Data-driven Programming

By

Johan Nel

A series of articles explaining the principles

Article 3: Customer view

November 2014

Table of contents

Source Code

Figures

1. Back to basics

In article 2 we looked at a generalised scenario using the typical Hello world application to lightheartedly describe how we typically do software development. In this article we will still use the "enhanced" hello world application as basis for describing a typical software development strategy. I have however decided to keep the applicaton menu for the next article.

2. A new technology on the horizon

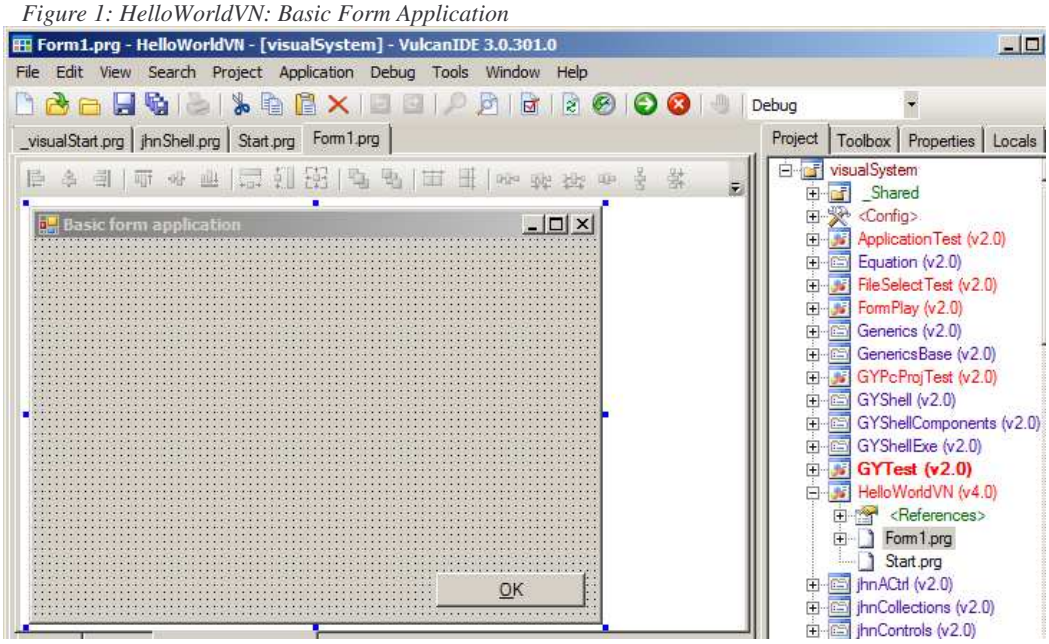
A joint application development (JAD) session was done at a client that wanted to convert to our HelloWorldWin application. It is a deal not to be missed having a client converting from one of our biggest competitors to our software. Some customisation is however required:

- The customer believe our system was too rigid and required a user to determine when Hello world, How are you? And Goodbye world will be performed;
- The customer wanted to be able to have the option of multiple execution of any of the above options in any sequence;
- The user wanted to decide when the processes were executed to satisfaction (or the users have done their 9-5 productive hours) before it will be terminated.

IT decided that this is an ideal time to leverage the current HelloWorldWin into the future and decide that Vulcan.NET is ideal for the new DOTNET technology. Software acquired, training courses attended, a functional specification is drafted from the JAD session, user acceptance is signed off, budget done and resources allocated for the estimated development time of 3 months.

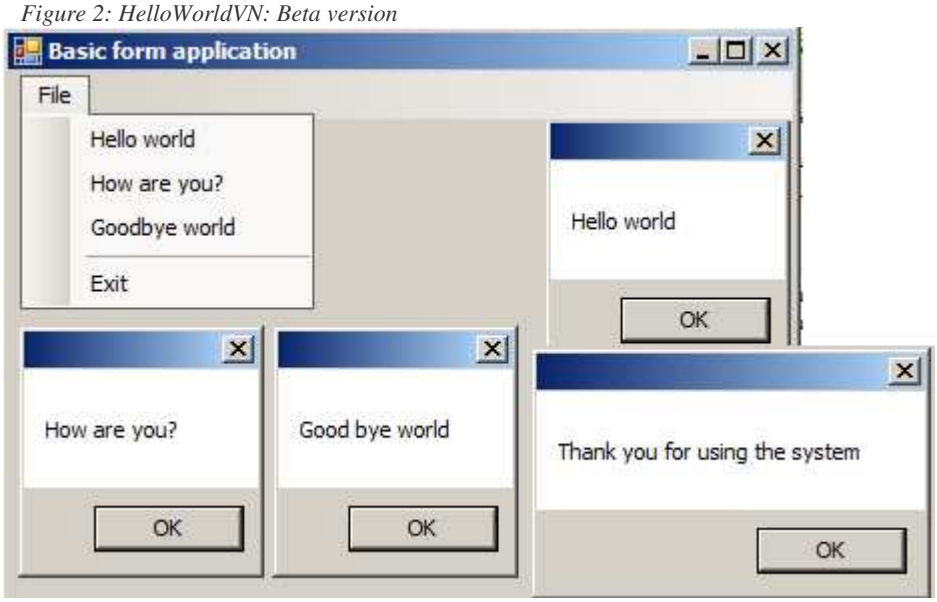
The developers realised that this is a typical DOTNET application and start by creating a new application HelloWorldVN, using the Basic Form Application in the Application Gallery (Figure 1) of their favourite development tool VIDE.

Figure 1: HelloWorldVN: Basic Form Application



Developers are stunned by the productivity and results achieved without having to write any code compared to the Clipper days, the DOTNET event model and the enhanced functionality found in the MessageBox class. Everybody is complimented regarding the strategic decision made. In record time and ahead of planning the first beta version is ready for demonstration after adding the details as per the user requirement and functional specification (Figure 2).

Figure 2: HelloWorldVN: Beta version



Demonstration arranged and presented, new client is very happy with the vision and insight shown by our company, and agree to sign the user acceptance and pay as stipulated in the contract, 70% of the quoted price, with the remainder after 60 days of implementation. Not

only does it provides all the features requested, but at no extra cost the Exit option includes a component thanking the user for using the system. Back at the office celebrations are the order of the day and the head of IT give all the developers 2 day paid leave as just reward for their commitment, hard work and long 26 hour days.

The marketing team, not to be outdone immideately set themselves targets and within 2 weeks have already signed deals for another 30 companies wanting to use our HelloWorldVN application. Subsequently, they also get some reward for achieving their targets, there are however some minor customisation that is required...

After 30 days of the system running at our first HelloWorldVN client the CEO storms into the IT department needing answers why we have not done the job according to customer requirement. Our customer has seen a 30% loss in production since converting from the competitor's application. IT cannot believe the statement since the system is doing exactly as was required by the client. The IT department arrange a brainstorming session and the code reviewed since nothing can be found looking at the design from the Window Editor (Listing 1):

Listing 1: HelloWorldVN: Version 1.0 source code

```
PARTIAL CLASS BasicForm INHERIT System.Windows.Forms.Form

  PROTECT oMenuStrip1 AS System.Windows.Forms.MenuStrip
  PROTECT oToolStripMenuItem1 AS System.Windows.Forms.ToolStripMenuItem
  PROTECT oToolStripMenuItem2 AS System.Windows.Forms.ToolStripMenuItem
  PROTECT oToolStripMenuItem3 AS System.Windows.Forms.ToolStripMenuItem
  PROTECT oToolStripMenuItem4 AS System.Windows.Forms.ToolStripMenuItem
  PROTECT oToolStripSeparator1 AS System.Windows.Forms.ToolStripSeparator
  PROTECT oToolStripMenuItem5 AS System.Windows.Forms.ToolStripMenuItem
  // User code starts here (DO NOT remove this line)  ##USER##
  METHOD InitializeForm() AS VOID
  // IDE generated code (please DO NOT modify)
     SELF:oMenuStrip1 := System.Windows.Forms.MenuStrip{}
     SELF:oToolStripMenuItem1 := System.Windows.Forms.ToolStripMenuItem{}
     SELF:oToolStripMenuItem2 := System.Windows.Forms.ToolStripMenuItem{}
     SELF:oToolStripMenuItem3 := System.Windows.Forms.ToolStripMenuItem{}
     SELF:oToolStripMenuItem4 := System.Windows.Forms.ToolStripMenuItem{}
     SELF:oToolStripSeparator1 := System.Windows.Forms.ToolStripSeparator{}
     SELF:oToolStripMenuItem5 := System.Windows.Forms.ToolStripMenuItem{}

     SELF:SuspendLayout()

     SELF:ClientSize := System.Drawing.Size{392 , 264}
     SELF:Location := System.Drawing.Point{100 , 100}
     SELF:Name := "BasicForm"
     SELF:Text := "Basic form application"

     SELF:oToolStripMenuItem1:Name := "ToolStripMenuItem1"
     SELF:oToolStripMenuItem1:Text := "File"
     SELF:oMenuStrip1:Items:Add( SELF:oToolStripMenuItem1)

     SELF:oToolStripMenuItem2:Click += System.EventHandler{ SELF , @HelloWorldClick() }
     SELF:oToolStripMenuItem2:Name := "ToolStripMenuItem2"
     SELF:oToolStripMenuItem2:Text := "Hello world"
     SELF:oToolStripMenuItem1:DropDown:Items:Add( SELF:oToolStripMenuItem2)

     SELF:oToolStripMenuItem3:Click += System.EventHandler{ SELF , @HowAreYouClick() }
     SELF:oToolStripMenuItem3:Name := "ToolStripMenuItem3"
     SELF:oToolStripMenuItem3:Text := "How are you?"
     SELF:oToolStripMenuItem1:DropDown:Items:Add( SELF:oToolStripMenuItem3)
```

```
    SELF:oToolStripMenuItem4:Click += System.EventHandler{ SELF , @GoodbyeWorldClick() }
    SELF:oToolStripMenuItem4:Name := "ToolStripMenuItem4"
    SELF:oToolStripMenuItem4:Text := "Goodbye world"
    SELF:oToolStripMenuItem1:DropDown:Items:Add( SELF:oToolStripMenuItem4)

    SELF:oToolStripSeparator1:Name := "ToolStripSeparator1"
    SELF:oToolStripSeparator1:Text := "-"
    SELF:oToolStripMenuItem1:DropDown:Items:Add( SELF:oToolStripSeparator1)

    SELF:oToolStripMenuItem5:Click += System.EventHandler{ SELF , @ExitClick() }
    SELF:oToolStripMenuItem5:Name := "ToolStripMenuItem5"
    SELF:oToolStripMenuItem5:Text := "E&xit"
    SELF:oToolStripMenuItem1:DropDown:Items:Add( SELF:oToolStripMenuItem5)

    SELF:oMenuStrip1:Location := System.Drawing.Point{0 , 0}
    SELF:oMenuStrip1:Name := "MenuStrip1"
    SELF:oMenuStrip1:Size := System.Drawing.Size{392 , 24}
    SELF:oMenuStrip1:TabIndex := 0
    SELF:Controls:Add(SELF:oMenuStrip1)

    SELF:ResumeLayout()

  RETURN

END CLASS
```

Two issues are identified during the review:

- There are redundant code in the application, all MenuClickEvents do exactly the same, hence creating unneccary overhead and complexity;
- Some oversight during Q&A since the application title does not show HelloWorldVN, but still show the default Basic Form Application, which might create confusion in selecting the correct application to do the job in between all the running applications on the user's computer, it will definitely enhance productivity if the HelloWorldVN application can be found quickly and not through trial and error.

IT department immediately get into action in making the application more efficient. The application title is changed and a new policy is implemented that MenuClickEvents will all call the same MenuItemClick event method (Listing 2).

Listing 2: Changes made to HelloWorldVN

```
METHOD InitializeForm()
  …
  SELF:Text := "Hello World VN - [V1.0]"
…
  SELF:oToolStripMenuItem2:Click += EventHandler{SELF, @MenuItemClick()}
  SELF:oToolStripMenuItem3:Click += EventHandler{SELF, @MenuItemClick()}
  SELF:oToolStripMenuItem4:Click += EventHandler{SELF, @MenuItemClick()}
…
RETURN

METHOD MenuItemClick(o AS OBJECT, e AS System.EventArgs) AS VOID
  MessageBox.Show(((ToolStripMenuItem)o):Text)
RETURN
```

The IT department feel they have made progress and a new release is deployed. When final payment is due, the customer report that no improvement in production was achieved and they demand the 70% down payment to be reversed. The IT department is up in arms since they have made the programme as efficient as it can be, it provides all the functionality

required, redundant code was elimated and the programme size was drastically reduced. Project slippages is the norm of the day and the new software project that were to be released Q4 has to be postponed to next year, due to most developers tied into customising new sales of HelloWorldVN and trying to resolve the issues at our first and biggest customer…

3. Customers are difficult to please

If we look at the above example there is nothing wrong with our design. However, the one aspect that the development team missed was an effective user interface. The system does what is required, there is nothing wrong with the coding principles, in default IDE Window Editor drag and drop development or reducing through some programming standards the number of MenuClick events that are created. However, from a user perspective the number of MessageBox.Show(<text>) events that can be performed in a certain period of time can drastically increase if instead of a MenuItem to activate the event, buttons on the form was used. The number of lines of code would practically stay exactly the same. No difference in functionality, just a different way of implementation keeping the end-user in mind.

4. Applications are merely data in the data-driven scenario

Enough said about programming standards. We are not here for that. We will still look at our HelloWorldVN application and what we can do to enhance it.

Firstly, if we look at the code we see that we have an application with BasicForm with not too much detail on it, except for a MenuStrip. The MenuStrip on its own does not really have much usage, except it will be displayed on the Form. To make the MenuStrip useful, we need to add ToolStripMenuItems, each item have some properties describing it and the same event can be performed on each of the MenuItems, MessageBox.Show(<Item>:Text) or terminate the application when Exit is clicked.

So let us take our application form and adapt it in a couple of steps with software developers' favourite tool called Copy & Replace:

- Create a new empty file called CustomerView.prg;
- Copy the source code from Form1.prg to CustomerView.prg;
- In CustomerView.prg replace all occurances of BasicForm with CustomerView;
- Replace all occurances of MenuStrip1 with CustromerViewGrid;

- Replace all occurances of oToolStripMenuItem with oCustomer.

The changes and removing of some redundancy to limit paper usage are listed in Listing 3.

Listing 3: End result of Copy and Replace

```
PARTIAL CLASS CustomerView INHERIT System.Windows.Form.Form

  PROTECT oCustomerViewGrid AS System.Windows.Forms.MenuStrip
  PROTECT oCustomer1 AS System.Windows.Forms.ToolStripMenuItem
  PROTECT oCustomer2 AS System.Windows.Forms.ToolStripMenuItem
  PROTECT oToolStripSeparator1 AS System.Windows.Forms.ToolStripSeparator
  PROTECT oCustomer5 AS System.Windows.Forms.ToolStripMenuItem
  // User code starts here (DO NOT remove this line)  ##USER##

  METHOD InitializeForm() AS VOID
  // IDE generated code (please DO NOT modify)
     SELF:oCustomerViewGrid := System.Windows.Forms.MenuStrip{}
     SELF:oCustomer1 := System.Windows.Forms.ToolStripMenuItem{}
     SELF:oCustomer2 := System.Windows.Forms.ToolStripMenuItem{}
     SELF:oToolStripSeparator1 := System.Windows.Forms.ToolStripSeparator{}
     SELF:oCustomer5 := System.Windows.Forms.ToolStripMenuItem{}

     SELF:SuspendLayout()

     SELF:ClientSize := System.Drawing.Size{392 , 264}
     SELF:Location := System.Drawing.Point{100 , 100}
     SELF:Name := "BasicForm"
     SELF:Text := "HelloWorldVN"

     SELF:oCustomer1:Name := "Customer1"
     SELF:oCustomer1:Text := "File"
     SELF:oCustomerViewGrid:Items:Add( SELF:oCustomer1)

     SELF:oCustomer2:Click += System.EventHandler{ SELF , @MenuItemClick() }
     SELF:oCustomer2:Name := "ToolStripMenuItem2"
     SELF:oCustomer2:Text := "Hello world"
     SELF:oCustomer1:DropDown:Items:Add( SELF:oCustomer2)

     SELF:oToolStripSeparator1:Name := "ToolStripSeparator1"
     SELF:oToolStripSeparator1:Text := "-"
     SELF:oCustomer1:DropDown:Items:Add( SELF:oToolStripSeparator1)

     SELF:oCustomer5:Click += System.EventHandler{ SELF , @ExitClick() }
     SELF:oCustomer5:Name := "ToolStripMenuItem5"
     SELF:oCustomer5:Text := "E&xit"
     SELF:oCustomer1:DropDown:Items:Add( SELF:oToolStripMenuItem5)

     SELF:oCustumerViewGrid:Location := System.Drawing.Point{0 , 0}
     SELF:oCustomerViewGrid:Name := "CustomerViewGrid"
     SELF:oCustomerViewGrid:Size := System.Drawing.Size{392 , 24}
     SELF:oCustomerViewGrid:TabIndex := 0
     SELF:Controls:Add(SELF:oCustomerViewGrid)

     SELF:ResumeLayout()

  RETURN

  METHOD MenuItemClick(o AS OBJECT, e AS System.EventArgs) AS VOID
     MessageBox.Show("Hello " + ((ToolStripMenuItem)o):Text)
  RETURN

  METHOD ExitClick(o AS OBJECT, e AS System.EventArgs) AS VOID
     MessageBox.Show("Goodbye " + ((ToolStripMenuItem)o):Text)
  RETURN

END CLASS
```

If we look at the marvellous development work that we have just done, in no time we have defined a new FormClass describing something that looks very similar to a badly designed CustomerView. Trying to read this in simple English does not sound very good, since it makes no sense that oCustomerViewGrid is of type MenuStrip. That is as good as telling

John that he belongs to the family Bird and that he is covered in purple feathers and has a wingspan of 1 metres. I would not blame John if he believes we smoking or sniffing something…

As efficient software developers we know how to rectify the situation in CustomerView.prg that was created by newly appointment fresh from college Rookie1:

- Replace all occurances of MenuStrip with DataGridView;
- Create a new variable called oCustList of type List<Customer>;
- Add Customer1,2, 5 to List<Customer>, etc.

As you might be aware our new CustomerView form is starting to take on the shape of a well defined Customer View using good practise structured OOP methodology. The big part that is still missing is that customer data should resides outside the application, it should not be hard coded. So the next step is to define an external source to hold our customer data and bind our customer view to the external data source. I am not going to go into the details since I believe most readers are capable to do this exercise.

Just some questions/statements that comes to mind:

- CustomerView need to know how many customers we have at compile time? No it will be resolved at runtime after all customers are loaded;
- CustomerView need to know the unique ID of each customer during compilation for us to develop a CustomerView? No it can be retrieved at runtime;
- CustomerView need to know the descriptive data of each customer at compile time? No it can get the information at runtime.

With the above statements I conclude this article, with hopefully some fruit for thougt.

5. Summary

In this article we have looked at some standard practises used in the design of applications. We also looked from a different angle at a menu and how we can adapt it to describe a typical programming task a Customer View. In the next article we will look at how we can use the lessons learned from creating a customer view and apply it to a menu view.

Hope you enjoyed reading the article and that it got some data-driven brain cells into action…

Till the next article: The application menu.