HandleView

Developer Preview

0.0.1

User manual

By Alain Guerard  
June 2020

**Content table**

[HandleView 4](#_Toc42443187)

[Knowledge requirements 4](#_Toc42443188)

[Disclaimer 4](#_Toc42443189)

[Basic concepts 5](#_Toc42443190)

[HTML and CSS for UI instead of Access Form 5](#_Toc42443191)

[Single Form Application (SFA) 5](#_Toc42443192)

[Application Host HTML File 5](#_Toc42443193)

[Why the X ? 6](#_Toc42443194)

[Component based 6](#_Toc42443195)

[RouterPort 6](#_Toc42443196)

[Navigation 7](#_Toc42443197)

[Route 7](#_Toc42443198)

[RouterConfiguration 8](#_Toc42443199)

[Interaction between UI and VBA - Responding to HTML DOM events 8](#_Toc42443200)

[What about Access cool tools and features ? 9](#_Toc42443201)

[Boilerplate 11](#_Toc42443202)

[Installation 11](#_Toc42443203)

[Required References 12](#_Toc42443204)

[Getting Started 13](#_Toc42443205)

[Framework Configurations 13](#_Toc42443206)

[The App Form 17](#_Toc42443207)

[Services and Dependency Injection 19](#_Toc42443208)

[Basics 19](#_Toc42443209)

[Configuring your services 20](#_Toc42443210)

[How and where to inject your services 22](#_Toc42443211)

[The bad news about DI 22](#_Toc42443212)

[Using useFactor parameter 23](#_Toc42443213)

[The Host HTML file 24](#_Toc42443214)

[Routes, routing and navigation 27](#_Toc42443215)

[Navigation 27](#_Toc42443216)

[Routing 30](#_Toc42443217)

[xoRoute Class 30](#_Toc42443218)

[Main Route 31](#_Toc42443219)

[Main route with params 31](#_Toc42443220)

[Nested Route 32](#_Toc42443221)

[Sibling Route 32](#_Toc42443222)

[Combine types 33](#_Toc42443223)

[Components 34](#_Toc42443224)

[Receiving parameters at initialization 34](#_Toc42443225)

[Receiving parameters for public method 34](#_Toc42443226)

[Reusable Component properties 34](#_Toc42443227)

[Interpolation 34](#_Toc42443228)

[Submitting Forms 34](#_Toc42443229)

[Using HTML Templates and CSS framework like Bootstrap 34](#_Toc42443230)

[Included XHV Components 34](#_Toc42443231)

[Third Party Components 34](#_Toc42443232)

[Context 34](#_Toc42443233)

[Logging Services 35](#_Toc42443234)

[Debugging 35](#_Toc42443235)

[Error management 35](#_Toc42443236)

# HandleView

HandleView is a UI framework written in VBA (and some javascript modules) for VBA that use the Microsoft Browser component and HTML files to build the user interface instead of using standard VBA Forms.

HandleView is written to be used in Microsoft Access but is easily portable to any Office application that is VBA compatible with minor adjustments.

HandleView can be seen as the glue between your VBA logic code and the Browser. It manages interactions between Browser events, the HTML document and your VBA components. You write your component logic in pure VBA code and the UI in HTML templates.  
Your components can be as simple as a button or complex as whole page.

## Knowledge requirements

To use HandleView without major difficulty , these knowledge requirements are assumed:

* Microsoft Access
* VBA (Obviously😊)
* Basic knowledge of OOP (Object Oriented Programming)
* Basic knowledge of HTML, CSS and JavaScript (The knowledge you’ll need will depend on the complexity of the user interface you want to produce)

## Disclaimer

I’m not a Fulltime Access Developer, neither a VBA specialist. Keep in mind that I developed this code for myself and decided to share it with you, if it can help you, than good 😊

Don’t expect the code to be fully mature and bug free. Error management is kept at a basic level. Also keep in mind that English is not my native language, so some part of this document can be funny to say the least.

*I hope that some of you will join with me in this adventure and help make the framework and the code better for all of us!*

At last, thanks to those who support me on Youtube and make me going on.

Thanks for your understanding!!!

Good coding!

Alain

# Basic concepts

These are the basic concepts behind the framework:

## HTML and CSS for UI instead of Access Form

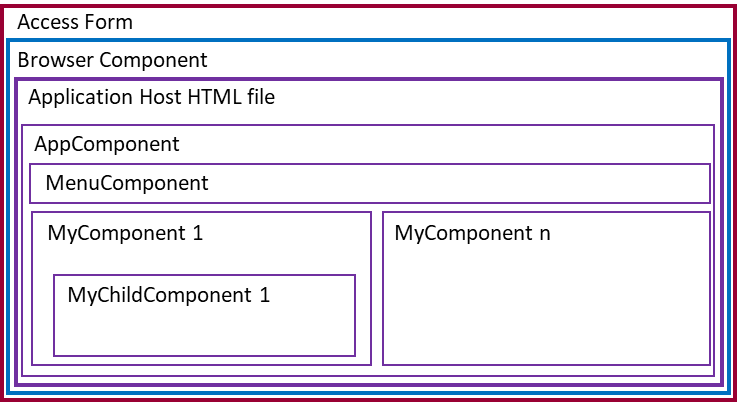
Access Forms are ok to implements desktop application, but even if Microsoft added many features over the time, it is still a bit basic and restrictive when you want to go a little bit further with the user experience.

What if we can make any possible UI like on a web app? Well now we can.

Instead of using Access Forms, HandleView use the Browser Component to render the UI. So you can code your UI in HTML and easily interact with your VBA code.

In reality, an HandleView App use only one Access Form with a Browser component in it. That’s it.

It looks like something like that:



## Single Form Application (SFA)

I like to think of these apps as Single Form Application (SFA), because oubviously, it only need one form to work, but also to remains that is works pretty much like a Single Page Application (SPA) on the Web build with Javascript Framework like Angular, React or VueJs. There is only one HTML page that is loaded at the beginning of the application and after, Handleview manipulate the DOM to change the HTML, render the UI (although the real rendering happens in the browser) and react to DOM events.

## Application Host HTML File

The Host file is the only complete HTML file in an App. It is a regular HTML file with Headers, Body and other HTML file’s code. We will render our application in this HTML file with HandleView by modifying the DOM (Document Object Model). It is also where we will include our CSS files and our JavaScript files if needed. This HTML file acts as our main container for our application.

**IMPORTANT:** Since the Browser Component prevents any JavaScript code to run in a local HTML file, it is imperative that the Host file contains a MOTW (Mark Of The Web) and be saved with the correct UTF format. If you don’t know what it is, just use the Host file that come with the Boilerplate and everything is gonna be fine 😊

## Why the X ?

Every module and classes of the framework are prefix with **xhv.** But why the x ?

For one, it’s pretty cool to prefix everything with an x, x-men, xFactor x-etera….

And second, the framework contains more than 30 classes modules and modules (without any components) and I don’t want them to be cluttered around all my specific apps classes and modules, so they’re all in the bottom and grouped. I could have used z, but it’s not as cool as x.

## Component based

The HandleView framework is based on components. A HandleView application is a composition of many components that act and communicate together.

We write the user interface of our components in HTML and CSS and code our logic in VBA and/or javascript.

Think of a component is a group composed of a HTML file, a VBA class and optional javascript file(s) and css file(s).

xHVComponent

VBA Class + HTML Template + (Optional JS + Optional CSS)

Our application must have a root component. The root component is where we’re gonna put all the other components of our application. By convention, this component is called **appComponent** and lives in the **RouterPort** named **<app-root>** that is inside the Host file.

What is a RouterPort ?

## RouterPort

A **RouterPort** is a HTML container that wrap every direct component – a direct component is the only child of a RouterPort vs child component that lives inside another component.

When HandleView renders a direct component, it automatically renders it inside a RouterPort and keep a Tree structure of all the active components in the actual rendered HTML page. Components nested inside other components are not rendered in a RouterPort, but inside their parent component.

RouterPorts are dynamic “containers” to place components in the UI. We say dynamic because as we navigate in our application, we can add and remove RouterPorts as we need them.

**For now, think of a Port as a container in our app where we can display our components.** We display our components by either Navigate to them of include them in other parent component.

The appComponent is the parent component of all other components.

## Navigation

Every application needs to navigate through different parts. Example, show a list of clients and, when you select a client, show the detail of this client.

**The navigation in HandleView is handle by the xhRouter class** (wich is a Static class) and is easily done by simply calling the **navigate** function of the xhRouter.

Ex:

'Navigate to the client list component

*xhRouter.Navigate "clientsList"*

**What the parameter clientsList represents ?**

It is the name of the route you want to navigate to…

**But how does it knows where to render the clientsList Route ?**

**Hey, what is a Route ?**

Keep reading on…

## Route

**A Route is a navigation command configurated in the application.**

A Route object contains the name of the component you want to display, the RouterPort (container) where to display the component, and any parameters you need to pass to your component for initialization.

Let’s say we have our clientsListComponent that display a list of clients, and when the user click on a client’s name, we want to show the clients details on a side panel.

We would configure a Route with these properties for the details part:

Component:clientDetailComponent (what component we want to show in the panel)  
RouterPort: sidepanel (The name of the panel)  
Parameters: the Id of the client. (Needed to get the info on the client and show its properties)

Instead of repeating these commands everywhere in the app, we configure them at one place in the code.

**But where do I configure Routes ?**

## RouterConfiguration

The **mod\_xhvConfigRouter** module is where we need to configure and edit the Routes. This module contains a single function, *ConfigureRoutes*, that is automatically called when the application start. Here is an example of a Route configuration added to the function:

*Set oRoute = New xhRoute*

*oRoute.path = "app"*

*oRoute.routerPortName = "app-root"*

*oRoute.componentName = "appComponent"*

*ConfigureRoutes.Add oRoute*

This route is the typical base route of any HandleView application and come with the Boilerplate Access Database.

We will see the details about Router’s configuration later… it’s just the Basic concepts for now.

## Interaction between UI and VBA - Responding to HTML DOM events

**How do we trap DOM events ?**

**How do we tell which VBA function to call when we interact with the HTML page in the UI ?**

The interaction between our VBA code and the HTML code is handle by HandleView through event catching and rerouting to VBA code. **Don’t worry, you do not have anything to do here for this to work.**  
  
Magic ? No 😊, in fact pretty simple.

Each HTML template is granted by the framework with a unique *eventDispatcher* element at runtime that automaticaly connect the HTML events with your VBA code via internal GUID.

Now all you have to do is place this little piece of code in your HTML element to react to HTML events in VBA:

xhv-eventlistener xhv-event="click" xhv-eventhandler="ShowMessage" xhv-params=""

Which means, when the user click on this HTML element (this can be a button or any clickable HTML element), call this VBA function (MyVBAFunction) in the component with these parameter values (xhv-params).

Example, you want to show a msgbox when the user pass over a HTML link:

In the HTML :

<a  xhv-eventlistener xhv-event="onmouseover" xhv-eventhandler="ShowMessage" xhv-params=""

Call msgbox  
</a>

In the component VBA code:

Public Sub ShowMessage(ParamArray args() As Variant)

MsgBox args(0)

End Sub

That’s it!

When the user hover the link, HandleView will dispatch the event to the ShowMessage Function of the component and include “Hi” as a string parameters to the function in the args() ParamArray.

Now, what you do in your function is up to you 😊 You want to save a client in your database, do it. You want to show a Popup windows created in JavaScript, call it (with a VBA helper function)**, it’s pretty full VBA from now on** 😊

## What about Access cool tools and features ?

You can keep on using any Access features if you want to. Example, if you want to make reports module and display them from the HTML UI, you can just call them in your VBA components.

Example, instead of showing a MsgBox in the previous ShowMessage function, you can display your Report

Public Sub ShowMessage(ParamArray args() As Variant)

DoCmd.OpenReport "Sales Report", acViewNormal, "Report Filter"

End Sub

You can use Queries, in fact I use a lot of them in the Data Services Classes.

You can call your own Access Forms from function if you want to, although the goal of HandleView is to program the UI in HTML, nothing stop you to mix Regular Forms with the HTML Form.

As I said earlier, once you are in your VBA function in your component, it full VBA, so do what you want in them.

***However, I tend to include all the UI in the HTML Form, I never used any other Forms in my Applications. Even reports are programmed in HTML and when I need more printing support than crappy browser print features, I export them in Excel and/or Word.***

# Boilerplate

**Hopefully , you do not have to code your application from the beginning.**

The Boilerplate you can download on GitHub contains an Access Application FrontEnd DataBase with the HandleView Framework Classes, an Access Backend Database and HTML, Javascript and CSS files you need to get you started.

The boilerplate contains:

* HandleView.DevPreview.0.0.1.accdb
* DemoBackEnd.accdb
* vba folder (which contains the HandleView classes and templates)
* app folder (which contains the HTML files templates, JS files and CSS files to get your started)

***Note****: The Boilerplate database use data from the DemoBackEnd.accdb for some examples.*

*You will need to use the Link Table Manager to point the table Users to your local path of the DemoBackEnd.accdb.*

## Installation

As far as installation is concern, all you need to do is save the files on your computer in a new folder.

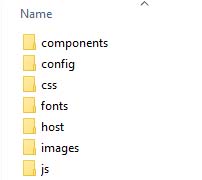
You should have something like that:

Une image contenant capture d’écran

Description générée automatiquement

The vba folder contains the export of the xhClasses, but they are already included in the FrontEnd Database, so you should not need them for now,

The app folder contains the following structure:



* components : will contains all the HTML and the related component JS files.
* config : contains an XML file with HandleView Framework error handling information. You should not modify it.
* css : contains a basic css file and will contain your own css files
* fonts : contains font files, *but at the moment of writing, Browser Component do not allow use of local font file, so don’t bother with this folder for now.*
* host: contains a basic HTML Host File for your application.
* images: well..images 😊
* js: guess what ? Basic JavaScript files for the Framework. You can add your own js files as you need them.

Ok you’re ready !



In the Demo app, I use BootStrap 4 Css Framework to format the HTML. You don’t need to use it and can use any other Css Framework if you want to. There are a lot of good Css frameworks out there, use the one you want or just create your own. You can also use any HTML/Css template available on the web, but you’ll need to tweak and configure them to work in your environment. Also don’t forget that it is the Internet Explorer engine behind the scene, so not always compatible with newer features. But you can always use some javascript polyfill to help…

Just delete it from the css folder if you don’t want to use it.

## Required References

Although HandleView is pure VBA, it requires some commun libraries:

*Microsoft Sripting Runtime  
Microsoft Internet Controls  
Microsoft HTML Library  
Microsoft VBScript Regular Expressions 5.5 or higher  
Microsoft ActiveX Data Objects 2.8 Library or higher*

**These references are already checked in the boilerplate.**

# Getting Started

You should make a copy of the entire Handleview folder to keep an empty Boilerplate for future project.

Let create a new folder and name it MyFirstHandleApp and copy all the files in it.

Lets rename HandleView.DevPreview.0.0.1.accdb to MyFirstHandleApp.accdb.

Ok now open MyFirstHandleApp.accdb Database.

You should have:

* a table named xhvAppConfig
* a table names xhvBackendDB
* a table named xhvConfig
* A form named App
* Couples of Modules
* The “xhv[ClassName]” classes which are the framework classes.
* A c\_appComponent class module
* Couples of other component classes

**Don’t open the App form now, you will get a bunch of errors because you need to configure your app before… but that’s not long, neither complicated…**

## Framework Configurations

The framework needs some informations about where to find the HTML files, where your Backend Database is, etc…

***Note****: You do not have to use a FrontEnd/BackEnd database pattern, but as I always work like that, I included some functionalities to facilitate it. If you do not follow this pattern, you can configure the framework to ignore these functionalities.*

***Note****: The Boilerplate database use data from the DemoBackEnd.accdb for some examples.*

*You will need to use the Link Table Manager to point the table Users to your local path of the DemoBackEnd.accdb.*

These informations are kept in the xhvConfig and xhvAppConfig tables.

So open these tables to edit the configurations to suit your development environment.

*Note: Although the ConfigValue fields are Strings, some value are interpreted as other Type (ex: Boolean)*

Here’s an explanation of the configurations:

### xhvConfig table

|  |  |  |
| --- | --- | --- |
| Configuration Name | Type | Description |
| Environment.RunningEnvironment | String | This tells the framework if you are in your development, test or production environment.  Suggested values are: DEV  TEST  PROD  Handleview do not specifically use this configuration but it can becomes handy when configurating others services such as Logging to configure Logging differently if your App runs in DEV or PROD environment. |
| Environment.ShowDetailFrameworkError | Boolean | Determine if the Exceptions catched in the framework classes should be shown with details or not for the End User.  For future use. Leave it to False for now.  Will tell the framework if a Framework error should show in an HTML page with all the Stack Trace – Perfect for debugging 😊. You’ll be able to not show the error in production. |
| Environment.UseComponentCacheCompressing | Boolean | For future Use. Leave it to False for now.  Will tell the framework if we want the HTML files kept in memory to be compressed. Will lower memory footprint, but for debugging, it will be better to leave it uncompressed so we can easily read the HTML code in the immediate window. |
| Environment.UseComponentsCacheManagement | Boolean | This tells to the framework if it caches the HTML Template file in memory or if it reloads it from the disk everytime a component is created.  I suggest to leave it to False for development, so you can change the HTML file in your editor and don’t have to restart the application to see the changes.  Change it for True in production for better performance. |

### xhvAppConfig table

|  |  |  |
| --- | --- | --- |
| Configuration Name | Type | Description |
| App.BackEndPath | String | The absolute path to your BackEnd Database.  Ex: C:\MyApp\DemoBackEnd.accdb  Don’t mind this setting if you don’t use a BackEnd Database.  Note: The Boilerplate database use data from the DemoBackEnd.accdb for some examples.  You will need to use the Link Table Manager to point the table Users to your local path of the DemoBackEnd.accdb. |
| App.BaseAppFolder | String | The absolute path where your front end database is with no trailing backslash.  Ex: C:\MyApp |
| App.BaseComponentFolder | String | The absolute path of the folder that contains the components folder with a trailing backslash.  Ex: C:\MyApp\app\components\  **We use an absolute path because these files are often kept on a network folder instead of delivering them on the client computer**. |
| App.EnabledLogging | Boolean | This tells the framework if you want to enable logging in the application.  So even if you configure the Logging in your App, you can always toggle it with this configuration value and still keep the configuration and initialisation code in your App |
| App.KeepBackEndOpen | Boolean | If you use a FrontEnd/BackEnd pattern, it tells the framework if we want to keep the BackEnd database link open. It will have a better performance if this link is kept open so Access don’t have to re-create a Record Locking Information file (.laccdb) everytime a connection is made to the BackEnd database.  Don’t mind this setting if you don’t use a BackEnd Database. |
| App.MinimumLogLevel | Integer | Tells the Loggin system what is the minimum level of logging you want to perform.  Allowed values are:   * 0 (for TRACE\_LEVEL) * 1 (for TRACE\_LEVEL) * 2 ( for INFO\_LEVEL) * 3 (for WARNING\_LEVEL) * 4 (for ERROR\_LEVEL) * 5 (for CRITICAL\_LEVEL)   NOTE: These values can be override for any logger in configuration code. |
| App.RootRouterPort | String | This is the name of the RouterPort that is the entry point for your application.  By convention, this RouterPort is named : app-root  This value will be used in the **mod\_xhvStartupApp** module while the application start up. |
| App.StartupHostUrl | String | The absolute path of the App Host HTML file used at the startup.  **The format is important!! If you do not put the = and double quote, Access will bitch about it** 😊  Ex: =”C:\MyApp\app\host\AppHost.html” |
| App.StartupRoute | String | This is the first Route the framework will call when your application will start. This route must be configurated in the **mod\_xhvConfigRouter** module.  Refer to the Router part of this manual for more explanation about Routes and Routes configuration. |

xhvBackendDB table

This table is for future use, when the Backend Database Services support will be improved.

You can ignore it for now.

## The App Form

The App Form is the only form needed in the application. You can add other forms and call them if you want, but all can be done in this App form.

You should not create your own App form, or if you create it, be sure that all properties of your form is identical to the one in the Boilerplate or you can have unexpected behaviors.

The App form is where the Browser component is and contains some simple code to Bootstrap the application. **You may not have anything to add in there if you do not have anything special to do at startup.**

Here’s what happen in the form:

**On Form\_Load,**

*setConst*This Sub calls the setConst Sub in the **mod\_xhvConst** module. This module is the place where we configure “constants” that the framework uses, example DEBUG\_MODE.  
  
*iniForm*We maximized the window and the Browser component, you can change this behavior if you want.   
  
*startupHost WB.Object.Document*  
We Bootstrap the framework with the HTML document of the browser. This is where we configure the Framework sementics used in the HTML files and get the framework configurations from the Access table xhvConfig. **You do not have to do anything here if you use the xhvConfig table for your framework configuration.***startupApp WB*  
This is the place where we will configure our

* Services,
* Logging behaviors,
* Routes.

These configurations are kept in the following modules:

* mod\_xhvConfigServices
* mod\_xhvConfigLogging
* mod\_xhvConfigRouter

So we’ll now look at these 3 modules and explain them

**Note: This is also the place where we open connaction to the BackEnd database if we use one.**

**On Form\_Unload,**

We close the the connection to the **BackEnd database.**

**On Form\_Close,**

We call **End**. The reason we call the End function when the form closes is to be sure that the static classes (those with VB\_PredeclardId = True) are cleared from memory. If we don’t do it, Static classes will stay in memory and the VBA project do not stop. So when we reopen the Form when developing, these classes are already loaded and may contains values from the previous Run, bringing unexpected behaviors, like having a History Cache…

# Services and Dependency Injection

## Basics

Handleview comes with a **very basic** Dependency Injection system, yet it can be very handy. For those who are not familiar with this concept, there are a lot of tutorials on the net about it, but in a nutshell, the Dependency Injection system allow us to “inject” services in our component at runtime, e.g using different concrete classes in our component while only declaring services/dependency as Interface in the component. This way, the component doesn’t have to know how the service accomplish its task and even what will be the class that will accomplish it and only call the functions from the Interface. Lets look at an example:

You have a component that shoes a list of clients.  
In the VBA code of the component, you get the list by filling a recorset object and then adding it to the HTML render of the component.

But. you could have done this:

Add “ClientService” class to your project  
Add “ClientModel” class to your project.   
Code a public function that returns a collection of ClientModel objects.  
Create a ClientService Object in your component, call is ReturnClients Metods and use the collection.  
This way your component doesn’t have to know how the collection was retrieve and can fill it with a text file if you want in your Service.

But now your component is tied with the ClientService.

Still, a better way could have been:

Add a IClientService interface class to your project exposing the ReturnClients method.  
Add a ClientService class that implements the IClientService Interface.  
Add “ClientModel” class to your project.   
  
In the Dependency Injection system **mod\_xhvConfigServices** module you can add the service as a singleton (more on this later):

xhvDI.addSingleton "IClientService", “ClientService”

Now in the component VBA class you declare a

Private Service as IClientService

And in the Class\_Initialize() Sub of the component, you add:

Set Service = xhvDI.inject("IClientService")

Now, here’s what will happen when you instanciate your component:

The framework will create a concrete object of type “ClientService” and keep it in memory in his “container” for future calls because your configure the service as singleton (more on this later) and return it to your component. All your component knows it that it is an implementation of IUserService and use it calling it’s public ReturnClients method.

If later you create a better ClientService (BetterClientService.cls) and want to use it in your application instead of ClientService, you just have to change the configuration in the **mod\_xhvConfigServices** module as:

xhvDI.addSingleton "IClientService", “BetterClientService”

and that’s it. Your component now use the BetterClientService to retrieve it’s list of clients. Maybe from another source DB, etc…

You can also use this technique for testing purpose, configuring your service as a TestClientService that reads data from a static file for testing prupose, and changing configuration later to ClientService when deploying your App.

**Better, you can also pass a function in the configuration settings and change the runtime returning type of the service based on the Configuration(“Environment.RunningEnvironment”) value.**

## Configuring your services

Ok, now you know (or already known) what is a service and what you can do with it, let see how to configure it in the framework.

***You can skip this part if you don’t intend to use services in the Dependency Injection system or services at all.***

The services can be added in 2 modes, singleton or transient.  
  
When you add your service as singleton, the system will reuse the same object over all injection calls throughout the application lifetime. The first time, it will create the object and keep it in its services “container”. All next calls, the system will return the same object, so it doesn’t have to recreate it. Be aware that if you keep data in this object (e.g properties or private variables) these values will live in the service between injection calls and can produce unexpected behaviors if you not expect it. If it’s a behavior you need, that’s ok though 😊

When you add service as transient, the service returned by the system will be a new object each time.

The services configuration is done in the **mod\_xhvConfigServices** module.

You will add your to the system with the Static Class **xhvDI.**

The **xhvDI** class have 3 methods. Here’s how to use them:

|  |  |  |
| --- | --- | --- |
| Method | Parameters | Description |
| addSingleton | **interface as String:** This is the identifier you want to use for your service. It can be anything, but it’s easier to use something like the interface name of this service so you’ll easily identify it role when you come back after 6 month in your code.  **Optional useClass as String:** This is the identifier of the concrete class you want to create when injecting your service in other component. This doesn’t have to be the exact name of the class, but it is more easy to deal with later on. If you do not provide a value for this parameter, you will need to provide one for the **useFactory** one.  **Optional useFactory as String:** This is the identifier of the function you want the Service Factory to use when it create your service. If you do not provide a value for this parameter, you will need to provide one for the **useClass** one. | This method configure a services as a singleton in the system.  If you fill the **useClass** parameter, this is the identifier the system will use to create your service even if you fill the **useFactory** parameter.  If neither of the parameter are filled, the framework will raise an exception because it won’t be able to know which type of object to create. |
| addTransient | Same as addSingleton | This method configure a services as transient in the system, meaning that the system will create a new object for each inject call. |
| inject | **interfaceName as String:** This is the identifier you used in the interface parameter when registering your service with **assSIngleton** or **addTransient** This tells the framework which service you want to inject. | This will create the service you request and return it to the requesting client.  The call to this method should happen in the Class\_Initialize() of the class requesting the service. |

## How and where to inject your services

You can choose where to inject your services in your code, but a good place to inject it in your component is in the Class\_Initialize() Sub of the component. All you have to do is call the xhvDI.inject mthod with the Service identifier as parameter.

Ex:

In your **mod\_xhvConfigServices** module  
xhvDI.addSingleton "IClientService", “ClientService”

In your **component class** module  
Private Service as IClientService

Private Sub Class\_Initialize()

Set base = New xhvBaseComponent

Set props = New Scripting.Dictionary

'Inject dependencies here if needed->

Set UserServices = xhvDI.inject("IClientService ")

this.nameType = COMPONENT\_NAME\_TYPE

this.templateUrl = TEMPLATE\_URL

this.templateString = ""

this.guid = NewGUID()

base.loadTemplate this

End Sub

Don’t bother about the other lines of code in the Class\_Initialize() for now. These lines are part of a component template class.

## The bad news about DI

Real Dependency Injection Framework use what we call Reflection to instanciate dependencies and analyze the parameters of the constructor’s class to know which dependencies to inject.

In VBA, we have neither of it. Class Constructor , nope, Reflection , nope. Analyzing the code through VBE, not without special rights your end users won’t have so not practical..

We will have to rely on a factory pattern and manual injection of services. For the manual injection, you’ve seen how to do it in the previous Section (How and where to inject your services), so now the really bad part, you will have to add your identifiers in a Select Case in the **xhvServiceFactory** class, also if you use the **useFactory** property, that is where you will code the function you identify with this parameter. (See next section)

That’s bad, but not complicated.

Just adjust your select case like this:

Public Function getServiceFromClassName(ClassName As String) As Object

Select Case ClassName

Case "xhvDBConfigService"

Set getServiceFromClassName = New s\_xhvDBConfigService

Case "UsersService"

Set getServiceFromClassName = New s\_UserServices

Case "BetterUserService"

Set getServiceFromClassName = New s\_BetterUserService

Case “myIdentifier”

Set getServiceFromClassName = New s\_myService

End Select

End Function

## Using useFactor parameter

If you used the useFactory parameter to create the object with a function,you must code your function in the **xhvServiceFactory** module. This function must be Public because it uses CallByName to execute the function..

For the same reason, the name of the function me be exactly the identifier you used when you registered the service with the xhvDI class.

This function must return as type Object, as the framework don’t know at runtime what is the class it needs to return.

# The Host HTML file

The Host file is the only complete HTML file in an App. It is a regular HTML file with Headers, Body and other HTML file’s code. We will render our application in this HTML file with HandleView by modifying the DOM (Document Object Model). It is also where we will include our CSS files and our JavaScript files if needed. This HTML file acts as our main container for our application.

**IMPORTANT:** Since the Browser Component prevents any JavaScript code to run in a local HTML file, it is imperative that the Host file contains a MOTW (Mark Of The Web) and be saved with the correct UTF format. If you don’t know what it is, just use the Host file that come with the Boilerplate and everything is gonna be fine 😊

The Host HTML file is located in the ../app/host folder.

Here is the code in this file, and we’ll take a look at the most significant lines.

<!-- saved from url=(0014)about:internet -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />

    <meta http-equiv="X-UA-Compatible" content="IE=11" />

    <link href="../css/style.css" rel="stylesheet" type="text/css">

</head>

<body class="vh-100 d-flex flex-column" style="border:1px solid #f00">

    <xhv-routerport name="app-root" class="d-flex vh-100 flex-column">

    </xhv-routerport>

    <!-- xoView Required JavaScript -->

    <script src="../js/xo/jquery-3.5.0.js"></script>

    <script src="../js/jquery-ui.js"></script>

    <script src="../js/handleview/xhv.required-0.0.2.js"></script>

    <!-- xoView Helpers Function JavaScript -->

    <script src="../js/xo/xo-helpers-1.0.0.js"></script>

    <script src="../js/xo/xo-init.js"></script>

    <!-- Bootstrap 4 JavaScript -->

    <script src="../js/bootstrap/bootstrap.min.js"></script>

</body>

</html>

<!-- saved from url=(0014)about:internet -->

This is the MOTW (Mark Of The Web) that we need to let the Browser execute JavaScript in a local HTML file.

<!DOCTYPE html>

HTML 5 markdown.

<meta http-equiv="X-UA-Compatible" content="IE=11" />

This line tells the Browser to treat the HTML as if we were using IE11. IE11 mode provides the highest support available for established and emerging industry standards, including the HTML5, CSS3 and others

    <link href="../css/style.css" rel="stylesheet" type="text/css">

Link our principal css file. The link is relative to the HTML file. So in this case the style.css file is located in the ..app/css folder.

You can add as many css files you want, but a simpler way to work is to link your other css file in the style.css file and make your style.css file act as a main hub for these other css files.

    <xhv-routerport name="app-root" class="d-flex vh-100 flex-column">

    </xhv-routerport>

Ok, so here is our RouterPort. That’s where the Framework will render our first Route when Bootstraping the application.

So when we bootstrap the application in the mod\_xhvStartupApp. startupApp() Sub, the Framework will Navigate to the “app” (Configuration("App.StartupRoute")) Route and this Route is configurated to display the appComponent in the “app-root” RouterPort. So the appComponent will render itself inside this element.

The class vh-100 is just a Bootstrap 4 class to make the page 100% view height. Css classes depends on the way you want your application UI to look like.

    <script src="../js/xo/jquery-3.5.0.js"></script>

    <script src="../js/jquery-ui.js"></script>

    <script src="../js/handleview/xhv.required-0.0.2.js"></script>

    <!-- xoView Helpers Function JavaScript -->

    <script src="../js/xo/xo-helpers-1.0.0.js"></script>

    <script src="../js/xo/xo-init.js"></script>

    <!-- Bootstrap 4 JavaScript -->

    <script src="../js/bootstrap/bootstrap.min.js"></script>

These are the links to JavaScript files used by the Framework. Handleview JavaScipt file uses JQuery library, so you need to include this library at the top. (jquery-3.5.0.js).

Xhv.required-0.0.1 is the mandatory JavaScript file need by the Framework to dispatch the HTML events to the VBA code.

xo-helpers-1.0.0.js contains some helper functions to facilitate the VBA code. These are not mandatory. You can remove it if you don’t need these simple functions. Yoiiu can also edit this file and make your own js function as needed.

xo-init.js contains some initilizations for the application. You can edit this file and add your own code depending on what you want to use in your application. For example, if you use other JavaScript libraries and need to initialize some of them, you can do it in this file.

This is not mandatory to use this file, if you want to initialize your js libraries somewhere else, it doesn’t affect Handleview in any way.

**You can also add any other JavaScript file or library you need in your application.**

**Please note that is is your responsibility to manage your JavaScript code and libraries, the Handleview Framework do not have any responsibilities except to include and load your files in memory.**

Ok, now we go on with the main course, the Routes and the Components!

# Routes, routing and navigation

**From the Basic Concepts :**

## Route

## **A Route is a navigation command configurated in the application.**

## A Route object contains the name of the component you want to display, the RouterPort (container) where to display the component, and any parameters you need to pass to your component for initialization.

## Let’s say we have our clientsListComponent that display a list of clients, and when the user click on a client’s name, we want to show the clients details on a side panel.

## We would configure a Route with these properties for the details part:

## Component:clientDetailComponent (what component we want to show in the panel)

## RouterPort: sidepanel (The name of the panel)

## Parameters: the Id of the client. (Needed to get the info on the client and show its properties)

## Instead of repeating these commands everywhere in the app, we configure them at one place in the code.

## But where do I configure Routes ?

## RouterConfiguration

## The mod\_xhvConfigRouter module is where we need to configure and edit the Routes. This module contains a single function, ConfigureRoutes, that is automatically called when the application start. Here is an example of a Route configuration added to the function:

## Set oRoute = New xhRoute

## oRoute.path = "app"

## oRoute.routerPortName = "app-root"

## oRoute.componentName = "appComponent"

## ConfigureRoutes.Add oRoute

## This route is the typical base route of any HandleView application and come with the Boilerplate Access Database.

## We will see the details about Router’s configuration later… it’s just the Basic concepts for now.

## Navigation

The navigation is, well, how we navigate in the application. Everytime you want to show a new “window”, you navigate to this “window”. In Handleview we do not have windows, instead we display component in our HTML page. Remember a component can be group of child components. Your appComponent will probably be the main component inside your app-root RouterPort. So inside your app-root RouterPort, you will display your appComponent with this Route: “app” that we configure with this code snippet inside the mod\_xhvConfigRouter module:

## Set oRoute = New xhRoute

## oRoute.path = "app"

## oRoute.routerPortName = "app-root"

## oRoute.componentName = "appComponent"

## ConfigureRoutes.Add oRoute

By calling this line in the mod\_xhvStartupApp. startupApp() Sub:

xhvRouter.navigate Configuration("App.StartupRoute")

This line will ask the framework to display the appComponent in the app-root RouterPort.

**Now, any time you want to display a component, you will call the navigate method of xhvRouter in your code with the appropriate Route path your previously configure.**

# Routing

**Routes are case sensitive**

## xoRoute Class

The xoRoute Class represents a route that we configure in the application.

### Properties

|  |  |  |
| --- | --- | --- |
| Properties | Type | Description |
| componentName | String | The name of the component to render when navigating to this route. This component will be render a a sibling node of the routerPort specified in the routerPortName property. |
| exitGate | String | **For future use.**  Will contain the name of the function to run to check if we can exit the route (ex: check if there is something to save before leaving) |
| guid | String | Unique identifier of the route.  Should be ReadOnly, but write is needed for cloning reason. |
| matchPattern | String | The pattern of the route to compare with a requested path. Dynamically calculed by the framework. ReadOnly. |
| nestedRoutes | Collection | Collection of nestedRoute. Used for nested route navigation when a component have a routerPort in it.  ReadOnly. |
| parentRoute | xoRoute | A reference to the parentRoute when the route is a nested route. |
| path | String | The path to match a requested path when navigating. If the path matches, this route is used for the *navigate* function of the router. |
| routerPortName | String | The name of the routerPort the requested route (component) will render. |
| securityGate | String | **For future use.**  Will contain the name of the function to run to check if the user can activate this route. |

### Methods

|  |  |  |
| --- | --- | --- |
| Properties | Return | Description |
| clone | xoRoute | Return a clone of the route. This function is needed by the framework internally. Don’t mind it. |

## Main Route

routeName

### Scenario:

Show a component in a specific port.

**Note:** All the components within the specified component of the the route will render, but if we have nested RouterPort inside the components, we will have to configure NestedRoute for the nested RouterPort to render their components.

Initialize in XoRouterConfiguration class

…  
oRoute.routerPortName = "app-root"

oRoute.componentName = "appComponent"

oRoute.path = "app"

Usage in code  
….  
xoRouter.navigate “app ”

## Main route with params

RouteName/Param1;Param2;…ParamX

### Scenario:

Same as simple route but with parameters passed to the component, example an id to show a specific model object.

Initialize in XoRouterConfiguration class

…  
oRoute.routerPortName = "content"

oRoute.componentName = "clientComponent"

oRoute.path = "client/?id?isActive"

Usage in code  
….  
Dim sParam as string  
sParam = “{id:123;isActive:true}”  
xoRouter.navigate “client”, sParam

## Nested Route

A nested route is a route that will render inside a routerPort defined inside a component that is rendered when navigating. If no nested route is configured for the nested routerPort, the routerPort will be available for future navigation purpose.

### Scenario:

We want to display a homepage with a main menu at the top that will be there all the time, with a content port in the middle that will show different “forms” when clicking on the main menu. We display the menu as component inside the page and add a routerPort named “content” in the middle form the forms.

Initialize in XoRouterConfiguration class

…  
oRoute.routerPortName = "mainport"

oRoute.componentName = "mainComponent"

oRoute.path = "mainpage"

Set oNestedRoute = New xoRoute

oNestedRoute.routerPortName = "content"

oNestedRoute.componentName = " formComponent1"

oNestedRoute.path = "form1"

Set oNestedRoute.parentRoute = oRoute

oRoute.nestedRoutes.Add oNestedRoute, oNestedRoute.guid

Usages in code  
….  
xoRouter.navigate “mainport//home”

xoRouter.navigate “mainport/123//home/123” (Nested routes with parameters)

## Sibling Route

A sibling route is a route that will render after the main route in a routerPort that is not a nested routerPort of the main route but a sibling one.

### Scenario:

We want to display a homepage with a main menu at the top that will change over time, with a content port in the middle that will show different “forms” when clicking on the main menu. We create a component that act as a container for the 2 needed routerPorts: “menu” and “content”. We want to update the menu component and the “form” in the “content” routerPort at the same time.

Be aware, that those routerPorts must exist in the DOM before you navigate to them, so they must be declared in a higher component.

Initialize in XoRouterConfiguration class

…  
oRoute.routerPortName = "menu"

oRoute.componentName = "menuComponent"

oRoute.path = "menu"

…  
oRoute.routerPortName = "content"

oRoute.componentName = "formComponent1"

oRoute.path = "form1"

Usage in code  
….  
xoRouter.navigate “menu||form1”

xoRouter.navigate “menu/123||form1” (Sibling routes with params)

## Combine types

You can combine different scenarios. Example, xoRouter.navigate “route1||route2/123//childroute2/123” will navigate the route1 and the route2 with parameter’s value of 123 with the nested routes “childroute2” with parameter’s value of 123.

# Components

So , I have to write a component, where do I begin ?

## Receiving parameters at initialization

Parameters are received int the init function in a dictionary. Order don’t matter

They come from the navigating route like home:/{‘id’:1}

## Receiving parameters for public method

Parameters are received from the eventDispatcher and are in a array. Order is important!! They are not named parameters.

## Reusable Component properties

When a component have property that need to be initialize with the value of attributes in HTML, these values are passed in the xhvIComponent\_onAfterInit function.

Theyare in a dictionary, so order don’t matter.

## Interpolation

To use interpolation**, properties must be placed in the props dictionary** of the component.

## Submitting Forms

## Using HTML Templates and CSS framework like Bootstrap

# Included XHV Components

# Third Party Components

# Context

* A Context is used to adapt behaviors of a component depending of the context it is used in.
* A Context is set by parent component.
* A Context is not a global store, it is only a string containing the chain of ancestors component
* Always provide a defaultBehavior for component, in case it is not in a context or not in a context it expects.

# Logging Services

Explanation about how to use the Logging Services will come in a later version of the document.

# Debugging

# Error management

* If you only have Rethrow in error handling, then there is no need for it as the calling function will handle it anyway.
* Error handled by HandleFrameworkError should never happen in the Application when deployed, there are error that means that something is wrong when developing and debugging. These are not errors that should happen in user environment. I means that is those error happen, your environment is not set adequately.
* You should not use this error handling strategie in your own code.