IMPLEMENTING FORM CALLS AND CALLBACKS

How-To Guide

www.foxincloud.com

BEHIND THE SCENE

Form instantiation and modal responses are key issues in adapting an application to FoxInCloud.

The purpose of this document is to explain the underlying logic of this process and present practical advice and examples on how developers should adapt their forms and form calls to FoxInCloud.

FoxInCloud handles form instances for you

As a standard behavior, FoxInCloud holds a single instance of each form to serve any request of any user, and produces HTML, CSS and javascript code to ensure a Web mode behavior similar to LAN operations.

To achieve this, FoxInCloud Application Server needs to get a grasp on the form, in other words keep an internal reference to each form instance.

Wherever your code instantiates a form, either by `DO FORM` or `CreateObject('myFormClass')`, it should be replaced by a call to the standard `wForm()` method that every form inherits from their `aw.vcx!awFrm` ancestor class after FoxInCloud Adaptation Assistant's step # 2-Adapt.

We'll explain later in this document what parameters `wForm()` expects, how FoxInCloud server instantiates the form, runs its `Init()` method and maintains a reference to it.

What's special about modal states

When running an application in LAN mode, each instance of the application serves a single user;
whenever application expects a decision from user, it simply can stop and wait until user has made his mind, then resume execution.

Behind the scene…

In web mode such a process can't happen just because application server is not dedicated to a single user, but serves requests from any user, just like a waiter in a bar serves all clients whereas a home maid serves a single person.

Another valid metaphor is a phone conversation: imagine you're on the phone and ask your correspondent a question; if you're doing this single task, you can wait until he or she answers; but if you have another task on hand, you will probably ask him/her to `call back` when he/she has the answer.

The same concept applies to modal response with a FoxInCloud server: when calling a modal form from which your application expects an answer, your code will indicate to the called form the name and/or address of the method it should 'callback' when user has made his decision.

In FoxInCloud, each visual object has a set of standard methods dedicated to this purpose, named **`.wFormCallBack?()`**[1]. These methods accept a single parameter where FoxInCloud can feed the value chosen by user, whatever the type of this value: string, boolean, number, date, object, etc.

Of course, this construct does not affect the way application operates in LAN mode: execution is suspended as usual.

What about modal responses from a form class?

In Visual FoxPro, getting a modal response from a form **`class`** (as opposed to **`DO FORM myForm.scx … TO`**) is tricky because the **`show()`** method does not return the value **`RETURN`**ed by the **`unload()`** method.

[1] This name is by pure convention; developer may use any method as long as it accepts one parameter where FoxInCloud can inject user's choice.

Behind the scene…

Usually, when instantiating a modal form class, developer codes:

```
PRIVATE puResponse
puResponse = <some Void Value>
o = createObject('formClass', parm1, parm2, …)
o.show(1)
<code processing puResponse>
```

in some event method of the called form:

```
thisForm.PropertyHoldingUserChoice = <whatever value user chooses>
```

finally, in **`unload()`** method:

```
puResponse = thisForm.PropertyHoldingUserChoice
RETURN
```

The good news is … FoxInCloud handles this tedious process for you!

You just need to set, in the called form, the **`.wcModalChoiceProp`** property to the name of the form's property holding user's choice; that would be '**`PropertyHoldingUserChoice`**' in the preceding example.

You can also store user's choice into the standard **`.wuValue`** property inherited from FoxInCloud's **`<VFP9>\Tools\ab\aw.vcx!awFrm`**.

Processing response from modal form

As mentioned earlier, in Web mode, program execution continues after form call; so, if code was kept the same, the instructions processing user's response, usually located right after modal form call, would get executed.

To avoid that, you need to move this code to the 'callback' method indicated when calling the form. We'll see examples of that later in this document.

Behind the scene…

How is the form instantiated?

FoxInCloud instantiates your form without parameters, then executes the form’s `Init()` method with the parameters passed.

If form is already instantiated when another user requires it, FoxInCloud just re-executes the form’s `Init()` method with the parameters passed, after adjusting form’s dataSessionID if it works in default DataSession

This implies that your forms are able to instantiate without any parameter passed to their `Init()` method, in other word, no `Error` or `RETURN .F.` should occur under such circumstances.

If ever some controls in your form are bound to an alias produced by calling form (say `‘parentAlias’`), a recommended practice is to code in the form’s `load()` method:

```
IF NOT Used(‘parentAlias’)
  CREATE CURSOR parentAlias … && parentAlias is a Cursor
&& Or
  USE parentAlias NODATA NOREQUERY && parentAlias is a View
&& Or
  USE parentAlias AGAIN SHARED && parentAlias is a Table
ENDIF
```

CALLING A TOP-LEVEL FORM

In a desktop application, top-level forms are usually called from a menu or some kind of `_Screen` form. No change is required in the way you call top-level forms in your LAN application, with code such as:

```
DO FORM myForm.scx [WITH parm1, parm2, …] [TO userResponse]
  or
loForm = CreateObject('myFormClass' [, parm1, parm2, …])
loForm.show()
```

In web mode though, your top-level forms should take place in a HTML page displayed in the browser.

FoxInCloud supports 2 ways of building such HTML pages including the HTML, CSS and JavaScript produced by FoxInCloud from your application forms:

Through 'traditional' method such as ASP, PHP…

You add the HTML, CSS and JavaScript produced by FoxInCloud from your form to the page by some **`'server-side include'`** directive, such as, in ASP: **`<!-- #INCLUDE FILE="myForm.htm" -->`**.

The drawback of this method is that top-level forms need be instantiated at server startup in order to have HTML CSS and JS available, thus their **`Init()`** method runs without parameter.

You can instruct FoxInCloud server to instantiate forms at server start-up by setting the **`'uFormsLaunchAtStartup'`** property in you server sub-class:

```
* ===========================================
DEFINE CLASS myServer AS awServer OF awServer.prg && FiC server class
…
uFormsLaunchAtStartup = 'myForm.scx'
…
ENDDEFINE && class myServer
* ===========================================
```

Calling a top-level form...

With FoxInCloud/wConnect 'Process' class/object.

If you're not familiar with the West-Wind wConnect framework, you just need to know that the 'Process' class[2] is where HTTP requests are processed with VFP code.

Based on the URL component to method name mapping scheme described in the wConnect documentation (http://www.west-wind.com/webconnection/docs/), each request on the wConnect server results in executing a given method of the corresponding Process object.

To build a HTML page including your form's HTML, CSS and JS, you just need to create methods in your awProcess sub-class named like your forms; for instance, if your form is named 'myForm', you can create a method called 'myForm' in your awProcess sub-class.

FoxInCloud's awProcess class includes a convenient standard method **wFormStandardPage()** doing all the work for you:

```
PROCEDURE myForm
this.wFormStandardPage('myForm.scx' [, parm1[, parm2...[, parmN]]])
```

awProcess::wFormStandardPage():

1. Instantiates **'myForm.scx'** if not already,
2. Executes **myForm.scx::Init()** with the parameters provided,
3. Generates HTML, CSS and JavaScript from your VFP live form,
4. Builds a complete page including your form's HTML, CSS and JS,
5. Sends back the page as a response to the client browser.

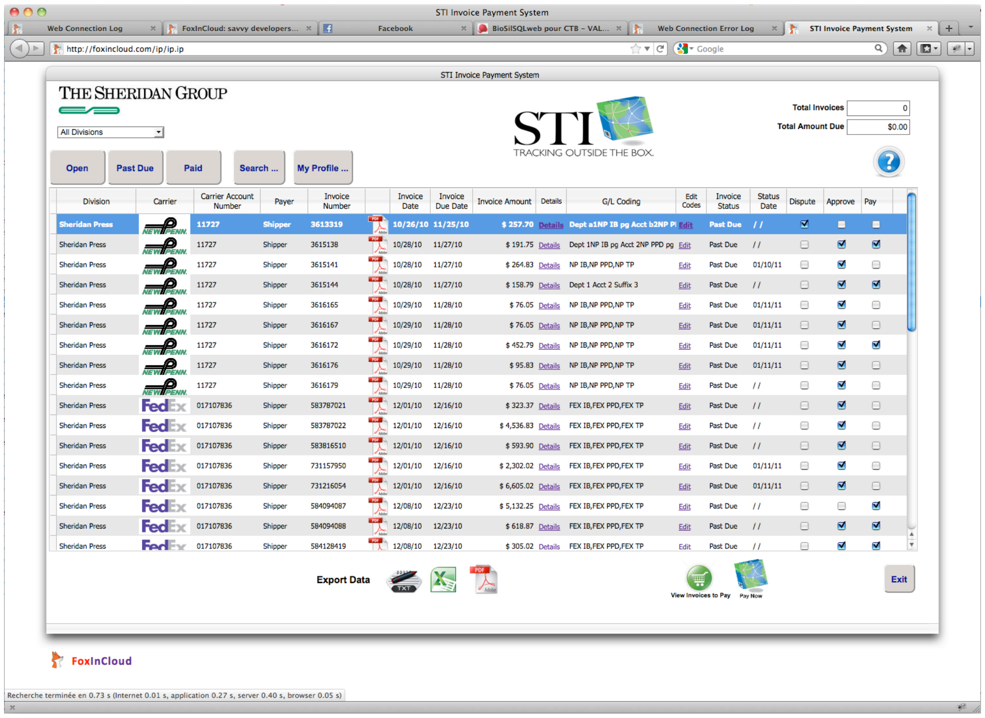
[2] Process class is named 'wwProcess' in wConnect, 'awProcess' is FoxInCloud's sub-class of 'wwProcess'.

Calling a top-level form…

With this method you could create a sub-class of awProcess like:

```
DEFINE CLASS myProcess AS awProcess OF awServer.prg
*-------------------------
PROCEDURE identification
this.wFormStandardPage('identification.scx', 'someParm')
EXTERNAL FORM identification.scx && instructs project manager to add form
*-------------------------
PROCEDURE orderEntry
this.wFormStandardPage('orderEntry.scx', 0)
EXTERNAL FORM orderEntry.scx && instructs project manager to add form
*-------------------------
* etc.
ENDDEFINE && CLASS myProcess
```

This method is preferable as it speeds up Server startup (no form to instantiate) and allows passing parameters to your Top Level Forms.


STI Invoice Payment System
Web Connection Log
FoxInCloud: savvy developers…
Facebook
BioSilSQLweb pour CTB – VAL…
Web Connection Error Log
STI Invoice Payment System
http://foxincloud.com/ip/ip.ip
Google
STI Invoice Payment System
THE SHERIDAN GROUP
All Divisions
Open
Past Due
Paid
Search …
My Profile …
STI
TRACKING OUTSIDE THE BOX.
Export Data
TXT
View Invoices to Pay
Pay Now
Exit
FoxInCloud

Total Invoices 0

Total Amount Due $0.00

Division	Carrier	Carrier Account Number	Payer	Invoice Number		Invoice Date	Invoice Due Date	Invoice Amount	Details	G/L Coding	Edit Codes	Invoice Status	Status Date	Dispute	Approve	Pay
Sheridan Press	NEW PENN.	11727	Shipper	3613319	PDF	10/26/10	11/25/10	$ 257.70	Details	Dept a1NP IB pg Acct b2NP P	Edit	Past Due	/ /	☑	☐	☐
Sheridan Press	NEW PENN.	11727	Shipper	3615138	PDF	10/28/10	11/27/10	$ 191.75	Details	Dept 1NP IB pg Acct 2NP PPD pg	Edit	Past Due	/ /	☐	☑	☑
Sheridan Press	NEW PENN.	11727	Shipper	3615141	PDF	10/28/10	11/27/10	$ 264.83	Details	NP IB,NP PPD,NP TP	Edit	Past Due	01/10/11	☐	☑	☐
Sheridan Press	NEW PENN.	11727	Shipper	3615144	PDF	10/28/10	11/27/10	$ 158.79	Details	Dept 1 Acct 2 Suffix 3	Edit	Past Due	/ /	☐	☑	☑
Sheridan Press	NEW PENN.	11727	Shipper	3616165	PDF	10/29/10	11/28/10	$ 76.05	Details	NP IB,NP PPD,NP TP	Edit	Past Due	01/11/11	☐	☑	☐
Sheridan Press	NEW PENN.	11727	Shipper	3616167	PDF	10/29/10	11/28/10	$ 76.05	Details	NP IB,NP PPD,NP TP	Edit	Past Due	01/11/11	☐	☑	☐
Sheridan Press	NEW PENN.	11727	Shipper	3616172	PDF	10/29/10	11/28/10	$ 452.79	Details	NP IB,NP PPD,NP TP	Edit	Past Due	01/11/11	☐	☑	☑
Sheridan Press	NEW PENN.	11727	Shipper	3616176	PDF	10/29/10	11/28/10	$ 95.83	Details	NP IB,NP PPD,NP TP	Edit	Past Due	01/11/11	☐	☑	☐
Sheridan Press	NEW PENN.	11727	Shipper	3616179	PDF	10/29/10	11/28/10	$ 76.05	Details	NP IB,NP PPD,NP TP	Edit	Past Due	/ /	☐	☑	☐
Sheridan Press	FedEx	017107836	Shipper	583787021	PDF	12/01/10	12/16/10	$ 323.37	Details	FEX IB,FEX PPD,FEX TP	Edit	Past Due	/ /	☐	☑	☐
Sheridan Press	FedEx	017107836	Shipper	583787022	PDF	12/01/10	12/16/10	$ 4,536.83	Details	FEX IB,FEX PPD,FEX TP	Edit	Past Due	/ /	☐	☑	☐
Sheridan Press	FedEx	017107836	Shipper	583816510	PDF	12/01/10	12/16/10	$ 593.90	Details	FEX IB,FEX PPD,FEX TP	Edit	Past Due	/ /	☐	☑	☐
Sheridan Press	FedEx	017107836	Shipper	731157950	PDF	12/01/10	12/16/10	$ 2,302.02	Details	FEX IB,FEX PPD,FEX TP	Edit	Past Due	01/11/11	☐	☑	☐
Sheridan Press	FedEx	017107836	Shipper	731216054	PDF	12/01/10	12/16/10	$ 6,605.02	Details	FEX IB,FEX PPD,FEX TP	Edit	Past Due	01/11/11	☐	☑	☐
Sheridan Press	FedEx	017107836	Shipper	584094087	PDF	12/08/10	12/23/10	$ 5,132.25	Details	FEX IB,FEX PPD,FEX TP	Edit	Past Due	/ /	☐	☐	☑
Sheridan Press	FedEx	017107836	Shipper	584094088	PDF	12/08/10	12/23/10	$ 618.87	Details	FEX IB,FEX PPD,FEX TP	Edit	Past Due	/ /	☐	☑	☑
Sheridan Press	FedEx	017107836	Shipper	584128419	PDF	12/08/10	12/23/10	$ 305.02	Details	FEX IB,FEX PPD,FEX TP	Edit	Past Due	/ /	☐	☑	☑

Recherche terminée en 0.73 s (Internet 0.01 s, application 0.27 s, server 0.40 s, browser 0.05 s)

Figure 1 - Sample page built with `awProcess::wFormStandardPage()`

CALLING A CHILD FORM

As instructed by Adaptation Assistant, calling a child form with FoxInCloud requires you replace your child form call instruction such as …

```
DO FORM myForm.scx [WITH parm1, parm2, …] [TO userResponse]
  or
loForm = CreateObject('myFormClass' [, parm1, parm2, …])
loForm.show()
```

… by a call to the standard `thisForm.wForm()` method that every form inherits from the `<VFP9>\Tools\ab\aw.vcx!awFrm` ancestor class after the 'automated adaptation' (FoxInCloud Adaptation Assistant step #2):

```
thisForm.wForm(;
  'Form.scx' or 'FormClass';
, CallBack;
, @m.parm1, @m.parm2, …
)
```

parameter 1: 'Form.scx' or 'FormClass':

This is self-explanatory: just pass a string indicating the name of the .scx file or the class of the form you want to launch; of course, the .scx file should be in the current `PATH` and the class library where form class is defined should be in either `Set('PROCEDURE')` or `Set('CLASSLIBRARY')`.

Parameter 2: CallBack:

This parameter can take several values:

– Passing an `Empty()` value such as `''` or `.F.` means the form is modeless (non modal) and your code expects no response from it;

– Passing `.T.` means the child form should be shown as modal (user can only interact with this form), but your code expects no response from it;

Calling a child form …

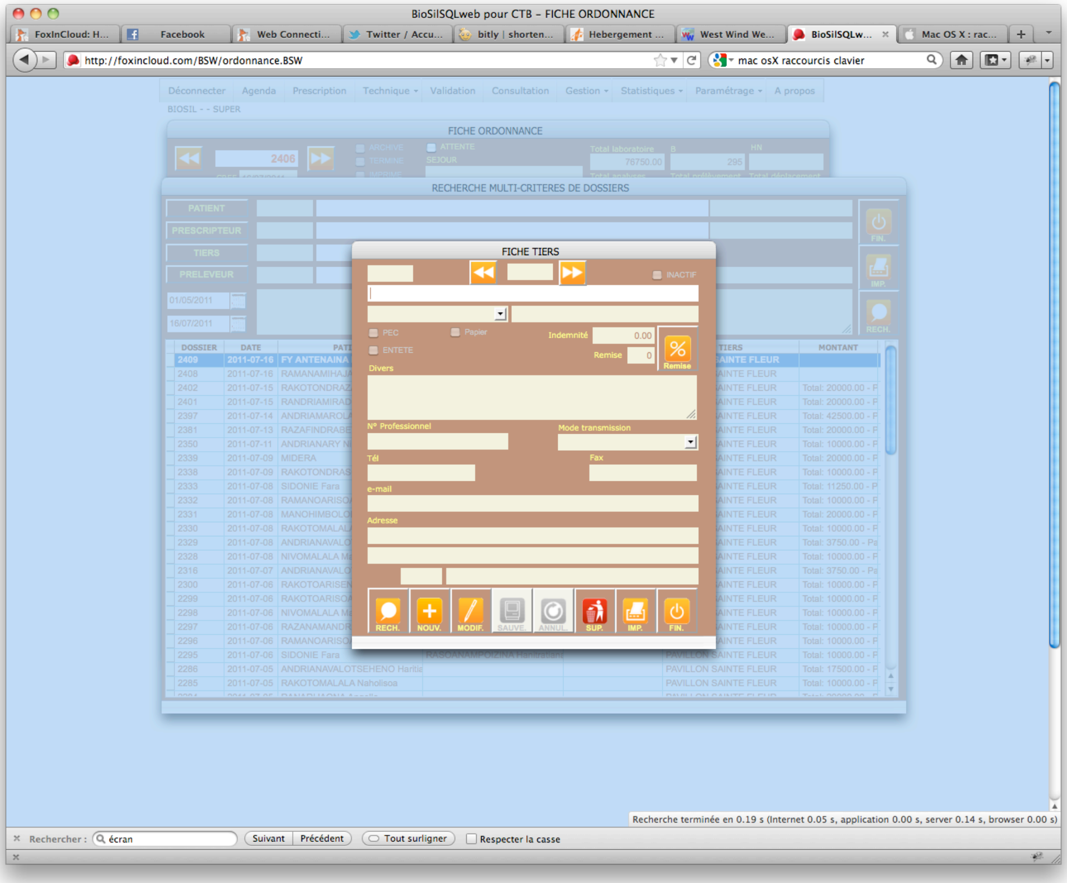
- Passing a `NOT Empty()` character string indicates the name of the 'callback' method into which user's choice should be injected. This value can be expressed in an either **relative** or **absolute** format:
  - **In current object**: this is the most current and object-oriented syntax, meaning the callback code lies in the same object that calls the form. It's also the easiest to implement as callback code can be moved within the same object or class. Typical values are `'wFormCallBack'` … to … `'wFormCallBack5',` methods that every object inherit from its `<VFP9>\Tools\ab\aw.vcx` parent class after Automated Adaptations by FoxInCloud Adaptation Assistant step 2.
  - **Relative to current object**: you can specify an address such as `'this.Parent.[someObject.]someMethod'`
  - **Absolute to the Form**: if you prefer a centralized method to process callbacks from several objects, you can pass `'thisForm.[someObject.]someMethod'`

`Other parameters: @m.parm1, @m.parm2`:

These are the parameters you want to pass to the form's `init()` method as you usually do in your code. In this example we pass parameters by reference (`@m.parm?`) just to illustrate that FoxInCloud passes parameters by reference all along the call chain so that arrays are supported.

On next page is a sample of sub-forms called from a top-level form.

Calling a child form ...



01/05/2011
16/07/2011

DOSSIER	DATE	PATI	TIERS	MONTANT
2409	2011-07-16	FY ANTENAINA	AINTE FLEUR	
2408	2011-07-16	RAMANAMIHAJA	AINTE FLEUR	
2402	2011-07-15	RAKOTONDRAZ	AINTE FLEUR	Total: 20000.00 - P
2401	2011-07-15	RANDRIAMIRAD	AINTE FLEUR	Total: 20000.00 - P
2397	2011-07-14	ANDRIAMAROLA	AINTE FLEUR	Total: 42500.00 - P
2381	2011-07-13	RAZAFINDRABE	AINTE FLEUR	Total: 20000.00 - P
2350	2011-07-11	ANDRIANARY N	AINTE FLEUR	Total: 10000.00 - P
2339	2011-07-09	MIDERA	AINTE FLEUR	Total: 20000.00 - P
2338	2011-07-09	RAKOTONDRAS	AINTE FLEUR	Total: 10000.00 - P
2333	2011-07-08	SIDONIE Fara	AINTE FLEUR	Total: 11250.00 - P
2332	2011-07-08	RAMANOARISOA	AINTE FLEUR	Total: 10000.00 - P
2331	2011-07-08	MANOHIMBOLO	AINTE FLEUR	Total: 20000.00 - P
2330	2011-07-08	RAKOTOMALALA	AINTE FLEUR	Total: 10000.00 - P
2329	2011-07-08	ANDRIANAVALO	AINTE FLEUR	Total: 3750.00 - Pa
2328	2011-07-08	NIVOMALALA Ma	AINTE FLEUR	Total: 10000.00 - P
2316	2011-07-07	ANDRIANAVALO	AINTE FLEUR	Total: 3750.00 - Pa
2300	2011-07-06	RAKOTOARISEN	AINTE FLEUR	Total: 10000.00 - P
2299	2011-07-06	RAKOTOARISOA	AINTE FLEUR	Total: 10000.00 - P
2298	2011-07-06	NIVOMALALA Ma	AINTE FLEUR	Total: 10000.00 - P
2297	2011-07-06	RAZANAMANDR	AINTE FLEUR	Total: 10000.00 - P
2296	2011-07-06	RAMANOARISOA	AINTE FLEUR	Total: 10000.00 - P
2295	2011-07-06	SIDONIE Fara	PAVILLON SAINTE FLEUR	Total: 10000.00 - P
2286	2011-07-05	ANDRIANAVALOTSEHENO Harilia	PAVILLON SAINTE FLEUR	Total: 17500.00 - P
2285	2011-07-05	RAKOTOMALALA Naholisoa	PAVILLON SAINTE FLEUR	Total: 10000.00 - P

Recherche terminée en 0.19 s (Internet 0.05 s, application 0.00 s, server 0.14 s, browser 0.00 s)

Figure 2 - 2 levels of modal sub-forms called from a top-level form

CALLING A MESSAGEBOX

You probably noticed that the 'automated adaptation' (step #2 of FoxInCloud Adaptation Assistant) has replaced calls to **`MessageBox()`** located within a form by a call to **`thisForm.wMessageBox().`**

`thisForm.wMessageBox()` is a wrapper around **`thisForm.wForm()`** which automatically passes the MessageBox form class as first parameter. The MessageBox form class is indicated in a property that each form inherits from **`<VFP9>\Tools\ab\aw.vcx!awFrm`**: '**`.wcMessageBoxClass'`**. By default **`.wcMessageBoxClass`** holds the name of your sub-class of **`<VFP9>\Tools\ab\aw.vcx!awFrmMB`**, typically **`yourApp.vcx!yourAppFrmMB`** created by automated adaptation, where '**`yourApp`**' is the 2-3 letter prefix you chose during automated adaptations setup step #3. You can also choose another subclass of yours if you see fit.

`thisform.wMessageBox()` syntax is as follows:

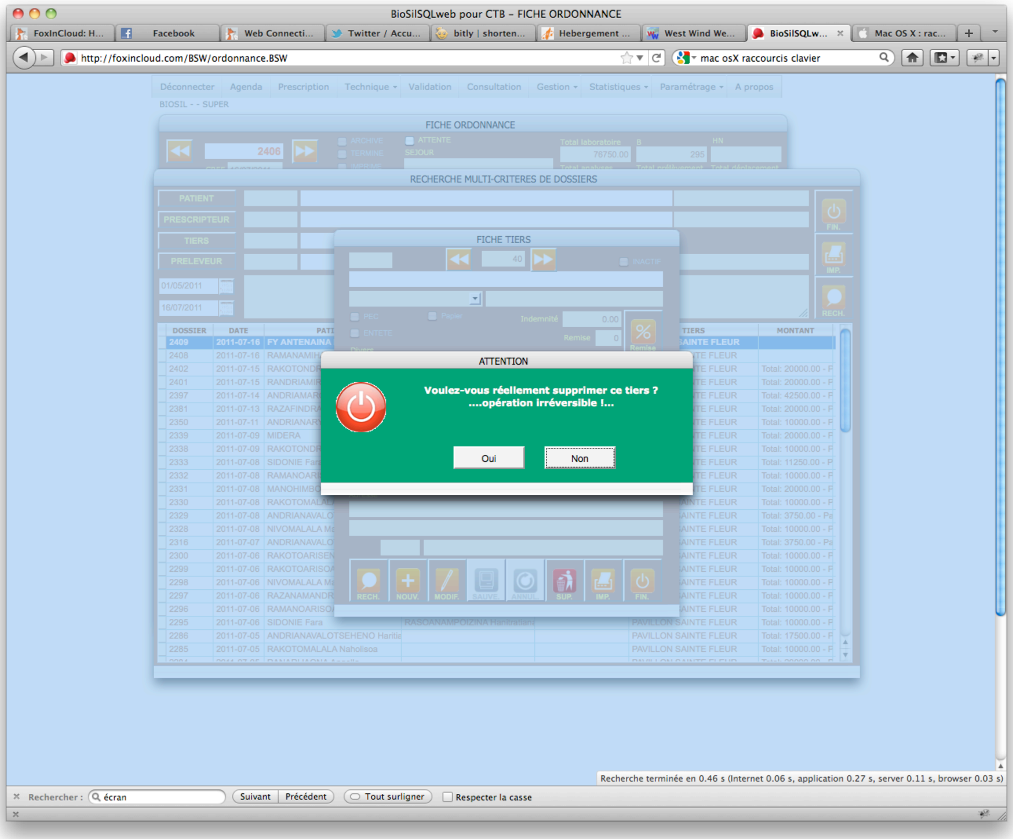
```
thisForm.wMessageBox (;
, CallBack;
, eMessageText [, nDialogBoxType ][, cTitleBarText][, nTimeout])
)
```

`parameter CallBack` obeys the same rules as described earlier for **`thisForm.wForm().`**

`Parameters eMessageText [, nDialogBoxType ][, cTitleBarText][, nTimeout]` are the standard VFP messageBox() parameters preserved by automated adaptations. Please see VFP help for more details about these parameters.

On next page is a sample of message box called by a level-2 sub-form of a top-level form.

Calling a messageBox …


BioSilSQLweb pour CTB – FICHE ORDONNANCE
http://foxincloud.com/BSW/ordonnance.BSW
FICHE ORDONNANCE
RECHERCHE MULTI-CRITERES DE DOSSIERS
FICHE TIERS
ATTENTION
Oui
Non

Voulez-vous réellement supprimer ce tiers ?
....opération irréversible !...

Recherche terminée en 0.46 s (Internet 0.06 s, application 0.27 s, server 0.11 s, browser 0.03 s)

Figure 3 - Top level form
calling a modal form
calling another modal form
calling a standard messageBox()

IMPLEMENTING A CALLBACK

Let's consider this typical event method in an object called 'btn':

```
* btn.Click()
<some code>
LOCAL luUserChoice
DO FORM question.scx WITH someParm TO luUserChoice
IF m.luUserChoice = <some Value>
  <some code processing someValue>
ELSE
  <some code processing other values>
ENDIF
```

In this example, **`question.scx`** is a modal form which **`RETURN`**s user's choice into the **`luUserChoice`** variable.

This code could/should be adapted as follows:

```
* btn.Click()
<some code>
thisForm.wForm('question.scx', 'wFormCallBack', @m.someParm)
```

Response code is then moved to the **`wFormCallBack()`** method of the same object 'btn':

```
* btn.wFormCallBack()
LPARAMETERS luUserChoice && FoxInCloud will pass user's choice here
IF m.luUserChoice = someValue
  <some code processing someValue>
ELSE
  <some code processing other values>
ENDIF
```

Note that, in this example, the standard FoxInCloud parameter **`'tuUserChoice'`** of method **`wFormCallBack()`** was renamed to **`'luUserChoice'`** to easily fit existing code.

Fox InCloud

www.foxincloud.com ı contact@foxincloud.com