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’).

```
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

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

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

1. Detailed sales and returns 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)

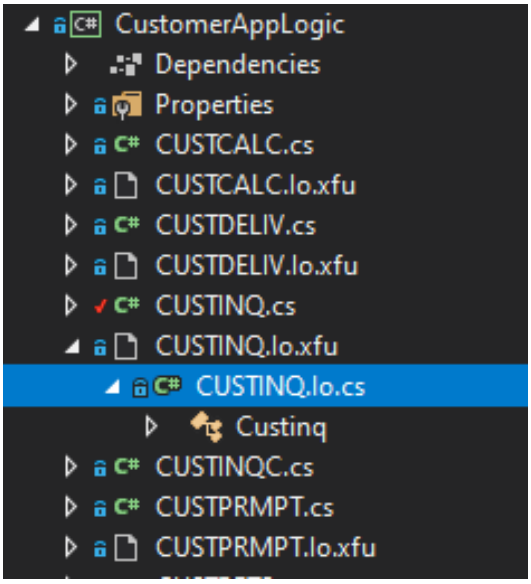
Let’s add the following database file declaration on Program Custinq:

```
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):

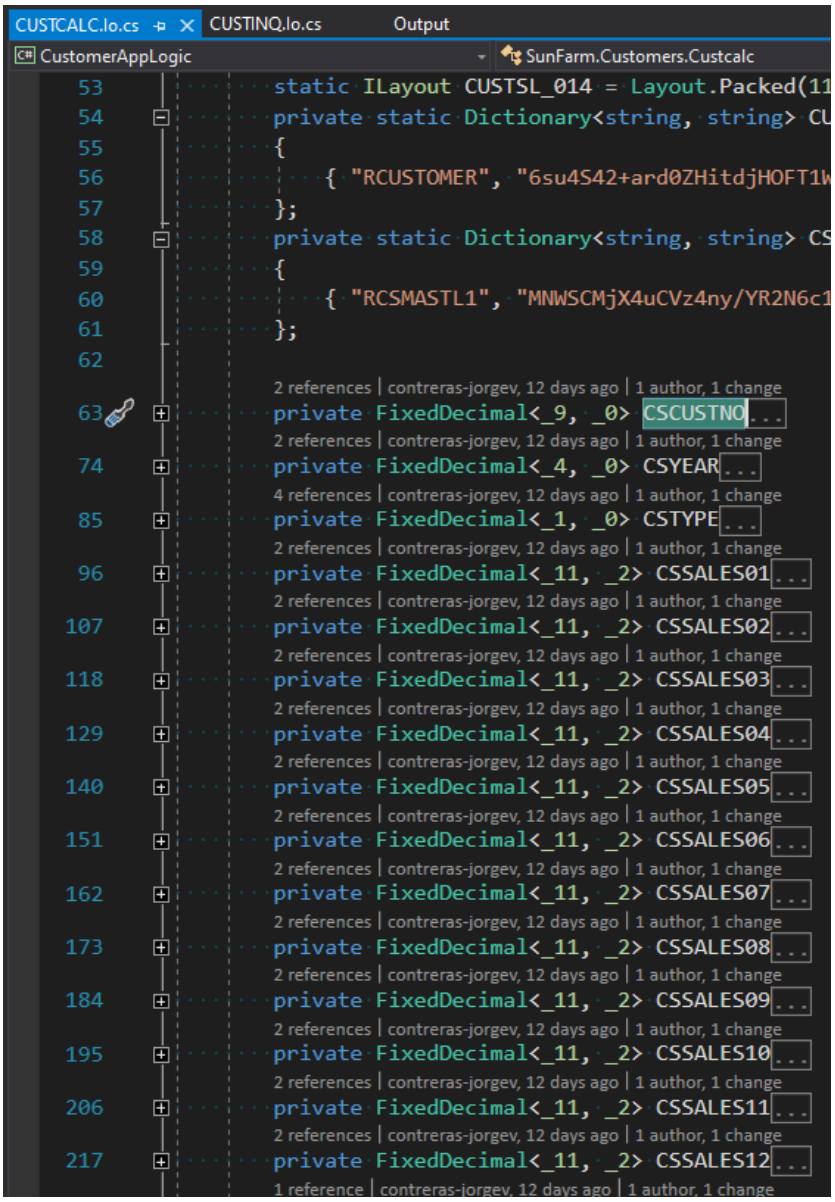
$\SunFarm\CustomerAppLogic\CUSTINQ.Io.cs:

```
        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:



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[1])

[1] 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):

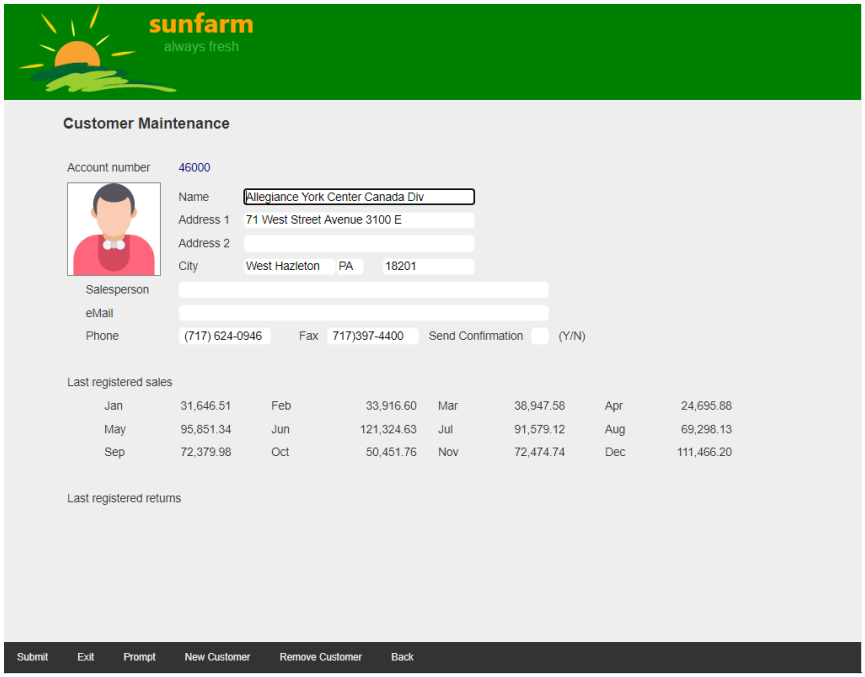
```
private void PopulateBufferCUSTDSPFCUSTREC(AdgDataSet _dataSet)
{
    var _table = _dataSet.GetAdgTable("CUSTREC");
    System.Data.DataRow _row = _table.Row;
    _row["CSRREC"] = ((string)(CSRREC));
.
.
.
    _row["SFYN01"] = ((string)(SFYN01));

    // 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:[2]

```
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"]));
}
```

[2] 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).



Add Returns Data to the Page

Similarly to how we added Sales data to the Page, we will add Returns data.

Back to the Business Logic, let’s declare one year’s worth of Returns data.

Locate the declaration of CSSALES01 thru CSSALES12 in CUSTINQ.Io.cs, add the declarations for Returns (same type, length and precision):

```
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;
```

```
private FixedDecimal<_11, _2> CSRETURN01;
private FixedDecimal<_11, _2> CSRETURN02;
private FixedDecimal<_11, _2> CSRETURN03;
private FixedDecimal<_11, _2> CSRETURN04;
private FixedDecimal<_11, _2> CSRETURN05;
private FixedDecimal<_11, _2> CSRETURN06;
private FixedDecimal<_11, _2> CSRETURN07;
private FixedDecimal<_11, _2> CSRETURN08;
private FixedDecimal<_11, _2> CSRETURN09;
private FixedDecimal<_11, _2> CSRETURN10;
private FixedDecimal<_11, _2> CSRETURN11;
private FixedDecimal<_11, _2> CSRETURN12;
```

CSRETURNnn is not a field in the database, but we want to use those fields in the CUSTREC_Model (and refer to those values in the Markup), such that we can present it in the Page.

For reference, the “Sales and Returns” database file definition is:

```
CREATE TABLE [dbo].[CSMASTER](
	[CSCUSTNO] [decimal](9, 0) NOT NULL,
	[CSYEAR] [numeric](4, 0) NOT NULL,
	[CSTYPE] [numeric](1, 0) NOT NULL,
	[CSSALES01] [decimal](11, 2) NOT NULL,
	[CSSALES02] [decimal](11, 2) NOT NULL,
	[CSSALES03] [decimal](11, 2) NOT NULL,
	[CSSALES04] [decimal](11, 2) NOT NULL,
	[CSSALES05] [decimal](11, 2) NOT NULL,
	[CSSALES06] [decimal](11, 2) NOT NULL,
	[CSSALES07] [decimal](11, 2) NOT NULL,
	[CSSALES08] [decimal](11, 2) NOT NULL,
	[CSSALES09] [decimal](11, 2) NOT NULL,
	[CSSALES10] [decimal](11, 2) NOT NULL,
	[CSSALES11] [decimal](11, 2) NOT NULL,
	[CSSALES12] [decimal](11, 2) NOT NULL,
 CONSTRAINT [ASNA_KEY_CSMASTERL1] UNIQUE NONCLUSTERED
(
	[CSCUSTNO] ASC,
	[CSYEAR] ASC,
	[CSTYPE] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY],
 CONSTRAINT [ASNA_KEY_CSMASTERL2] UNIQUE NONCLUSTERED
(
	[CSCUSTNO] ASC,
	[CSYEAR] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Where CSSALES01 may refer to a RETURNS value when the record type CSTYPE is “2”.

There is no need then to add the new class variables CSRETURN01 thru CSRETURN12 to the I/O methods to PopulateBufferCSMASTERL1 and PopulateBufferCSMASTERL1. (We will set their value in code in the new LoadLastSalesAndReturns method.

To complete the I/O we still need to add the new Returns class variables to the Workstation I/O operations such that they appear in the DataSet and may be copied to the corresponding properties in the CUSTREC_Model. That is, update:

1. PopulateBufferCUSTDSPFCUSTREC
2. PopulateFieldsCUSTDSPFCUSTREC

```
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["CSSALES12"] = ((decimal)(CSSALES12));

    _row["CSRETURN01"] = ((decimal)(CSRETURN01));
    _row["CSRETURN02"] = ((decimal)(CSRETURN02));
    _row["CSRETURN03"] = ((decimal)(CSRETURN03));
    _row["CSRETURN04"] = ((decimal)(CSRETURN04));
    _row["CSRETURN05"] = ((decimal)(CSRETURN05));
    _row["CSRETURN06"] = ((decimal)(CSRETURN06));
    _row["CSRETURN07"] = ((decimal)(CSRETURN07));
    _row["CSRETURN08"] = ((decimal)(CSRETURN08));
    _row["CSRETURN09"] = ((decimal)(CSRETURN09));
    _row["CSRETURN10"] = ((decimal)(CSRETURN10));
    _row["CSRETURN11"] = ((decimal)(CSRETURN11));
    _row["CSRETURN12"] = ((decimal)(CSRETURN12));
}
private void PopulateFieldsCUSTDSPFCUSTREC(AdgDataSet _dataSet)
{
    var _table = _dataSet.GetAdgTable("CUSTREC");
    System.Data.DataRow _row = _table.Row;
    CSRREC = ((string)(_row["CSRREC"]));
.
.
.
    SFYN01 = ((string)(_row["SFYN01"]));

    // Important: match the CUSTREC_Model field declaration field names and order.
    CSSALES01 = ((decimal)(_row["CSSALES01"]));
.
.
.
    CSSALES12 = ((decimal)(_row["CSSALES12"]));

    CSRETURN01 = ((decimal)(_row["CSRETURN01"]));
    CSRETURN02 = ((decimal)(_row["CSRETURN02"]));
    CSRETURN03 = ((decimal)(_row["CSRETURN03"]));
    CSRETURN04 = ((decimal)(_row["CSRETURN04"]));
    CSRETURN05 = ((decimal)(_row["CSRETURN05"]));
    CSRETURN06 = ((decimal)(_row["CSRETURN06"]));
    CSRETURN07 = ((decimal)(_row["CSRETURN07"]));
    CSRETURN08 = ((decimal)(_row["CSRETURN08"]));
    CSRETURN09 = ((decimal)(_row["CSRETURN09"]));
    CSRETURN10 = ((decimal)(_row["CSRETURN10"]));
    CSRETURN11 = ((decimal)(_row["CSRETURN11"]));
    CSRETURN12 = ((decimal)(_row["CSRETURN12"]));
}
```

Next, we will add Returns definitions to the CUSTREC_Model:

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

public class CUSTREC_Model : RecordModel
{

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

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

    [Dec(11, 2)]
    public decimal CSRETURN01 { get; private set; }

    [Dec(11, 2)]
    public decimal CSRETURN02 { get; private set; }

    [Dec(11, 2)]
    public decimal CSRETURN03 { get; private set; }

    [Dec(11, 2)]
    public decimal CSRETURN04 { get; private set; }

    [Dec(11, 2)]
    public decimal CSRETURN05 { get; private set; }

    [Dec(11, 2)]
    public decimal CSRETURN06 { get; private set; }

    [Dec(11, 2)]
    public decimal CSRETURN07 { get; private set; }

    [Dec(11, 2)]
    public decimal CSRETURN08 { get; private set; }

    [Dec(11, 2)]
    public decimal CSRETURN09 { get; private set; }

    [Dec(11, 2)]
    public decimal CSRETURN10 { get; private set; }

    [Dec(11, 2)]
    public decimal CSRETURN11 { get; private set; }

    [Dec(11, 2)]
    public decimal CSRETURN12 { get; private set; }
```

Before we build and run the changes so far, let's initialize the year's Returns variables, my adding a few lines to LoadLastSalesAndReturns:

```
$\SunFarm\CustomerAppLogic\CUSTINQ.cs

private void LoadLastSalesAndReturns()
{
    CSSALES01 = CSSALES02 = CSSALES03 = CSSALES04 = CSSALES05 = CSSALES06 = 0;
    CSSALES07 = CSSALES08 = CSSALES09 = CSSALES10 = CSSALES11 = CSSALES12 = 0;

    CSRETURN01 = CSRETURN02 = CSRETURN03 = CSRETURN04 = CSRETURN05 = CSRETURN06 = 0;
    CSRETURN07 = CSRETURN08 = CSRETURN09 = CSRETURN10 = CSRETURN11 = CSRETURN12 = 0;

    FixedDecimal < _9, _0> CustomerNumber = new FixedDecimal<_9, _0>();

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

Build and run. Nothings appears to have changed, we need to add the Return data to the Markup.
[3]

---

[3] Commit: "Adding Returns to the CUSTREC_Model DataSet"

Displaying the current Values of Returns for the year

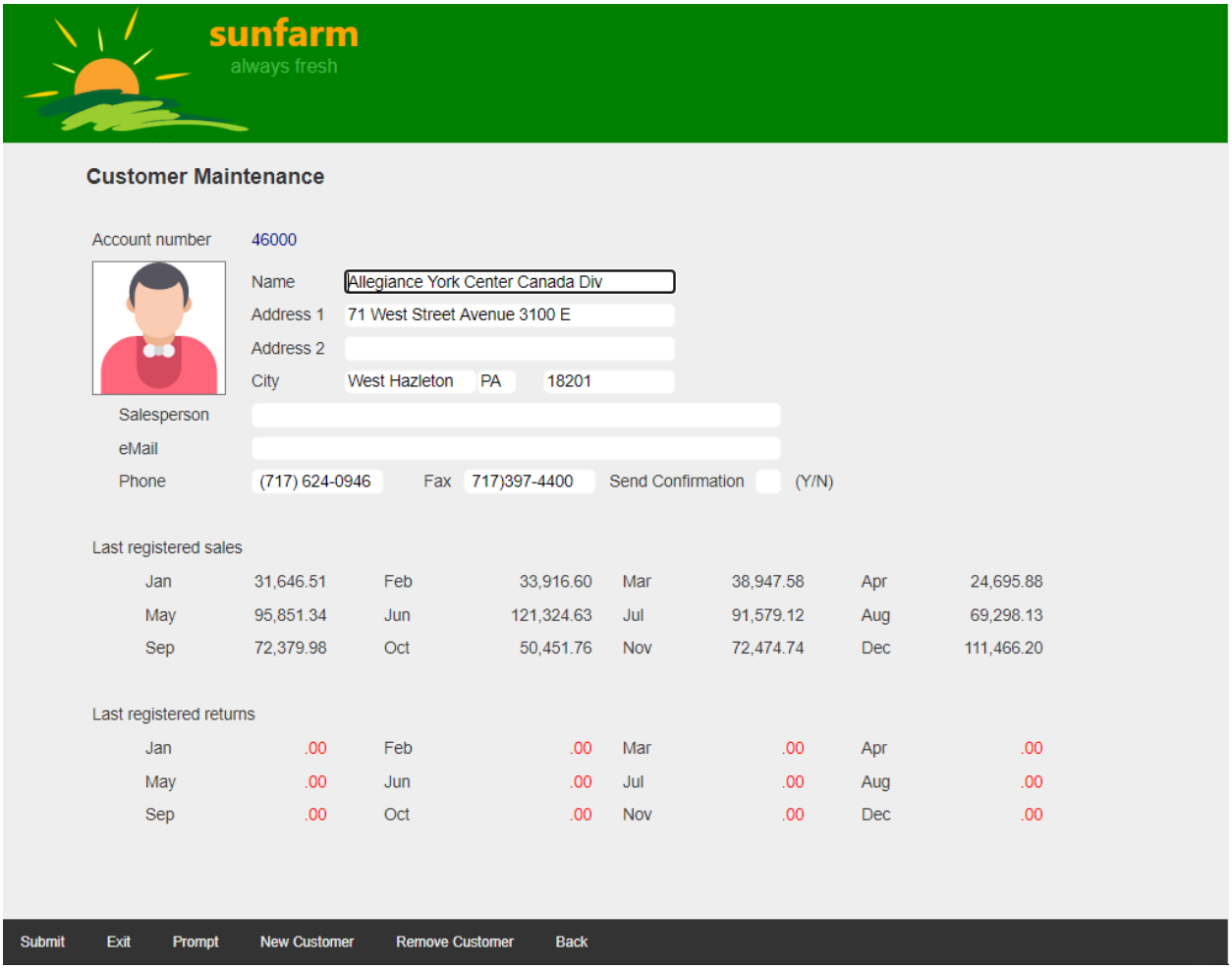
Add the following Markup to the CUSTDSP Razor Page:

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

```
<DdsRecord For="CUSTREC" StretchConstantText=false KeyNames="ENTER 'Submit'; F3 'Exit'; F4
'Prompt'; F6 'New Customer'; F11 'Remove Customer'; F12 'Back';">
.
.
.
    <div Row="17">
        <DdsConstant Col="8" Text="Last registered returns" />
    </div>

    <div Row="18">
        <DdsConstant Col="12" Text="Jan" />
        <DdsDecField Col="15" For="CUSTREC.CSRETURN01" EditCode=One Color="red" />
        <DdsConstant Col="30" Text="Feb" />
        <DdsDecField Col="35" For="CUSTREC.CSRETURN02" EditCode=One Color="red" />
        <DdsConstant Col="48" Text="Mar" />
        <DdsDecField Col="51" For="CUSTREC.CSRETURN03" EditCode=One Color="red" />
        <DdsConstant Col="66" Text="Apr" />
        <DdsDecField Col="69" For="CUSTREC.CSRETURN04" EditCode=One Color="red" />
    </div>
    <div Row="19">
        <DdsConstant Col="12" Text="May" />
        <DdsDecField Col="15" For="CUSTREC.CSRETURN05" EditCode=One Color="red" />
        <DdsConstant Col="30" Text="Jun" />
        <DdsDecField Col="35" For="CUSTREC.CSRETURN06" EditCode=One Color="red" />
        <DdsConstant Col="48" Text="Jul" />
        <DdsDecField Col="51" For="CUSTREC.CSRETURN07" EditCode=One Color="red" />
        <DdsConstant Col="66" Text="Aug" />
        <DdsDecField Col="69" For="CUSTREC.CSRETURN08" EditCode=One Color="red" />
    </div>
    <div Row="20">
        <DdsConstant Col="12" Text="Sep" />
        <DdsDecField Col="15" For="CUSTREC.CSRETURN09" EditCode=One Color="red" />
        <DdsConstant Col="30" Text="Oct" />
        <DdsDecField Col="35" For="CUSTREC.CSRETURN10" EditCode=One Color="red" />
        <DdsConstant Col="48" Text="Nov" />
        <DdsDecField Col="51" For="CUSTREC.CSRETURN11" EditCode=One Color="red" />
        <DdsConstant Col="66" Text="Dec" />
        <DdsDecField Col="69" For="CUSTREC.CSRETURN12" EditCode=One Color="red" />
    </div>
```

Note how we are setting the Color to “red” (in the Business Logic, we will convert negative values to absolute values, on the presentation layer we use red to indicate negative instead.

Re-run the Application and navigate to a Customer “Update” Page:



4

Values are there but not the ones that we want (all zeroes), we need to work more on the Business Logic.

Business Logic should remember the Last Year’s registered Sales and Returns

The Presentation layer is ready to display the current values of Sales and Returns for a given Customer.

To improve the Logic we will work on CUSTINQ program.

---

[4] Commit “Initial Returns data rendered on the Page”

While adding new code, we can rely on more modern C# logic and forget about RPG life-cycle for a change.

One possible implementation would be to declare a class in CUSTINQ program called YearData, which will help us abstract the twelve variables and perform some discreet operation on that set.

Add the following simple C# class at the end of program file CUSTINQ:

```
internal class YearData
{
    const int MONTHS_IN_YEAR = 12;
    public decimal[] month;

    public YearData()
    {
        month = new decimal[MONTHS_IN_YEAR];
    }

    public YearData(decimal jan, decimal feb, decimal mar, decimal apr, decimal may, decimal jun,
decimal jul, decimal aug, decimal sep, decimal oct, decimal nov, decimal dec) : this()
    {
        month[0] = jan;
        month[1] = feb;
        month[2] = mar;
        month[3] = apr;
        month[4] = may;
        month[5] = jun;
        month[6] = jul;
        month[7] = aug;
        month[8] = sep;
        month[9] = oct;
        month[10] = nov;
        month[11] = dec;
    }

    public decimal Sum()
    {
        decimal result = 0;
        foreach (var amt in month)
            result += amt;

        return result;
    }
}
```

Our algorithm would consist of:

1. Start lastYearSales and lastYearReturns with the minimum value.
2. Loop reading records for the Customer of interest until the Customer Number is different from what we are processing.
3. In the Loop in (2), depending on the record Type, we collect Sales or Returns in a dictionary collection that uses Year as key. We also keep the lastYearSales and lastYearReturns updated. (We know that the records are sorted by year within a Customer).
4. Once the Loop in (2) is done, we can update the CSSALESnn variable group and/or CSRETURNnn variable group, by extracting from the Dictionary the lastYear's data.

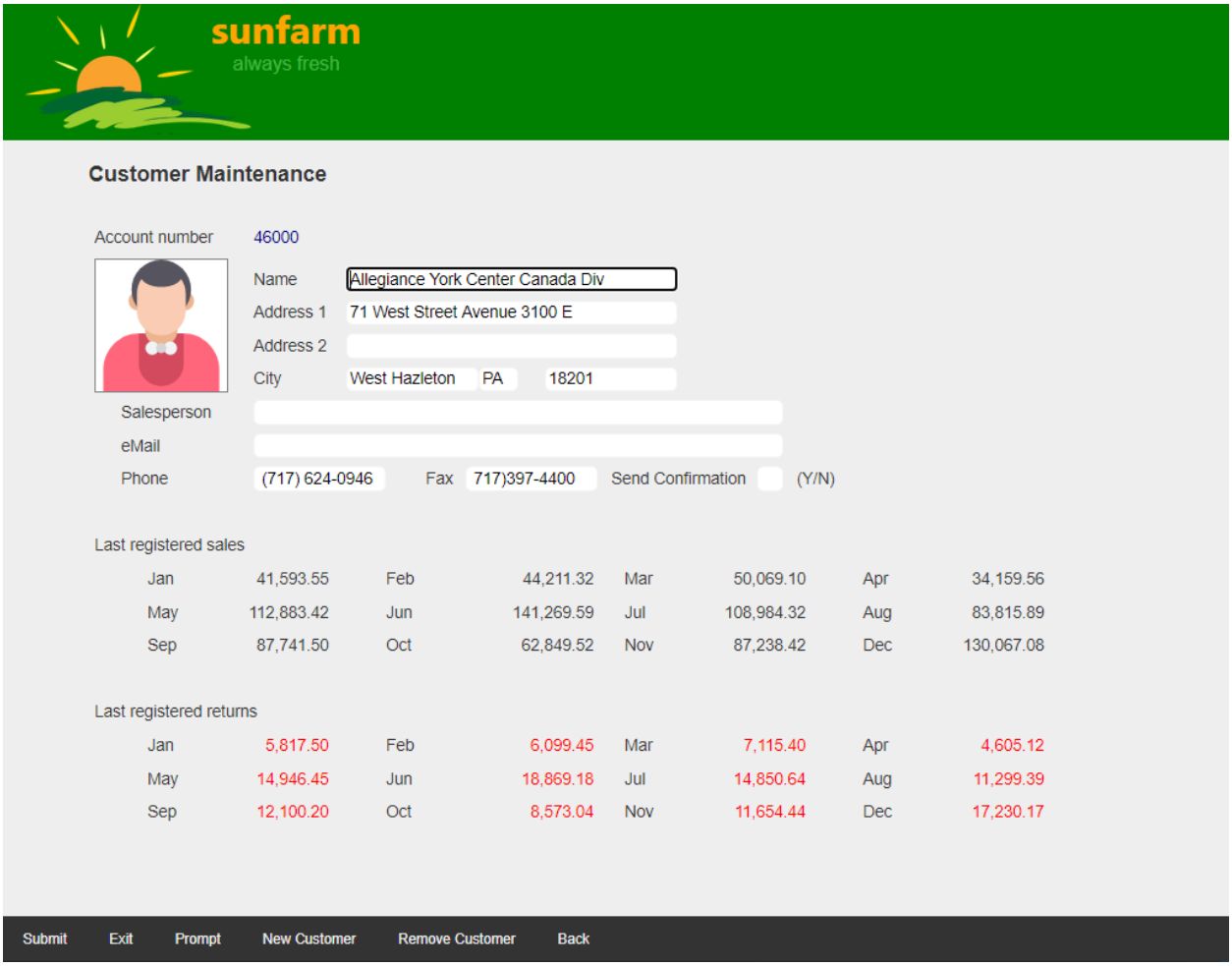
```
private void LoadLastSalesAndReturns()
{
    CSSALES01 = CSSALES02 = CSSALES03 = CSSALES04 = CSSALES05 = CSSALES06 = 0;
    CSSALES07 = CSSALES08 = CSSALES09 = CSSALES10 = CSSALES11 = CSSALES12 = 0;

    CSRETURN01 = CSRETURN02 = CSRETURN03 = CSRETURN04 = CSRETURN05 = CSRETURN06 = 0;
    CSRETURN07 = CSRETURN08 = CSRETURN09 = CSRETURN10 = CSRETURN11 = CSRETURN12 = 0;

    FixedDecimal < _9, _0> CustomerNumber = new FixedDecimal<_9, _0>();
```

```
    CustomerNumber = CMCUSTNO.MoveRight(CustomerNumber);
    if (!CSMASTERL1.Seek(SeekMode.SetLL, CustomerNumber))
        return;

    Dictionary<decimal, YearData> salesForCustomer = new Dictionary<decimal, YearData>();
    Dictionary<decimal, YearData> returnsForCustomer = new Dictionary<decimal, YearData>();
    decimal lastYearSales = decimal.MinValue;
    decimal lastYearReturns = decimal.MinValue;

    while (CSMASTERL1.ReadNextEqual(false, CustomerNumber))
    {
        if (CSTYPE == 1)
        {
            lastYearSales = CSYEAR;
            salesForCustomer.Add(lastYearSales, new YearData(CSSALES01, CSSALES02, CSSALES03,
CSSALES04, CSSALES05, CSSALES06, CSSALES07, CSSALES08, CSSALES09, CSSALES10, CSSALES11,
CSSALES12));
        }
        else
        {
            lastYearReturns = CSYEAR;
            returnsForCustomer.Add(lastYearReturns, new YearData(CSSALES01, CSSALES02, CSSALES03,
CSSALES04, CSSALES05, CSSALES06, CSSALES07, CSSALES08, CSSALES09, CSSALES10, CSSALES11,
CSSALES12));
        }
    }

    if (lastYearSales > decimal.MinValue)
    {
        CSSALES01 = salesForCustomer[lastYearSales].month[0];
        CSSALES02 = salesForCustomer[lastYearSales].month[1];
        CSSALES03 = salesForCustomer[lastYearSales].month[2];
        CSSALES04 = salesForCustomer[lastYearSales].month[3];
        CSSALES05 = salesForCustomer[lastYearSales].month[4];
        CSSALES06 = salesForCustomer[lastYearSales].month[5];
        CSSALES07 = salesForCustomer[lastYearSales].month[6];
        CSSALES08 = salesForCustomer[lastYearSales].month[7];
        CSSALES09 = salesForCustomer[lastYearSales].month[8];
        CSSALES10 = salesForCustomer[lastYearSales].month[9];
        CSSALES11 = salesForCustomer[lastYearSales].month[10];
        CSSALES12 = salesForCustomer[lastYearSales].month[11];
    }

    if (lastYearReturns > decimal.MinValue)
    {
        CSRETURN01 = returnsForCustomer[lastYearReturns].month[0];
        CSRETURN02 = returnsForCustomer[lastYearReturns].month[1];
        CSRETURN03 = returnsForCustomer[lastYearReturns].month[2];
        CSRETURN04 = returnsForCustomer[lastYearReturns].month[3];
        CSRETURN05 = returnsForCustomer[lastYearReturns].month[4];
        CSRETURN06 = returnsForCustomer[lastYearReturns].month[5];
        CSRETURN07 = returnsForCustomer[lastYearReturns].month[6];
        CSRETURN08 = returnsForCustomer[lastYearReturns].month[7];
        CSRETURN09 = returnsForCustomer[lastYearReturns].month[8];
        CSRETURN10 = returnsForCustomer[lastYearReturns].month[9];
        CSRETURN11 = returnsForCustomer[lastYearReturns].month[10];
        CSRETURN12 = returnsForCustomer[lastYearReturns].month[11];
    }
}
```

Understanding in detail how the method is correct is outside the scope of this Guide. The assumption is that the reader is familiar with C# and can read code of this complexity.[5]

[5] Commit: “Page Renders Last recorded Sales and Returns in the Year”

Build the new version and run it.



Additional Table heading data: Year, Total, Trend

Let's define one more CSS Style - to make Sales and Returns label stronger.

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

.large-bold-text {
    font-size: large !important;
    font-weight: bold;
}
```

On the row for the Table Heading "Last registered sales" we will add:
1. Year for which the sales data was found in the database
2. Total Sales
3. Sales change in percent for December compared to January for that Year

The new Model properties are:

```
    [Char(20)]
    public string YEAR_SALES { get; private set; }

    [Dec(13, 2)]
    public decimal TOTAL_SALES { get; private set; }

    [Char(20)]
    public string PERCENT_CHANGE_SALES { get; private set; }
```

Note how we define the properties as output-only (public get, private set). We define TOTAL_SALES as decimal (so we can take advantage of EditCode formatter). But since we want an uncommon formatting for the Year and Sales change in percent for December, we will use business logic with nice .net framework formatting classes to format this data as a string and use it as such.

The Sales heading (Row 12) now becomes:

```
<div Row="12">
    <DdsConstant Col="8" class="large-bold-text" Text="Last registered sales" />
    <DdsCharField Col="27" class="large-bold-text" For="CUSTREC.YEAR_SALES" />
    <DdsDecField Col="63" class="large-bold-text" For="CUSTREC.TOTAL_SALES" EditCode=One />
    <DdsCharField Col="75" class="large-bold-text" For="CUSTREC.PERCENT_CHANGE_SALES"
Color="green" />
</div>
```

On the Business Logic, IO partial class, we need to define the new DataSet variables and make sure the values are copied when the Workstation file is written/read.

```
C:\Projects\SunFarm\CustomerAppLogic\CUSTINQ.Io.cs

    .
    .
    .
    private FixedDecimal<_11, _2> CSRETURN12;

    FixedString<_20> YEAR_SALES;
    FixedDecimal<_13, _2> TOTAL_SALES;
    FixedString<_20> PERCENT_CHANGE_SALES;


private void PopulateBufferCUSTDSPFCUSTREC(AdgDataSet _dataSet)
{
    var _table = _dataSet.GetAdgTable("CUSTREC");
    System.Data.DataRow _row = _table.Row;
    _row["CSRREC"] = ((string)(CSRREC));
    .
    .
    .
    _row["YEAR_SALES"] = ((string)(YEAR_SALES));
    _row["TOTAL_SALES"] = ((decimal)(TOTAL_SALES));
    _row["PERCENT_CHANGE_SALES"] = ((string)(PERCENT_CHANGE_SALES));
}

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

```
}
```

Now, let's use the new Workstation DataSet fields in the logic. First initialize fields (empty and/or zero), then compute according to the last year read:

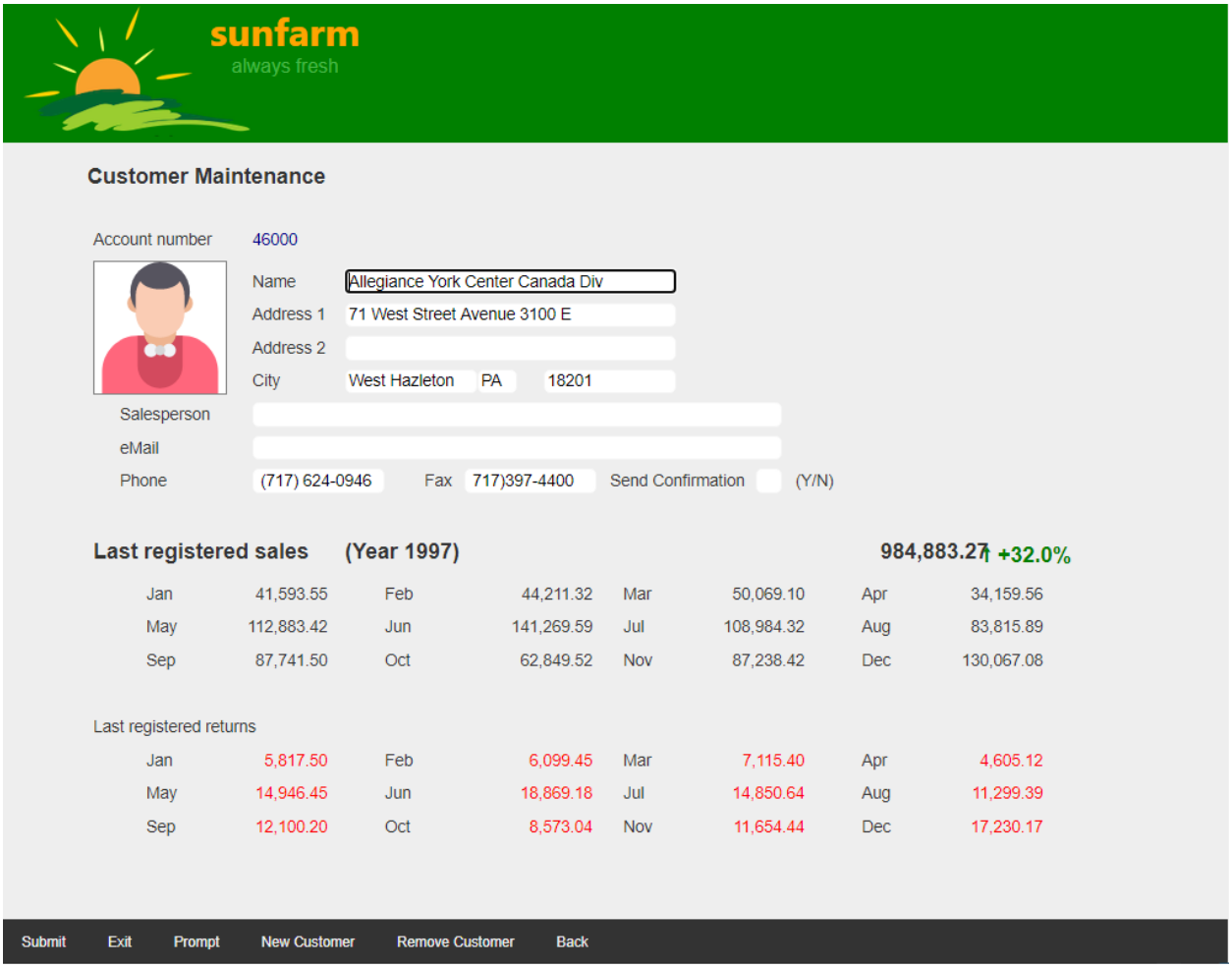
Initialize:

```
private void LoadLastSalesAndReturns()
{
    CSSALES01 = CSSALES02 = CSSALES03 = CSSALES04 = CSSALES05 = CSSALES06 = 0;
    CSSALES07 = CSSALES08 = CSSALES09 = CSSALES10 = CSSALES11 = CSSALES12 = 0;

    CSRETURN01 = CSRETURN02 = CSRETURN03 = CSRETURN04 = CSRETURN05 = CSRETURN06 = 0;
    CSRETURN07 = CSRETURN08 = CSRETURN09 = CSRETURN10 = CSRETURN11 = CSRETURN12 = 0;

    YEAR_SALES = string.Empty;
    TOTAL_SALES = 0;
    PERCENT_CHANGE_SALES = string.Empty;

    FixedDecimal< _9, _0> CustomerNumber = new FixedDecimal<_9, _0>();
.
.
.
```

Compute:

```
if (lastYearSales > decimal.MinValue)
{
    CSSALES01 = salesForCustomer[lastYearSales].month[0];
    CSSALES02 = salesForCustomer[lastYearSales].month[1];
    CSSALES03 = salesForCustomer[lastYearSales].month[2];
    .
    .
    .
    CSSALES12 = salesForCustomer[lastYearSales].month[11];

    YEAR_SALES = $"(Year {lastYearSales})";
    TOTAL_SALES = salesForCustomer[lastYearSales].Sum();

    if (CSSALES12 > CSSALES01 && CSSALES12 > 0)
    {
        decimal calc = (CSSALES01 * 100) / CSSALES12;
        PERCENT_CHANGE_SALES = $"↑ +{Math.Round(calc, 1)}%";
    }
    else if (CSSALES12 < CSSALES01 && CSSALES01 > 0)
    {
        decimal calc = (CSSALES12 * 100) / CSSALES01;
        PERCENT_CHANGE_SALES = $"↓ +{Math.Round(calc, 1)}%";
    }
}
```

Build CustomerAppLogic and Run the Website Application:



The Sales heading is more useful, but the Total Sales for the year are not aligned where we want it.

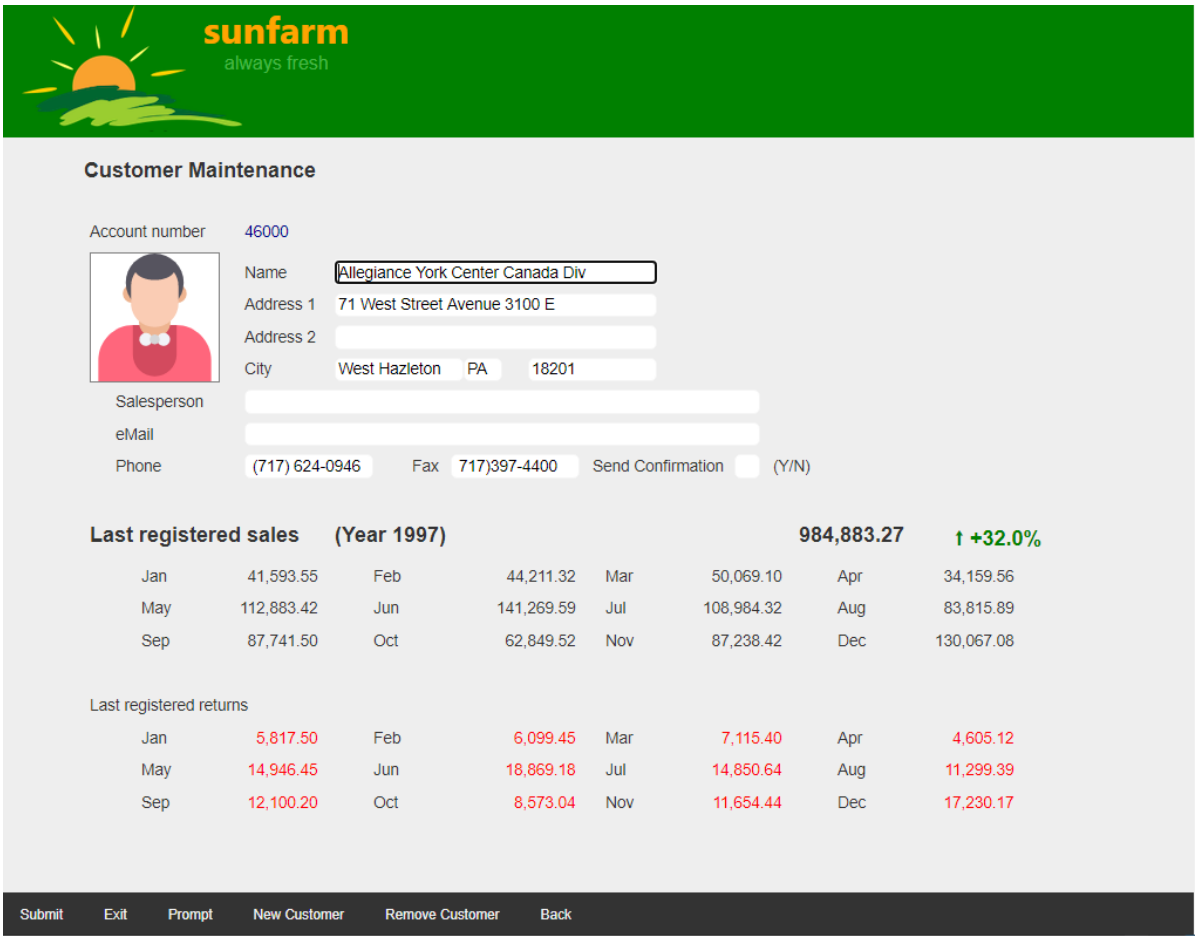
Recall that CUSTREC.TOTAL_SALES is a Decimal Field. Decimal fields are aligned to the right by default.

Let's change its CSS style to correct the problem. We have solved this problem before. We already have a CSS Style called left-aligned-field.

The field For="CUSTREC.TOTAL_SALES" in the Markup already has a class="large-bold-text", standard HTML CSS are by definition cascading, meaning that several styles may be applied. The syntax is simple, we can just add one more style separated with a space:

```
<div Row="12">
    <DdsConstant Col="8" class="large-bold-text" Text="Last registered sales" />
    <DdsCharField Col="27" class="large-bold-text" For="CUSTREC.YEAR_SALES" />
    <DdsDecField Col="63" class="large-bold-text left-aligned-field"
      For="CUSTREC.TOTAL_SALES" EditCode=One />
    <DdsCharField Col="75" class="large-bold-text" For="CUSTREC.PERCENT_CHANGE_SALES"
Color="green" />
</div>
```

Re-run the Website Application, the heading is now complete:[6]



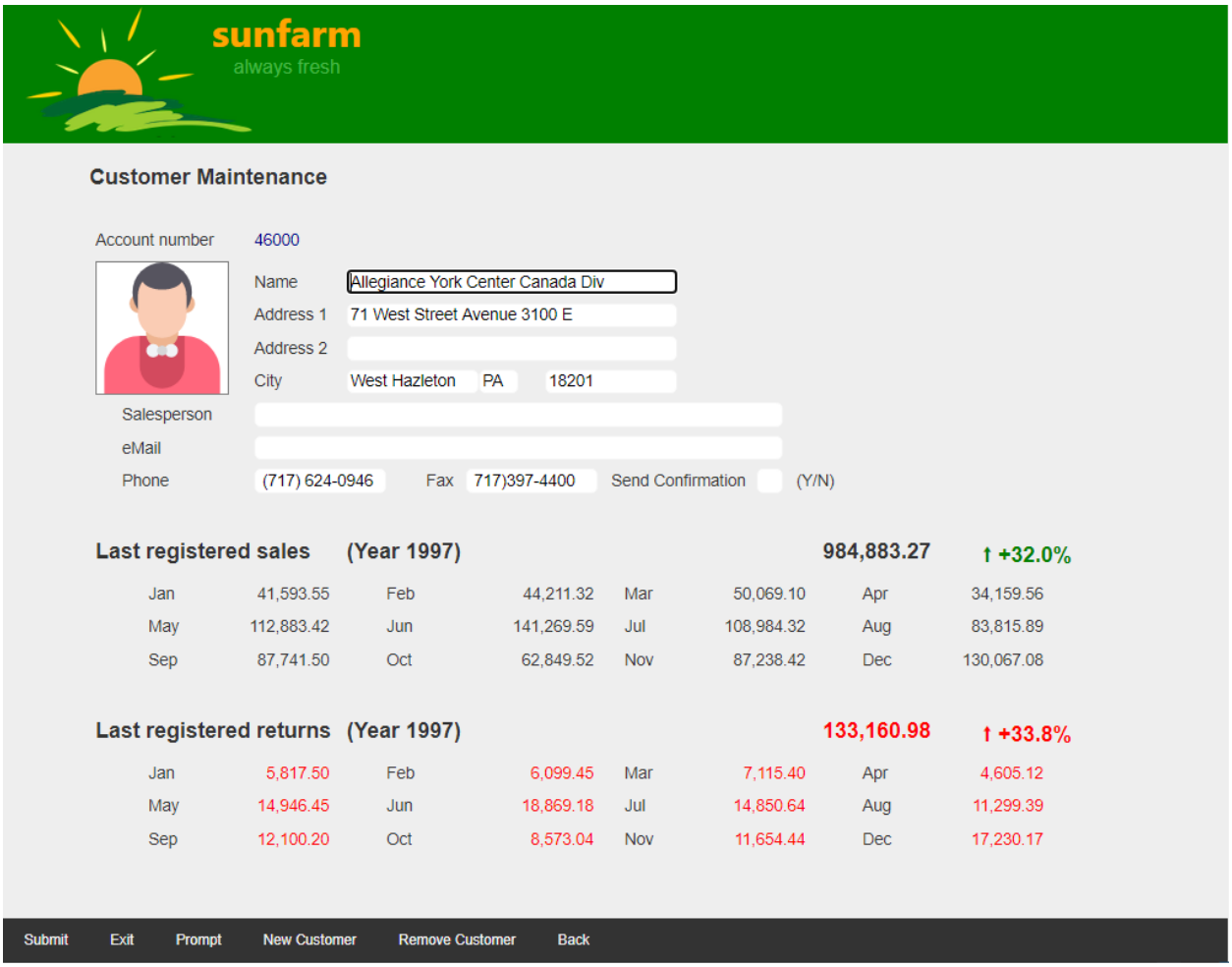
[6] Commit: "Last Registered Sales Heading"

We can repeat the steps to add a proper header to the “Last registered returns table”. We will show only the new markup, you can use the giithub project history to look at the rest of the code.

The Markup for Row 17 is:

```
<div Row="17">
    <DdsConstant Col="8" class="large-bold-text" Text="Last registered returns" />
    <DdsCharField Col="27" class="large-bold-text" For="CUSTREC.YEAR_RETURNS" />
    <DdsDecField Col="63" class="large-bold-text left-aligned-field"
      For="CUSTREC.TOTAL_RETURNS" EditCode=One Color="red" />
    <DdsCharField Col="75" class="large-bold-text" For="CUSTREC.PERCENT_CHANGE_RETURNS"
Color="red" />
</div>
```

The two headings should now show like this:[7]



[7] Commit: “Last registered returns heading complete”