Data-driven Programming

By

Johan Nel

A series of articles explaining the principles

Article 8: Multiple applications

December 2014

Table of contents

Listings

Figures

1. Introduction

In article 7 we have extended our data-driven application to make use of two core driver classes namely jhnMemberInterface and jhnSetupDict that in combination get and serve the other classes in our Hello World application. In this article we will look at multiple applications inside the data-driven framework. Unfortunately I am getting a bit side-tracked and our RDBMS will have to wait till hopefully only the next article in the series.

2. Scenario

Our clients are quite happy with how we leveraged our application into the .net world. One of our software users however came up with a very futuristic idea. They want a tool that can assist them to guide companies into the future. A JAD session was held and during post-review by the IT team, it was unanimously agreed that we can deliver the solution without additional resources and with almost 100% guarantee that it will be within project budget and on time. The development team tasked with the problem came back and had the following specification:

- They need to create a new class called FutureVNApp;
- They need to create an AppForm class called FutureAppForm;
- They need to create a new MainMenu class called FutureAppMainMenu;
- The FutureAppMainMenu will contain a SubMenu class called FutureSubMenu;
- The FutureSubMenu will contain MenuItems for each of the tasks at hand and display the necessary guidance in a MessageBox.Show when clicked;
- The Start section of the jhnIniFile will point to this new application class and execute the FutureVNApp.

With everybody in agreement, the team sets off to do the development with targeted delivery of the beta version in 30 working days.

At the last meeting, a new intern just started his practical with the company and was asked to attend the meeting to gather experience in software development methodology. Needless to say, full of theoretical knowledge, developers soon find him becoming a nuisance, since he is constantly asking question that most of them have already forgotten about, and a feeling inside the IT team starts growing that this new kid on the block have so much to learn, but is a pain in the butt. We get paid to rollout code and not to answer some silly questions about why we doing things the way we do. He will soon discover that you doing things the way you are because that is the way it works in reality.

Observing the developers working on the task at hand, one day he said he feel they approaching the development incorrectly. Well, the coordinator, knowing the intern will only be a couple of days still with the company, suggest why don't he go and create some proof of concept of how he think it should be done. The intern agreed and for the next week everybody is quite happy with the newfound freedom of not having to answer some silly questions. All agree that it was quite a cunning move by the coordinator to get the kid off their cases.

3. The beta demonstration

Two days before the deadline the development team is ready with there demonstration. A presentation to IT management is arranged and the following demonstrated:

3.1 Modifications to ddFrameWork.exe.ini

The following additions and changes ([start] class=6) were made to the ini file:

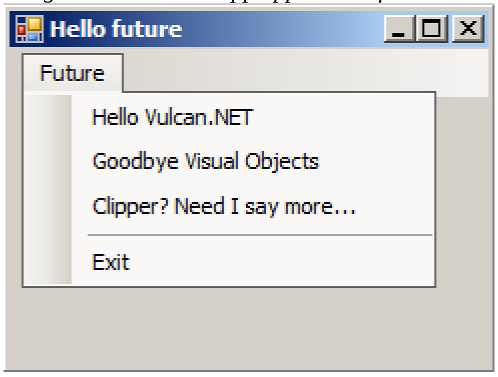
Listing 1: ddFrameWork.exe.ini changes for FutureVNApp

```
[start]
class=6
[class]
…
6=classtype_no:1;class_id:FutureVNApp;text:Hello future
7=classtype_no:2;class_id:FutureVNForm;text:Hello future application form
8=classtype_no:3;class_id:FutureMainMenu;descript:FutureApp main menu
9=classtype_no:3;class_id:FutureSubMenu;text:FutureApp sub menu
[classmember]
…
7=class_no:6;membertype_no:2;member_no:7;seq:0
8=class_no:7;membertype_no:3;member_no:8;seq:0
9=class_no:8;membertype_no:4;member_no:9;seq:0;text:&Future
10=class_no:9;membertype_no:4;member_no:5;seq:0;text:Hello &Vulcan.NET
…
```

3.2 Demonstration of the application: FutureVNApp

Demonstration of the system also went according to plan and all are happy with the result (Figure 1).

Figure 1: FutureVNApp application form



3.3 *The intern wants to speak*

As the meeting was getting to a close and the chairman ask if somebody still wants to add something, our intern that was lately very occupied and refrained from making a nuisance of himself, asked if he can say a few words, since it is the last time before he goes back to college with the whole IT team together. Everybody agrees and he gave his farewell speach…

4. Redundancy in our application

The intern thanks the team for their support during the time spend at the company and wish them well for the future and hope that he can continue building his knowledge at the company during holidays. During the last couple of weeks, he found some ways that he feels the company could benefit from. Out comes the laptop and presentation with all getting prepared for a *"this is going to be a long and boring (yawn)"* day.

4.1 *Adding class members*

Firstly the intern presented findings regarding our Application, AppForm, Menu and MenuItem classes. There are a lot of repetive code in each of the initialization methods to add members to a class (Listing 2):

Listing 2: Repeatitive code in Initialize methods

```
METHOD InitializeForm(oPC AS jhnParameterCollection) AS VOID
  SELF:nID := oPC:GetInt("member_no")
  SELF:Name := oPC:GetParameter("member_id")
  SELF:Text := oPC:GetParameter("text")
  SELF:SuspendLayout()
  SELF:ControlsAdd()
  SELF:ResumeLayout()
RETURN

METHOD ControlsAdd() AS VOID
  LOCAL aMbr AS jhnParameterCollection[]
  aMbr := jhnSetupDict.Inst:ClassMemberGet(SELF:nID)
  BEGIN SCOPE
     LOCAL delCtrlAdd AS MemAdd
     delCtrlAdd := MemAdd{SELF, @ControlAdd()}
     FOR LOCAL mbr := 0 AS INT UPTO aMbr:Length - 1
        jhnMemberInterface.Inst:MemberAdd(aMbr[mbr], delCtrlAdd)
     NEXT
  END SCOPE
RETURN

METHOD ControlAdd(o AS OBJECT) AS VOID
  SELF:Controls:Add((Control)o)
RETURN
```

He therefore changed it to rather include it in the jhnMemberInterface class and modified the <Member[s]>Add() methods as follow (Listing 3):

Listing 3: Redundancy removed from Initialize methods

```
METHOD InitializeForm(oPC AS jhnParameterCollection) AS VOID
  LOCAL delCtrlAdd AS MemAdd
  SELF:nID := oPC:GetInt("member_no")
  SELF:Name := oPC:Get("member_id")
  SELF:Text := oPC:Get("text")
  delCtrlAdd := MemAdd{SELF, @ControlAdd()}
  SELF:SuspendLayout()
  jhnMemberInterface.Inst:MemberAdd(SELF:nID, delCtrlAdd)
  SELF:ResumeLayout()
RETURN

METHOD ControlAdd(o AS OBJECT) AS VOID
  SELF:Controls:Add((Control)o)
RETURN
```

Two new overloaded MemberAdd() methods were created in the MemberInterface class, accepting the Owner class number [and delegate MemAdd] (Listing 4):

Listing 4: Additional overloaded MemberAdd() methods of the jhnMemberInterface class

```
METHOD MemberAdd(iCls AS INT) AS OBJECT[]
  LOCAL aMbr AS jhnParameterCollection[]
  LOCAL o AS System.Collections.Generic.List<OBJECT>
  o := System.Collections.Generic.List<OBJECT>{}
  aMbr := jhnSetupDict.Inst:ClassMemberGet(iCls)
  FOR LOCAL mbr := 0 AS INT UPTO aMbr:Length - 1
     o:Add(SELF:MemberAdd(aMbr[mbr]))
  NEXT
RETURN o:ToArray()

METHOD MemberAdd(iCls AS INT, memadd AS MemAdd) AS VOID
  LOCAL aO AS OBJECT[]
  LOCAL cnt AS INT
  aO :=  SELF:MemberAdd(iCls)
  cnt := aO:Length - 1
  FOR LOCAL i := 0 AS INT UPTO cnt
     memadd(aO[i])
  NEXT
RETURN
```

Suddenly it went very quiet in the boardroom and everybody sat on the edge of his or her seat…

4.2 The start is obsolete

Secondly the intern states that the [start] section is obsolete in the ini file. He presented his alternative to the ApplicationDriver class, indicating that the start or application can be found by interrogating the LkpItemGet(<classtype=”application”>) method of the SetupDict class.

In the event that more than 1 application is found, he addressed by a new class type “applist” (Listing 5).

Listing 5: Application driver class modification

```
INTERNAL METHOD Exec() AS VOID
  STATIC LOCAL iCount := 0 AS INT
  IF iCount++ = 0
     BEGIN SCOPE
        LOCAL appidx AS INT[]
        LOCAL ddSD AS jhnSetupDict
        appidx := (ddSD := jhnSetupDict.Inst):LkpItemGet("application")
        IF appidx:Length = 1
           jhnMemberInterface.Inst:MemberAdd(ddSD:ClassPropertyGet(appidx[0]))
        ELSEIF appidx:Length > 1
           jhnMemberInterface.Inst:MemberAdd(;
                                      ddSD:ClassPropertyGet(ddSD:LkpItemGet("applist")[0]))
        ELSE
           MessageBox.Show("No application defined!", ;
                             SELF:GetType():ToString() + ":Exec()", ;
                             MessageBoxButtons.OK, MessageBoxIcon.Stop)
        ENDIF
     END SCOPE
  ELSE
     MessageBox.Show(;
           "Only one instance of the application driver is allowed per active session!", ;
           SELF:GetType():ToString(), ;
           MessageBoxButtons.OK, MessageBoxIcon.Stop)
  ENDIF
RETURN
```

The following additions were made to the ini file (Listing 6):

Listing 6: The intern’s additions to the ini file

```
[lkpitem]
…
12=lkpdef_no:1;lkpitem_id:applist;defaultclass:jhnApplication
[class]
13=classtype_no:12;class_id:AppList;text:Data-driven applications
14=classtype_no:2;class_id:AppListForm;text:Application list form
15=classtype_no:3;class_id:AppListMenu;text:Menu of Applications
16=classtype_no:3;class_id:AppListSubMenu;descript:Sub menu of applications
…
[classmember]
14=class_no:13;membertype_no:2;member_no:14;seq:0
15=class_no:14;membertype_no:3;member_no:15;seq:0
16=class_no:15;membertype_no:4;member_no:16;seq:0;text:&Application
…
```

The overloaded Exec(<class_no>) is also obsolete and no longer needed. By now everybody is looking at the intern with different eyes…

4.3 The application class need some changes

Thirdly, the intern presented changes he made to the Application class (Listing 7). He states that this might all be confusing, however it will be explained when he present the changes made to the MenuItem class.

In principle, based on the application list, if more than 1 application is found, a new AppForm will be displayed with menu items for each application. The new AppForm will become the main application form and any application activated will basically be a [data]form of the

AppForm.

By this time there are different thoughts going through the minds of those in the boardroom.

Listing 7: The data-driven application class changes

```
HIDDEN METHOD InitializeApp(oPC AS jhnParameterCollection) AS VOID
  STATIC iApp := 0 AS INT
  SELF:nID := oPC:GetInt("class_no")
  SELF:Name := oPC:Get("class_id")
  SELF:Text := oPC:Get("text")
  BEGIN SCOPE
     LOCAL aMbr AS jhnParameterCollection[]
     LOCAL aO AS OBJECT[]
     LOCAL cnt AS INT
     aO := jhnMemberInterface.Inst:MemberAdd(SELF:nID)
     IF (cnt := aO:Length - 1) < 0
        oPC:DisplayMembers("No members found")
     ENDIF
     FOR LOCAL i := 0 AS INT UPTO cnt
        IF aO[i]:GetType():IsSubclassOf(typeof(Form))
           BEGIN SCOPE
              LOCAL oForm AS Form
              oForm := (Form)aO[i]
              IF iApp++ = 0
                 oForm:IsMdiContainer := TRUE
                 oForm:StartPosition := FormStartPosition.CenterScreen
                 oForm:ClientSize := System.Drawing.Size{250, 200}
                 Application.Run(oForm)
              ELSE
                 oForm:Show()
              ENDIF
           END SCOPE
        ELSE
           MessageBox.Show(e"Unknown class member!\nExpecting a form\n\n" + ;
                             Object type returned\t: " + aO[i]:GetType():ToString(), ;
                             SELF:GetType():ToString() + ":InitializeApp()", ;
                             MessageBoxButtons.OK, MessageBoxIcon.Error)
        ENDIF
     NEXT
  END SCOPE
RETURN
```

Needless to say, the older developers have totally lost track of what the intern is talking about and have started to think about early retirement…

4.4 The menu item class changes

Fourthly, the intern made some changes to the MenuItem class. He created an abstract base MenuItem class and subclassed the MenuItem class inheriting from the base class. Two initialize methods were created. One building a list of submenu items from applications found in the ini file that is added to the AppListSubMenu and an initialize method for normal menu items (Listing 8 and Listing 9). The intern then speaks about how the menuitem is internally used, and the couple of people in the audience that have followed him till now, also starts wandering if they are in the wrong industry and should look at a career change.

All said and done the intern then present the IT team with a live demonstration of the modifications and to all present, it seems that what he did is actually doing the job.

The meeting is adjourned and the IT manager requests the intern to please come see him.

Listing 8: The data-driven menu item abstract class

```
#using System.Windows.Forms
#using jhnFT.Utils.Config

INTERNAL ABSTRACT CLASS jhnMenuItemBase INHERIT ToolStripMenuItem
  PROTECT nID, nOwnerID AS INT

  CONSTRUCTOR()
     SUPER()
  RETURN

  PROPERTY Class_No AS INT
     GET
        RETURN SELF:nID
     END GET
  END PROPERTY

  PROPERTY Owner_No AS INT
     GET
        RETURN SELF:nID
     END GET
  END PROPERTY

  PROTECTED METHOD MenuItemClick(o AS OBJECT, e AS EventArgs) AS VOID
     LOCAL oPC AS jhnParameterCollection[]
     LOCAL cnt AS INT
     oPC := jhnSetupDict.Inst:ClassMemberGet(((jhnMenuItemBase)o):Class_No)
     cnt := oPC:Length - 1
     IF cnt >= 0
        FOR LOCAL i := 0 AS INT UPTO cnt
           LOCAL obj AS OBJECT
           obj := jhnMemberInterface.Inst:MemberAdd(oPC[i])
           IF obj:GetType():IsSubclassOf(typeof(Form))
              ((jhnMenuItemBase)o):ChangeEnabledStatus()
              BEGIN SCOPE
                 LOCAL oForm AS Form
                 oForm := (Form)obj
                 oForm:FormClosed += ;
                      FormClosedEventHandler{(jhnMenuItemBase)o , ;
                                                 @MenuEventChangeEnabledStatus()}
                 oForm:Show()
              END SCOPE
           ELSE
              MessageBox.Show(((ToolStripMenuItem)o):Text:Replace("&", ""))
           ENDIF
        NEXT
     ELSE
        MessageBox.Show(((ToolStripMenuItem)o):Text:Replace("&", ""))
     ENDIF
  RETURN

  PROTECTED ;
  METHOD MenuEventChangeEnabledStatus(o AS OBJECT, e AS FormClosedEventArgs) AS VOID
     SELF:ChangeEnabledStatus()
  RETURN

  HIDDEN METHOD ChangeEnabledStatus() AS VOID
     SELF:Enabled := !SELF:Enabled
  RETURN

  PROTECTED METHOD MenuItemProcess(o AS OBJECT) AS VOID
     IF o:GetType():IsSubclassOf(typeof(ToolStripItem))
        SELF:MenuItemAdd((ToolStripItem)o)
     ELSE
        MessageBox.Show("Unable to process object\n\nObject type\t: " + ;
                             o:GetType():ToString(), ;
                          SELF:GetType():ToString() + ":MenuItemProcess(o)")
     ENDIF
  RETURN

  PROTECTED METHOD MenuItemAdd(o AS ToolStripItem) AS VOID
     IF o:GetType():IsSubclassOf(typeof(ToolStripMenuItem)) && ;
           !((ToolStripMenuItem)o):HasDropDown
        ((ToolStripMenuItem)o):Click += EventHandler{SELF, @MenuItemClick()}
     ENDIF
     SELF:DropDown:Items:Add(o)
  RETURN

  PROTECTED METHOD MenuClose(o AS OBJECT, e AS EventArgs) AS VOID
     MessageBox.Show("Thank you for using the application\n” + ;
                     Hope to see you soon again\n\n" + SELF:Text:Replace("&", ""))
     Application.Exit()
  RETURN

END CLASS
```

Listing 9: The data-driven menu item class

```
CLASS jhnMenuItem INHERIT jhnMenuItemBase
  CONSTRUCTOR(p AS jhnParameterCollection)
     SUPER()
     SELF:nID := p:GetInt("member_no")
     SELF:nOwnerID := p:GetInt("class_no")
     SELF:Name := p:Get("member_id")
     SELF:Text := p:Get("text")
     IF SELF:nOwnerID > 0
        IF SELF:Name:ToLower():StartsWith("applist")
           SELF:InitializeAppMenuItem()
        ELSE
           SELF:InitializeMenuItem()
        ENDIF
     ENDIF
  RETURN

  METHOD InitializeMenuItem() AS VOID
     LOCAL delMIAdd AS MemAdd
     delMIAdd := MemAdd{SELF, @MenuItemProcess()}
     TRY
        jhnMemberInterface.Inst:MemberAdd(SELF:nID, delMIAdd)
     CATCH ex AS Exception
        MessageBox.Show(ex:Message, SELF:GetType():ToString())
     END TRY
  RETURN

  METHOD InitializeAppMenuItem() AS VOID
     LOCAL delMIAdd AS MemAdd
     delMIAdd := MemAdd{SELF, @MenuItemProcess()}
     TRY
        BEGIN SCOPE
           LOCAL p, oAPC AS jhnParameterCollection
           LOCAL iCls AS INT[]
           LOCAL iCnt AS INT
           iCls := jhnSetupDict.Inst:LkpItemGet("application")
           iCnt := iCls:Length - 1
           FOR LOCAL i := 0 AS INT UPTO iCnt
              p := jhnSetupDict.Inst:ClassPropertyGet(iCls[i])
              oAPC := jhnParameterCollection{}
              oAPC:Add("classmember_no", (i * -1):ToString())
              oAPC:Add("class_no", "0")
              oAPC:Add("seq", i:ToString())
              oAPC:Add("member_no", p:Get("class_no"))
              oAPC:Add("member_id", p:Get("class_id"))
              oAPC:Add("membertype_no", p:Get("classtype_no"))
              oAPC:Add("membertype_id", p:Get("classtype_id"))
              oAPC:Add("text", p:Get("text"))
              oAPC:Add("defaultclass", SELF:GetType():ToString())
              jhnMemberInterface.Inst:MemberAdd(oAPC, delMIAdd)
           NEXT
        END SCOPE
     CATCH ex AS Exception
        MessageBox.Show(ex:Message, SELF:GetType():ToString())
     END TRY
  RETURN

END CLASS
```

Figure 2: The data-driven application framework

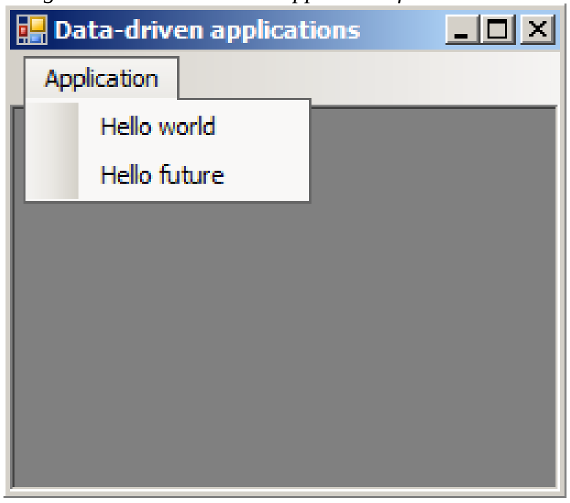


Figure 3: The Hello world application

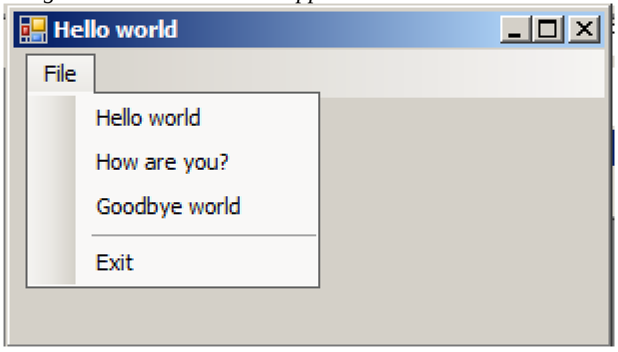
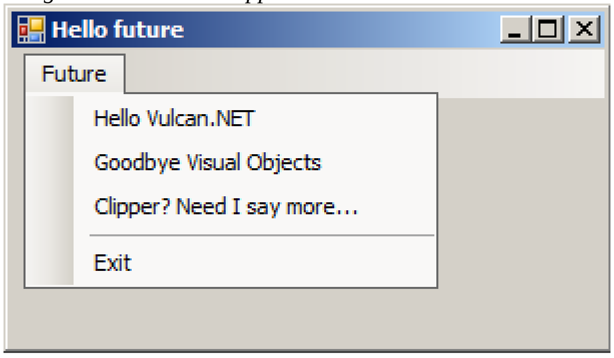


Figure 4: The Future application



5. A happy ending

The intern meets with the IT manager, and the manager informs him that his work has really made an impression. On behalf of the company, he would like to offer him a position inside the company after completion of his studies. If he accepts, the company will pay for all his study expenses, on the condition that he signs a 3-year contract with the company.

6. Summary

We have created a framework for running multiple applications from inside of one application (framework). I know it was done a bit tongue in the cheek, and some of the concepts are quite difficult to describe in words. It will just not do justice for what went into developing the concepts. I suggest the readers use the debugger to step through the code to try and understand the logic.

Till our next article: Presentation of business data – Extending the AppForm. Happy reading till the next article!