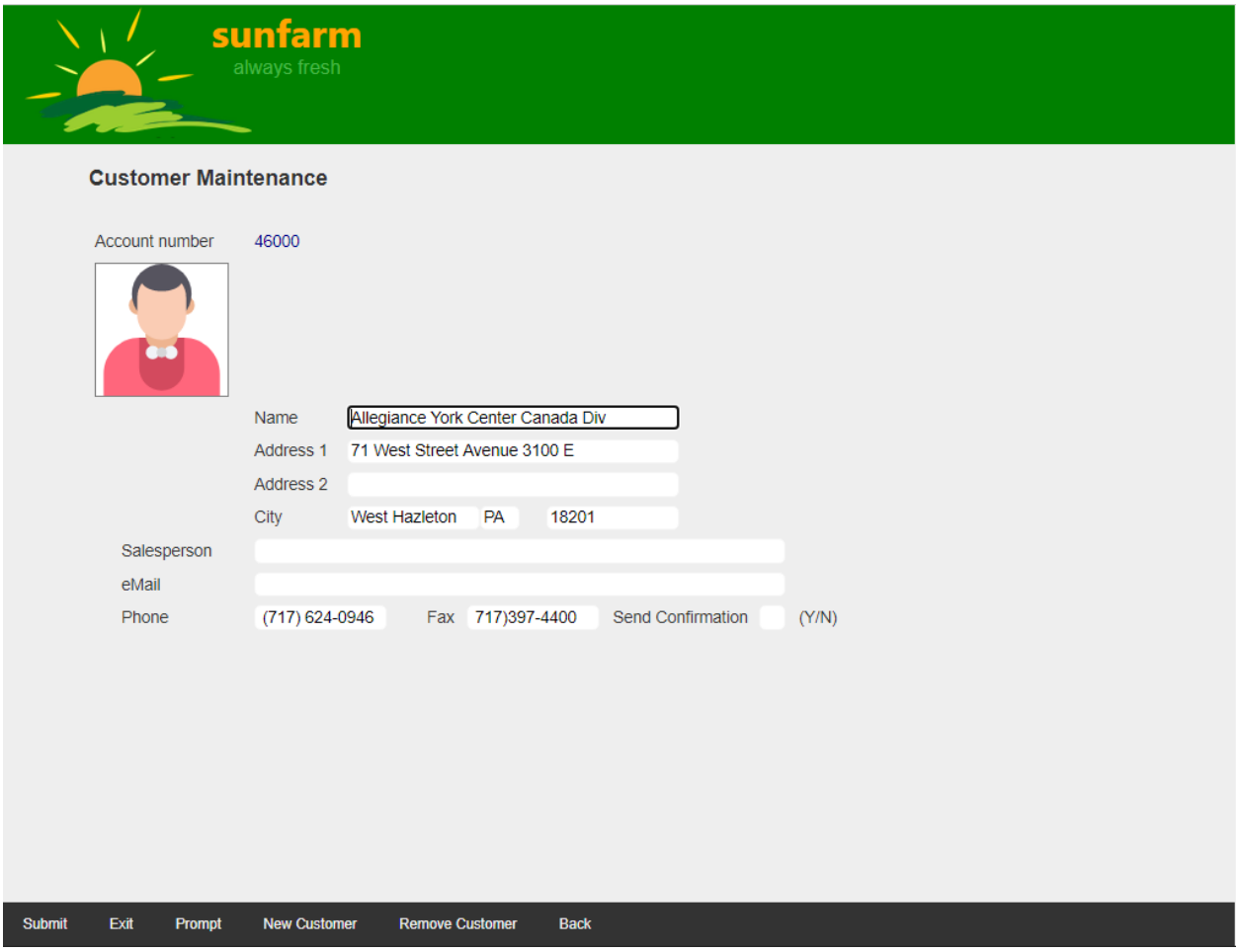
```
<div Row="3">
    <img id="customer-icon" ExpoCol="8" src="~/customer-icon.svg" />
</div>
```

We have our image in the proper vertical position, but it has the wrong horizontal position, width and height.



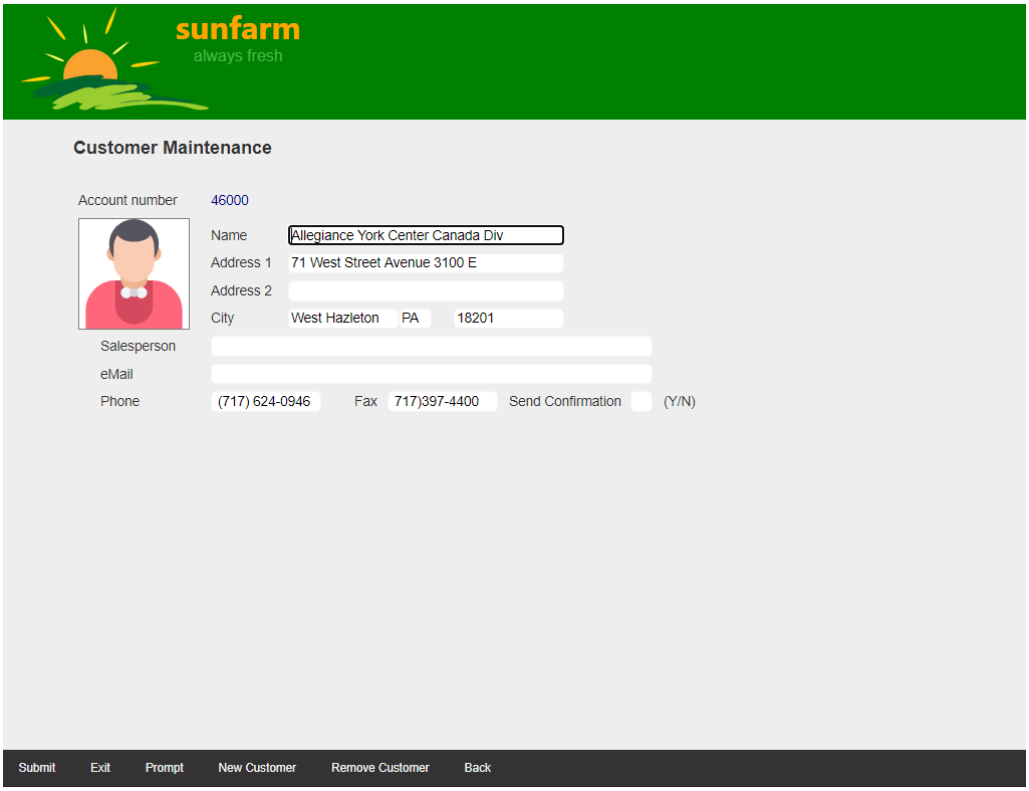
We need to complete the position and dimensions of the image by adding the following CSS style to our site.css (in wwwroot/css folder as before):

```
#customer-icon {
    position: relative;
    width: 109px;
    border-color: gray;
    border-width: thin;
    border-style: solid;
    background-color: white;
}
```



Notice how Row="3" became as tall as the height of the image, pushing all the rest of the rows down. This may or not be what we intended.

If we prefer to not push the rest of rows down, we can change the position style from: “relative” to: “absolute”.
[15]

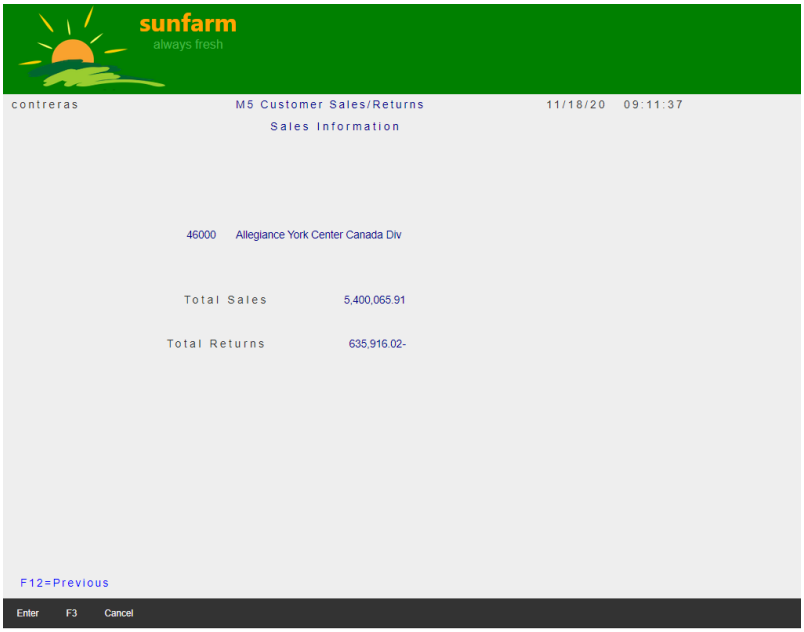


[15] Commit: “Adding an Image to the Page”

Merging two Green-screen pages into one: More intense business logic changes required

Oftentimes after reorganizing green-screen elements on a Web page - particularly on a desktop Browser or large tablet - we end up with lots of space where more information can be displayed.

The SunFarm Customer application as migrated has a menu option to display a page called “Display Sales" which shows the following:



This page shows only two new data items which can easily fit on the “Customer Maintenance” page we have been enhancing. Furthermore, the database contains much more useful information about the Sales that we could display.

If we look at database records (logical file CSMASTERL1), we can see that for each customer we keep monthly sales and returns information on a given year (sales records have the CSTYPE code ‘1’ and returns the code ‘2’).

Displaying the total sales (all recorded years) is too limited.

```sql
SELECT TOP (1000) [CSCUSTNO]
      ,[CSYEAR]
      ,[CSTYPE]
      ,[CSSALES01]
      ,[CSSALES02]
      ,[CSSALES03]
      ,[CSSALES04]
      ,[CSSALES05]
      ,[CSSALES06]
      ,[CSSALES07]
      ,[CSSALES08]
      ,[CSSALES09]
      ,[CSSALES10]
      ,[CSSALES11]
      ,[CSSALES12]
  FROM [ERNUBO].[dbo].[CSMASTERL1#CSMASTERL1]
```

100 %

Results | Messages

	CSCUSTNO	CSYEAR	CSTYPE	CSSALES01	CSSALES02	CSSALES03	CSSALES04	CSSALES05	CSSALES06	CSSALES07	CSSALES08	CSSALES09	CSSALES10	CSSALES11
1	100	1997	1	32742.97	88260.26	63079.75	67713.86	40099.80	35812.20	32191.14	77286.72	37776.28	104244.64	64120.24
2	100	1997	2	-4412.35	-12173.32	-8595.50	-9476.52	-5619.60	-5101.81	-4331.88	-10981.44	-5018.16	-14537.28	-8425.29
3	100	1996	1	46378.81	116899.34	84325.33	92091.74	56702.58	51753.54	45680.70	105830.40	52283.44	138505.96	83565.94
4	100	1996	2	-4165.63	-10181.08	-7463.06	-7867.08	-4835.16	-4318.09	-4110.60	-8860.80	-4786.08	-11887.92	-7761.09
5	100	1995	1	27909.37	80378.10	56700.67	60925.62	35052.08	31052.80	27370.26	69705.60	32531.40	95226.16	57584.28
6	100	1995	2	-4215.55	-12615.40	-8722.94	-9795.24	-5648.16	-5141.41	-4124.52	-11491.20	-4798.80	-15220.32	-8342.97
7	100	1994	1	53612.25	128919.74	93986.17	102581.50	64591.10	59315.58	52880.22	117609.12	59970.56	152315.64	93044.46
8	100	1994	2	-4298.11	-9871.48	-7371.14	-7631.40	-4792.20	-4259.77	-4251.24	-8490.24	-4945.92	-11411.28	-7840.53
9	100	1993	1	23414.97	72835.14	50660.79	54476.58	30343.56	26632.60	22888.58	62463.68	27625.72	86546.88	51387.52
10	100	1993	2	-3961.15	-12999.88	-8792.78	-10056.36	-5619.12	-5123.41	-3859.56	-11943.36	-4521.84	-15845.76	-8203.05
11	200	1997	1	88260.26	63079.75	67713.86	40099.80	35812.20	32191.14	77286.72	37776.28	104244.64	64120.24	120754.14
12	200	1997	2	-12173.32	-8595.50	-9476.52	-5619.60	-5101.81	-4331.88	-10981.44	-5018.16	-14537.28	-8425.29	-16899.48
13	200	1996	1	116899.34	84325.33	92091.74	56702.58	51753.54	45680.70	105830.40	52283.44	138505.96	83565.94	160167.06
14	200	1996	2	-10181.08	-7463.06	-7867.08	-4835.16	-4318.09	-4110.60	-8860.80	-4786.08	-11887.92	-7761.09	-13682.52
15	200	1995	1	80378.10	56700.67	60925.62	35052.08	31052.80	27370.26	69705.60	32531.40	95226.16	57584.28	110624.78
16	200	1995	2	-12615.40	-8722.94	-9795.24	-5648.16	-5141.41	-4124.52	-11491.20	-4798.80	-15220.32	-8342.97	-17785.56
17	200	1994	1	128919.74	93986.17	102581.50	64591.10	59315.58	52880.22	117609.12	59970.56	152315.64	93044.46	175668.50
18	200	1994	2	-9871.48	-7371.14	-7631.40	-4792.20	-4259.77	-4251.24	-8490.24	-4945.92	-11411.28	-7840.53	-13068.60
19	200	1993	1	72835.14	50660.79	54476.58	30343.56	26632.60	22888.58	62463.68	27625.72	86546.88	51387.52	100834.62
20	200	1993	2	-12999.88	-8792.78	-10056.36	-5619.12	-5123.41	-3859.56	-11943.36	-4521.84	-15845.76	-8203.05	-18614.04
21	200	1992	1	141448.94	104155.81	113580.06	72988.42	67386.42	60588.54	129896.64	68166.48	166634.12	103031.78	191678.74
22	200	1992	2	-9523.48	-7240.82	-7357.32	-4710.84	-4163.05	-4353.48	-8081.28	-5067.36	-10896.24	-7881.57	-12416.28
23	300	1997	1	63079.75	67713.86	40099.80	35812.20	32191.14	77286.72	37776.28	104244.64	64120.24	120754.14	37078.80

Depending on the needs of SunFarm company we could argue that it would be more beneficial to show:

1. Detailed sales and return number for the last registered year.
2. Totals for that period
3. Sales trend (last month vs first month of that year)
4. A chart showing how sales progressed throughout the year.

Business Logic for Customer Maintenance Page

CUSTDSPF Displayfile is used by program CUSTINQ.

If you look at CUSTINQ.cs source file $\SunFarm\CustomerAppLogic\CUSTINQ.cs:

```
namespace SunFarm.Customers
{
    [ASNA.QSys.HostServices.ActivationGroup("*DFTACTGRP")]
    [ProgramEntry("_ENTRY")]
    public partial class Custinq : ASNA.QSys.HostServices.Program
    {
.
.
.
        WorkstationFile CUSTDSPF;
        DatabaseFile CUSTOMERL2;
        DatabaseFile CUSTOMERL1;
```

You can see that originally this program:

1. Uses Workstation file: CUSTDSPF
2. Uses one physical file CUSTOMER (thru two logical files: CUSTOMERL1 and CUSTOMERL2). Logical files are by Customer Number or by Customer Name (then by Customer Number)

There is no CSMASTER (Sales and Returns file).

If we are to write sales-returns related fields on the “CUSTREC” record on the Displayfile, we need to use the CSMASTER file (or one of its logical files)

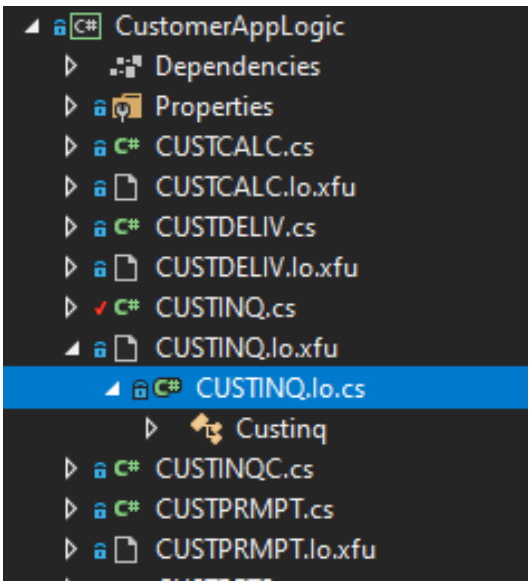
Let’s add the following database file declaration on Program Custinq:

```
namespace SunFarm.Customers
{
    [ASNA.QSys.HostServices.ActivationGroup("*DFTACTGRP")]
    [ProgramEntry("_ENTRY")]
    public partial class Custinq : ASNA.QSys.HostServices.Program
    {
.
.
.
        WorkstationFile CUSTDSPF;
        DatabaseFile CUSTOMERL2;
        DatabaseFile CUSTOMERL1;
        DatabaseFile CSMASTERL1;    // Sales and Returns file
```

Keep in mind that we are not using RPG (or Visual RPG) language in this migration. RPG languages would declare all fields in the database file automatically for the developer to use and populate. This does not happen when using any other language such as C#

ASNA QSys Runtime framework provides a source file for every migrated Program to deal with the task of declaring fields as variables and populating the values in-out those fields. The naming convention is to use the same name as the Program, with the extension ***.Io.cs***

The IO source completes the class (using partial class declaration). The corresponding IO partial class file shows in the Solution Explorer Project tree under the Program C# file as shown:



Declaring CSMASTERL1 is not enough, we need to:

1. Initialize @ _instanceInit() - where we connect with methods to populate in-out.
2. Implement populate in-out methods in the IO partial class.
3. Set its Job's overrider class.
4. Open and Close the new file.

After we initialize the database file, the ASNA QSys Runtime framework will call the populate in-out methods as appropriate when reading and writing records from/to the database file.

Initialize the new Database File

Since we are merging two green-screens into one Web Page, we can take advantage of existing code. CUSTCALC program formerly used to compute the Total Sales and Returns already has this code.

If we follow the Custcalc() constructor we can see how it will call **_instanceInit** and then perform two more method calls related to this new file (including opening it).

While we are at it, we can see how the **Dispose** method will take care of closing the file.

```
#region Constructor and Dispose
        public Custcalc()
        {
            _IN = new IndicatorArray<Len<_1, _0, _0>>((char[])null);
            _instanceInit();
            CSMASTERL1.Overrider = Job;
            CSMASTERL1.Open(Job.Database, AccessMode.Read, false, false, ServerCursors.Default);
            CUSTOMERL1.Overrider = Job;
            CUSTOMERL1.Open(Job.Database, AccessMode.Read, false, false, ServerCursors.Default);
        }

        override public void Dispose(bool disposing)
        {
            if (disposing)
            {

                CUSTOMERL1.Close();
                CSMASTERL1.Close();
            }
            base.Dispose(disposing);
        }
#endregion
```

We will copy the lines in blue above to the CUSTINQ.cs constructor and destructor:

```
#region Constructor and Dispose
        public Custinq()
        {
            _IN = new IndicatorArray<Len<_1, _0, _0>>((char[])null);
            _instanceInit();
            .
            .
            .

            CSMASTERL1.Overrider = Job;
            CSMASTERL1.Open(Job.Database, AccessMode.Read, false, false, ServerCursors.Default);
        }

        override public void Dispose(bool disposing)
        {
            if (disposing)
            {
                .
                .
                .

                CSMASTERL1.Close();
            }
            base.Dispose(disposing);
        }
#endregion
```

Then we will instantiate the CSMASTERL1 variable inside the _instanceInit method (again, by copying two lines from the same method in CUSTCALC):

```
void _instanceInit()
{
    .
    .
    .
    CUSTOMERL2 = new DatabaseFile(PopulateBufferCUSTOMERL2, PopulateFieldsCUSTOMERL2, null,
"CUSTOMERL2", "*LIBL/CUSTOMERL2", CUSTOMERL2FormatIDs)
    { IsDefaultRFN = true };
    CUSTOMERL1 = new DatabaseFile(PopulateBufferCUSTOMERL1, PopulateFieldsCUSTOMERL1, null,
"CUSTOMERL1", "*LIBL/CUSTOMERL1", CUSTOMERL1FormatIDs, blockingFactor : 0)
    { IsDefaultRFN = true };

    CSMASTERL1 = new DatabaseFile(PopulateBufferCSMASTERL1, PopulateFieldsCSMASTERL1, null,
"CSMASTERL1", "*LIBL/CSMASTERL1", CSMASTERL1FormatIDs)
    { IsDefaultRFN = true };

    CUSTDS = new DataStructure(CUSTDS_000, CUSTDS_001, CUSTDS_002, CUSTDS_003, CUSTDS_004,
    .
    .
    .
}
```

Notice how the PopulateBufferCSMASTERL1 and PopulateFieldsCSMASTERL1 methods appear undefined, because they are. We will complete next.

Implement populate in-out methods in the IO class

Since we are merging two green-screen pages into one Web page, we can take advantage on existing code. Locate the implementation of methods PopulateBufferCSMASTERL1 and PopulateFieldsCSMASTERL1 in CUSTCALC.Io.cs and add them to the CUSTINQ.Io.cs file (at the end if the source file):

$\SunFarm\CustomerAppLogic\CUSTINQ.Io.cs:

```
        private void PopulateBufferCSMASTERL1(string _, AdgDataSet _dataSet)
        {
            var _table = _dataSet.GetAdgTable("*FILE");
            System.Data.DataRow _row = _table.Row;
            _row["CSCUSTNO"] = ((decimal)(CSCUSTNO));
            _row["CSYEAR"] = ((decimal)(CSYEAR));
            _row["CSTYPE"] = ((decimal)(CSTYPE));
            _row["CSSALES01"] = ((decimal)(CSSALES01));
            _row["CSSALES02"] = ((decimal)(CSSALES02));
            _row["CSSALES03"] = ((decimal)(CSSALES03));
            _row["CSSALES04"] = ((decimal)(CSSALES04));
            _row["CSSALES05"] = ((decimal)(CSSALES05));
            _row["CSSALES06"] = ((decimal)(CSSALES06));
            _row["CSSALES07"] = ((decimal)(CSSALES07));
            _row["CSSALES08"] = ((decimal)(CSSALES08));
            _row["CSSALES09"] = ((decimal)(CSSALES09));
            _row["CSSALES10"] = ((decimal)(CSSALES10));
            _row["CSSALES11"] = ((decimal)(CSSALES11));
            _row["CSSALES12"] = ((decimal)(CSSALES12));
        }
        private void PopulateFieldsCSMASTERL1(string _, AdgDataSet _dataSet)
        {
            var _table = _dataSet.SetActive("*FILE");
            System.Data.DataRow _row = _table.Row;
            CSCUSTNO = ((decimal)(_row["CSCUSTNO"]));
            CSYEAR = ((decimal)(_row["CSYEAR"]));
            CSTYPE = ((decimal)(_row["CSTYPE"]));
            CSSALES01 = ((decimal)(_row["CSSALES01"]));
            CSSALES02 = ((decimal)(_row["CSSALES02"]));
            CSSALES03 = ((decimal)(_row["CSSALES03"]));
            CSSALES04 = ((decimal)(_row["CSSALES04"]));
            CSSALES05 = ((decimal)(_row["CSSALES05"]));
            CSSALES06 = ((decimal)(_row["CSSALES06"]));
            CSSALES07 = ((decimal)(_row["CSSALES07"]));
            CSSALES08 = ((decimal)(_row["CSSALES08"]));
            CSSALES09 = ((decimal)(_row["CSSALES09"]));
            CSSALES10 = ((decimal)(_row["CSSALES10"]));
            CSSALES11 = ((decimal)(_row["CSSALES11"]));
            CSSALES12 = ((decimal)(_row["CSSALES12"]));
        }
    }
}
```

Notice how one method populates the DataSet associated with the CSMASTERL1 database file (from fields in the class), while the other does the opposite: populate the fields in the class from the associated with the CSMASTERL1 DataSet.

Now the class variables corresponding to the fields in the file record are undefined (as indicated by Visual Studio smart editor).

If you are following closely, you would have correctly guessed that the next step is to copy the declaration of the missing fields from CUSTCALC.Io.cs to CUSTINQ.Io.cs. The easiest way to do it, is to locate one of the fields in CUSTCALC.Io.cs, let's say in PopulateBufferCSMASTERL1 CSCUSTNO and press F12 to jump to the definition, collapse the getter/setter implementation, copy the range of lines (lines 63-217) as shown below:

```
CUSTCALC.Io.cs   CUSTINQ.Io.cs   Output
CustomerAppLogic                 SunFarm.Customers.Custcalc
53    static ILayout CUSTSL_014 = Layout.Packed(11
54    private static Dictionary<string, string> CU
55    {
56        { "RCUSTOMER", "6su4S42+ard0ZHitdjHOFT1W
57    };
58    private static Dictionary<string, string> CS
59    {
60        { "RCSMASTL1", "MNWSCMjX4uCVz4ny/YR2N6c1
61    };
62
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
63    private FixedDecimal<_9, _0> CSCUSTNO...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
74    private FixedDecimal<_4, _0> CSYEAR...
      4 references | contreras-jorgev, 12 days ago | 1 author, 1 change
85    private FixedDecimal<_1, _0> CSTYPE...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
96    private FixedDecimal<_11, _2> CSSALES01...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
107   private FixedDecimal<_11, _2> CSSALES02...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
118   private FixedDecimal<_11, _2> CSSALES03...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
129   private FixedDecimal<_11, _2> CSSALES04...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
140   private FixedDecimal<_11, _2> CSSALES05...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
151   private FixedDecimal<_11, _2> CSSALES06...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
162   private FixedDecimal<_11, _2> CSSALES07...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
173   private FixedDecimal<_11, _2> CSSALES08...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
184   private FixedDecimal<_11, _2> CSSALES09...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
195   private FixedDecimal<_11, _2> CSSALES10...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
206   private FixedDecimal<_11, _2> CSSALES11...
      2 references | contreras-jorgev, 12 days ago | 1 author, 1 change
217   private FixedDecimal<_11, _2> CSSALES12...
      1 reference | contreras-jorgev, 12 days ago | 1 author, 1 change
```

CUSTINQ.Io.cs is almost complete.

If you look at the declaration of any of the fields we just copied from CUSTCALC.Io.cs to CUSTINQ.Io.cs you will see that they all have a getter and setter that refers to a data-structure. We do not need that complexity. The new code we will write to prepare the fields we will write to the Displayfile will be done with modern C# code.

Simplify the declaration of the fields in CUSTINQ.Io.cs to the following (getter and setter removed):

```
private FixedDecimal<_9, _0> CSCUSTNO;
private FixedDecimal<_4, _0> CSYEAR;
private FixedDecimal<_1, _0> CSTYPE;
private FixedDecimal<_11, _2> CSSALES01;
private FixedDecimal<_11, _2> CSSALES02;
private FixedDecimal<_11, _2> CSSALES03;
private FixedDecimal<_11, _2> CSSALES04;
private FixedDecimal<_11, _2> CSSALES05;
private FixedDecimal<_11, _2> CSSALES06;
private FixedDecimal<_11, _2> CSSALES07;
private FixedDecimal<_11, _2> CSSALES08;
private FixedDecimal<_11, _2> CSSALES09;
private FixedDecimal<_11, _2> CSSALES10;
private FixedDecimal<_11, _2> CSSALES11;
private FixedDecimal<_11, _2> CSSALES12;
```

Set Job's overrider class

This code is already in the constructor lines we copied before.

Open and Close the new file

This code is already in the constructor and Dispose lines we copied before.

The new file needs to know its Format Identifier

If you tried to compile the changes to the Business Logic so far, you will hit one error. The _initInstance is missing the definition of the CSMASERL1 record format: "RCSMASTL1" ID string.

Copy the line from CUSTCALC.Io.cs to the CUSTINQ.Io.cs

```
private static Dictionary<string, string> CSMASTERL1FormatIDs = new
Dictionary<string, string>()
{
    { "RCSMASTL1", "MNWSCMjX4uCVz4ny/YR2N6c1XTM=" }
};
```

The CustomerAppLogic should now build without errors, and the Application should run. (Refer to the History of the reference source in GitHub for complete changes[16])

[16] Commit: "Declaration of Sales Returns file in CUSTINQ.cs"

Displaying new data on Customer Maintenance Page

We know that the Program CUSTINQ is now able to deal with Sales and Return information for a customer (previously only in CUSTCALC).

Let's focus now in the Front-End. Open the CUSTDSPF.cshtml markup file and locate DdsRecord For="CUSTREC" tag-helper.

We have used Rows 2 thru 10, let's now add more information on Rows 12 thru 20

Last registered sales	**(Year 1997)**					**960,992.57**	**↓ +21.3%**
Jan	121,174.32	Feb	50,489.49	Mar	100,567.71	Apr	96,354.00
May	99,382.42	Jun	116,623.63	Jul	54,574.63	Aug	55,472.98
Sep	116,586.43	Oct	61,889.79	Nov	62,112.07	Dec	25,765.10
Last registered returns	**(Year 1997)**					**131,644.38**	
Jan	16,950.64	Feb	6,806.59	Mar	13,552.22	Apr	13,089.60
May	13,158.84	Jun	16,079.07	Jul	7,643.46	Aug	7,341.96
Sep	16,092.86	Oct	8,439.19	Nov	8,820.75	Dec	3,669.20

We have several new constants, and around thirty new fields. Let's concentrate on the new fields we want to display. Fields are referred to by the tag-helpers using the attribute **For=**

First let's start by displaying Sales information. Rows 12 thru 15:

```
<div Row="12">
    <DdsConstant Col="8" Text="Last registered sales" />
</div>
<div Row="13">
    <DdsConstant Col="12" Text="Jan" />
    <DdsDecField Col="15" For="CUSTREC.CSSALES01" EditCode=One />
    <DdsConstant Col="30" Text="Feb" />
    <DdsDecField Col="35" For="CUSTREC.CSSALES02" EditCode=One />
    <DdsConstant Col="48" Text="Mar" />
    <DdsDecField Col="51" For="CUSTREC.CSSALES03" EditCode=One />
    <DdsConstant Col="66" Text="Apr" />
    <DdsDecField Col="69" For="CUSTREC.CSSALES04" EditCode=One />
</div>
<div Row="14">
    <DdsConstant Col="12" Text="May" />
    <DdsDecField Col="15" For="CUSTREC.CSSALES05" EditCode=One />
    <DdsConstant Col="30" Text="Jun" />
    <DdsDecField Col="35" For="CUSTREC.CSSALES06" EditCode=One />
    <DdsConstant Col="48" Text="Jul" />
    <DdsDecField Col="51" For="CUSTREC.CSSALES07" EditCode=One />
    <DdsConstant Col="66" Text="Aug" />
    <DdsDecField Col="69" For="CUSTREC.CSSALES08" EditCode=One />
</div>
<div Row="15">
    <DdsConstant Col="12" Text="Sep" />
    <DdsDecField Col="15" For="CUSTREC.CSSALES09" EditCode=One />
    <DdsConstant Col="30" Text="Oct" />
    <DdsDecField Col="35" For="CUSTREC.CSSALES10" EditCode=One />
    <DdsConstant Col="48" Text="Nov" />
    <DdsDecField Col="51" For="CUSTREC.CSSALES11" EditCode=One />
```

```
    <DdsConstant Col="66" Text="Dec" />
    <DdsDecField Col="69" For="CUSTREC.CSSALES12" EditCode=One />
</div>
```

If we ignore the display attributes and concentrate on the new data-fields we have thirty new fields:

```
For="CUSTREC.CSSALES01"
For="CUSTREC.CSSALES02"
For="CUSTREC.CSSALES03"
For="CUSTREC.CSSALES04"
For="CUSTREC.CSSALES05"
For="CUSTREC.CSSALES06"
For="CUSTREC.CSSALES07"
For="CUSTREC.CSSALES08"
For="CUSTREC.CSSALES09"
For="CUSTREC.CSSALES10"
For="CUSTREC.CSSALES11"
For="CUSTREC.CSSALES12"
```

You may have noticed that Visual Studio Razor Page smart editor is highlighting names with the reference to known CUSTREC Model property as undefined.

Let's declare the new fields in the Model ($\SunFarm\CustomerAppSite\Areas\CustomerAppViews\Pages\CUSTDSPF.cshtml.cs):

The last field in the Model CUSTDSP.CUSTREC_Model was:

```
            [Char(1)]
            public string SFYN01 { get; set; }
```

Let's add the rest (after SFYN01):

```
[Dec(11, 2)]
public decimal CSSALES01 { get; private set; }

[Dec(11, 2)]
public decimal CSSALES02 { get; private set; }

[Dec(11, 2)]
public decimal CSSALES03 { get; private set; }

[Dec(11, 2)]
public decimal CSSALES04 { get; private set; }

[Dec(11, 2)]
public decimal CSSALES05 { get; private set; }

[Dec(11, 2)]
public decimal CSSALES06 { get; private set; }

[Dec(11, 2)]
public decimal CSSALES07 { get; private set; }

[Dec(11, 2)]
public decimal CSSALES08 { get; private set; }

[Dec(11, 2)]
public decimal CSSALES09 { get; private set; }

[Dec(11, 2)]
public decimal CSSALES10 { get; private set; }
```

```
[Dec(11, 2)]
public decimal CSSALES11 { get; private set; }

[Dec(11, 2)]
public decimal CSSALES12 { get; private set; }
```

The Markup will find the proper references to the Model file. How do we know the length and type for the fields? For now we can find the IO field definition in file

$\SunFarm\CustomerAppLogic\CUSTCALC.Io.cs

```
private FixedDecimal<_11, _2> CSSALES01;
private FixedDecimal<_11, _2> CSSALES02;
.
.
.
private FixedDecimal<_11, _2> CSSALES12;
```

(We could also look into the Database definition)

Defining the fields in the Workstation DataSet

In the Business Logic, when the user selects option 2 (Update), we have code in CUSTINQ.cs to locate the subfile record selected and call the **RcdUpdate** method:

$\SunFarm\CustomerAppLogic\CUSTINQ.cs:

```
else if (SFSEL == 2)
{
    // Maintainance.
    CUSTDSPF.ChainByRRN("SFL1", (int)sflrrn, _IN.Array);
    RcdUpdate();
}
```

The Record Update method will loop around as long as *IN12 is off and on each turn, Write the Subfile Controller MSGSFC and then Execute format CUSTREC on the Workstation file:

$\SunFarm\CustomerAppLogic\CUSTINQ.cs:

```
void RcdUpdate()
{
    Indicator _LR = '0';
    ClearSel();
    AddUpd = "U";
    CMCUSTNO = (decimal)SFCUSTNO;

    .
    .
    .
    do
    {
        CUSTDSPF.Write("MSGSFC", _IN.Array);
        CUSTDSPF.ExFmt("CUSTREC", _IN.Array);

        // React to input read from record CUSTREC (including indicator state)
        .
        .
        .
```

```
        }
    } while (!((bool)_IN[12])); // Repeat
}
```

Before Executing the Format “CUSTREC”, which first Writes the record from the Business Logic’s DataSet to the Display file (then sent to the Browser for input), we need to Load the Last Sales and Returns data from the CSMASTERL1 file.

The method to Load Sales and Return Data will be called LoadLastSalesAndReturns and starts by seeking the first record that matches the Customer Number and reads that record (we’ll come back to this method to complete it later):

```
private void LoadLastSalesAndReturns()
{
    FixedDecimal<_9, _0> CustomerNumber = new FixedDecimal<_9, _0>();

    CustomerNumber = CMCUSTNO.MoveRight(CustomerNumber);
    if ( CSMASTERL1.Seek(SeekMode.SetLL, CustomerNumber) )
        CSMASTERL1.ReadNextEqual(false, CustomerNumber);
}
```

Let’s add a reference to this method from within the RcdUpdate method (line in blue below):

```
void RcdUpdate()
{
    Indicator _LR = '0';
    ClearSel();
    AddUpd = "U";
    CMCUSTNO = (decimal)SFCUSTNO;

    .
    .
    .
    do
    {

        LoadLastSalesAndReturns();

        CUSTDSPF.Write("MSGSFC", _IN.Array);
        CUSTDSPF.ExFmt("CUSTREC", _IN.Array);

        // React to input read from record CUSTREC (including indicator state)
        .
        .
        .

        }
    } while (!((bool)_IN[12])); // Repeat
}
```

By the time the Execute Format executes, the sales information (variables CSSALES01 thru CSSALES12) will have the values read from the logical file, but we still need to add code to populate the Workstation DataSet for record (table CUSTREC).

CUSTDSPF.Write operation (implicit in CUSTDSPF.ExFmt) will call the IO method PopulateBufferCUSTDSPFCUSTREC. We need to add code to complete the Sales information from the class variables to the table in the DataSet (code in blue below):

```
private void PopulateBufferCUSTDSPFCUSTREC(AdgDataSet _dataSet)
{
    var _table = _dataSet.GetAdgTable("CUSTREC");
```

```
    System.Data.DataRow _row = _table.Row;
    _row["CSRREC"] = ((string)(CSRREC));
.
.
.
    _row["SFYN01"] = ((string)(SFYN01));

    // Important: match the CUSTREC_Model field declaration field names and order.
    _row["CSSALES01"] = ((decimal)(CSSALES01));
    _row["CSSALES02"] = ((decimal)(CSSALES02));
    _row["CSSALES03"] = ((decimal)(CSSALES03));
    _row["CSSALES04"] = ((decimal)(CSSALES04));
    _row["CSSALES05"] = ((decimal)(CSSALES05));
    _row["CSSALES06"] = ((decimal)(CSSALES06));
    _row["CSSALES07"] = ((decimal)(CSSALES07));
    _row["CSSALES08"] = ((decimal)(CSSALES08));
    _row["CSSALES09"] = ((decimal)(CSSALES09));
    _row["CSSALES10"] = ((decimal)(CSSALES10));
    _row["CSSALES11"] = ((decimal)(CSSALES11));
    _row["CSSALES12"] = ((decimal)(CSSALES12));
}
```

In preparation to complete the IO code for when the Workstation file is read (with the response coming from the Browser), let's add similar lines of code to the PopulateFieldsCUSTDSPFCUSTREC method, to populate Sales class variables from the DataSet table, with the lines in blue below:[17]

```
private void PopulateFieldsCUSTDSPFCUSTREC(AdgDataSet _dataSet)
{
    var _table = _dataSet.GetAdgTable("CUSTREC");
    System.Data.DataRow _row = _table.Row;
.
.
.
    SFYN01 = ((string)(_row["SFYN01"]));

    // Important: match the CUSTREC_Model field declaration field names and order.
    CSSALES01 = ((decimal)(_row["CSSALES01"]));
    CSSALES02 = ((decimal)(_row["CSSALES02"]));
    CSSALES03 = ((decimal)(_row["CSSALES03"]));
    CSSALES04 = ((decimal)(_row["CSSALES04"]));
    CSSALES05 = ((decimal)(_row["CSSALES05"]));
    CSSALES06 = ((decimal)(_row["CSSALES06"]));
    CSSALES07 = ((decimal)(_row["CSSALES07"]));
    CSSALES08 = ((decimal)(_row["CSSALES08"]));
    CSSALES09 = ((decimal)(_row["CSSALES09"]));
    CSSALES10 = ((decimal)(_row["CSSALES10"]));
    CSSALES11 = ((decimal)(_row["CSSALES11"]));
    CSSALES12 = ((decimal)(_row["CSSALES12"]));
}
```

[17] Commit: "First Sales information added to Page"

Compile the Business Logic Project and run the Website. Selecting any of the records of the Customer list and opting to “Update”, will present the Customer Maintenance Page with the first year of Sales information (note: the label says “Last registered sales”, we will fix that later).