New UI Classes From Carlos Alloatti

Doug Hennig

Carlos Alloatti is a prolific developer of classes that can make your application's user interface sparkle. This month, Doug looks at a couple of new libraries Carlos created that can replace the VFP Toolbar and provide a tabbed document interface to your applications.

Carlos Alloatti specializes in developing libraries VFP developers can use to make the user interface of their applications more attractive, modern-looking, and easier to use. His web site, http://www.ctl32.com, has downloads and documentation for many libraries I use and I'm sure you'll find useful. I discussed many of these in the July and September 2011 and January 2012 issues of FoxRockX ("The ctl32 Library, Parts 1, 2 and 3").

Carlos recently released two new libraries: TBZ and TDI. TBZ provides easy-to-use buttons that replace the VFP Toolbar. TDI provides a tabbed document interface to forms in an application. I'll discuss both of these libraries in this article.

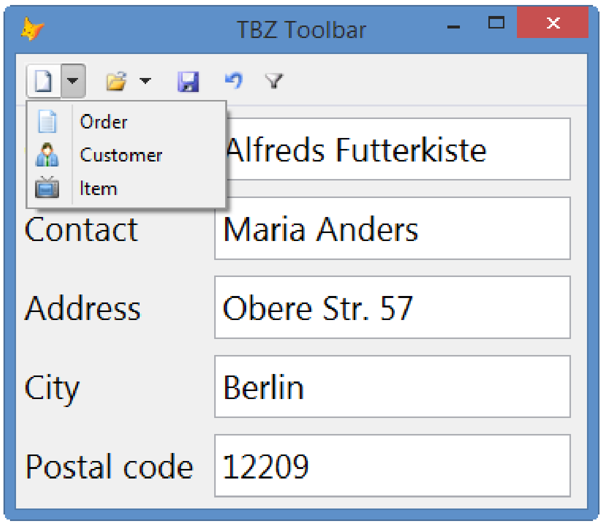
Toolbar buttons

VFP developers often use a toolbar instead of adding buttons directly to a form because controls in a toolbar don't get focus. However, toolbars aren't the easiest controls to work with:

- The user can move and close a toolbar, which often leads to support calls like "how do I get the toolbar back." They can even undock the toolbar with an inadvertent double-click.
- You can't create a toolbar instance; you have to create a separate class even if only one instance of the toolbar is used and it's never subclassed.
- Toolbars are odd controls to work with at design time. When you drop a control on one, VFP adds it to the left edge of the toolbar regardless of where you actually dropped the control. Separators are weird to work with too; I always have to reposition them a couple of times to get them to the correct place.
- You can't visually add a toolbar to a form; doing so creates a formset. Programmatically adding one to a form is odd too: you have to do it someplace like Activate rather than Init or it won't attach to the form.
- Toolbars take up some of the form height but you can't see that at design time so you have to pretend the toolbar is there when sizing the form and placing controls.

Carlos' TBZ library, available for download from http://www.ctl32.com/tbz/tbz.html, provides controls you add to a VFP form rather than a toolbar. Because they act like normal VFP controls at design time and don't receive focus at runtime, you avoid the issues discussed above while getting the benefits of a toolbar.

One other benefit you get from TBZ is that in additional to command buttons (the _TBCommand class), checkboxes (_TBCheck and _TBToggle for a graphical version), and radio buttons (_TBRadio), Carlos provides drop-down (_TBDropDown and _TBDropDownVert) and split button (_TBSplitButton and _TBSplitButtonVert) controls. These are buttons that include shortcut menus; in the case of a split button, you can either click the button to take some action or the down arrow beside it to display the menu. See **Figure 1** for an example.



Alfreds Futterkiste

Contact	Maria Anders
Address	Obere Str. 57
City	Berlin
Postal code	12209

Figure 1. The TBZ split button provides both a command button and a shortcut menu.

Using TBZ

Start by adding _TBZ.VCX to your project. When you build the project, 74 PRGs, all starting with "_api," are automatically added. These PRGs use a rather ingenious technique: each declares a Windows API function with an alias the same as the PRG name and then calls that function. The benefit of this approach is that there's no need to declare functions before they're used. For example, when the Windows API CreateFont function is needed, call _APICreateFont. The first time it's called, the PRG is executed, which declares CreateFont as _APICreateFont and then executes it. The second time _APICreateFont is used, the Windows API function is called instead of the PRG because of the order in which VFP looks for names.

In addition to the PRGs, eight PNGs, _TBZ01.PNG through _TBZ08.PNG, are automatically added to the project. These PNGs contain images used for the various styles of the controls.

To create a toolbar in a form, drop instances of TBZ classes onto the form. For example, to create the sample TBZToolbar.SCX included with the downloads for this article, I dropped two _TBSplitButton, two _TBCommand, and a _TBToggle onto the form and positioned them accordingly. I also dropped a _TBLine on the form to act as a separator line for the toolbar. The interesting thing about _TBLine is that it sizes itself automatically to the width of the form so you don't have to worry about sizing it, setting Anchor to handle resizing, etc.

After adding the controls to the form, add code to the Click method of each. For _TBDropDown and _TBSplitButton, add code to the DropDownClick method to define and display a shortcut menu and take the necessary action. **Listing 1** shows the code that displays the menu shown in Figure 1. In a real application, the MESSAGEBOX statement would be replaced with the appropriate code.

Listing 1. This code displays a shortcut menu for the first split button in the form.

```
local lnResult
This.tbMenuItemAdd(1, 0, 0, 0, 0, ;
    '&Order', 'invoice.png')
This.tbMenuItemAdd(2, 0, 0, 0, 0, ;
    '&Customer', 'customer.png')
This.tbMenuItemAdd(3, 0, 0, 0, 0, ;
    '&Item', 'item.png')
lnResult = This.tbMenuShow()
do case
    case lnResult = 1
        messagebox('Create new order')
    case lnResult = 2
        messagebox('Create new customer')
    case lnResult = 3
        messagebox('Create new item')
endcase
```

Add items to the shortcut menu by calling the tbMenuItemAdd method of the button and display the menu by calling tbMenuShow. The parameters for tbMenuItemAdd are:

- The ID of the item. This is the value returned by tbMenuShow when the user selects an item.
- The parent ID of the item. To create a submenu, specify the ID of the item this item should be under. **Figure 2** shows an example of a submenu.
- The type of menu item. You can use constants defined in _TBZ.H for the values. The choices are 0 for a normal item, 1 for a separator, 2 to display a button as a checkmark, and 3 if the item contains a submenu.
- 1 if the item is checked (or displays a button if the third parameter is 2) or 0 if not.
- 0 if the item is enabled or 1 if not.
- The caption of the item. Use & to specify that the next character is the hot key for the item.
- An image file to use as a picture for the item.

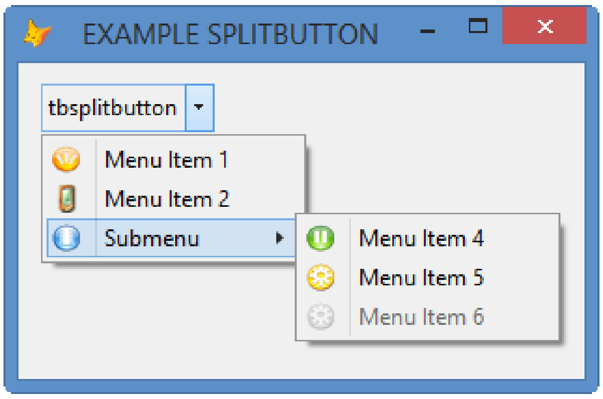


Figure 2. Submenus are easy to create using tbMenuItemAdd.

You may also want to set properties of the controls. There are a lot of properties that control the appearance of the controls, all documented on the TBZ web page, but the main ones are listed in **Table 1**.

Although it's just a sample form and doesn't do much, TBZToolbar.SCX shows one use for split buttons: allowing the user to select one of several choices. For example, clicking New (the leftmost button) would add a new customer but clicking the drop-down part of the button allows the user to select what to create a new record of: Order, Customer, or Item. Similarly, clicking Open (the second button in the toolbar) would display a dialog of orders for the current customer but clicking the drop-down displays the most recent orders for the customer.

Table 1. The most common properties of the TBZ controls.

Property	Description
Picture	The image to use for the button.
Caption	The caption for the button.
PicturePosition	Determines where the picture goes; only applicable when Caption isn't blank.
Alignment	Specifies the alignment of the caption.
ToolTipText	The tooltip to display for the control. TBZ uses a custom tooltip window rather than the VFP window.
tbAlign	Controls how automatic positioning works: 0 means no automatic positioning, 1 (the default) positions controls starting at the left edge of the form, and 2 positions them starting at the right edge.
tbGroupID	Radio controls with the same value act like a group.
tbMarginH	The space between the left or right edge of the form and the first control when tbAlign is 1 or 2.
tbMarginV	The space between the top or bottom of the form and the controls when tbAlign is 1 or 2.
tbStyle	Specifies which of the eight PNG files provides the visual style for the control; see the TBZ web page for screen shots of the various styles.
tbValue	The value of the radio button or checkbox; use this instead of Value.

TBZ issues

There are a couple of things I couldn't get to work at first. One was that neither drop-down nor split buttons would display a shortcut menu. The cause turned out to be a case-sensitivity issue in the _GetMouseStatus method of _TB, the parent class for the TBZ controls. Replace the commented-out line in that method with the one following it:

```
*** If  m.laMouse[1].Name = 'UISHAPER' Then
    If  upper(m.laMouse[1].Name) = 'UISHAPER'
```

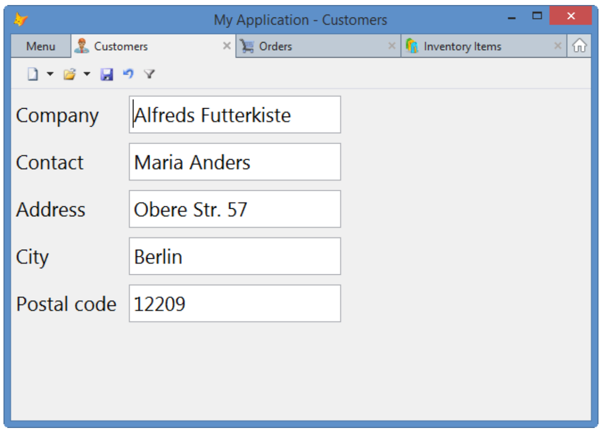
The second wasn't really a problem, just a difference in the way the controls worked from the documentation. The TBZ web page states that the controls are automatically laid out at runtime, similar to how a VFP Toolbar works. However, I found I had to manually position the controls where I wanted them to appear because they weren't automatically positioned. The cause turned out to be simple: the TBZ controls use BINDEVENT to bind their _ArrangeControls method, which does the positioning, to the Init method of the form, and my sample form didn't have code in Init, which caused the event binding to fail. Adding a comment to Init took care of that issue.

How it works

Carlos has a good description of how TBZ works on the TBZ web page. He includes information on how to create your own style PNG files if you want to use a different theme as well as GDI+ drawing issues he ran into and how he solved them.

Tabbed document interface

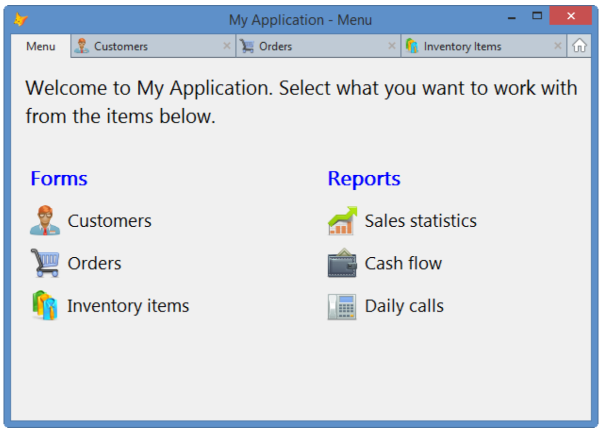
Some users live in their Internet browsers. They're used to a web page just being another tab within their browser window. Carlos' TDI library, available for download from http://www.ctl32.com/tdi/tdi.html, allows you to create a similar user interface for your application: each open form is just another tab within a main window. **Figure 3** shows an example, with three forms open as tabs.



Company	Alfreds Futterkiste
Contact	Maria Anders
Address	Obere Str. 57
City	Berlin
Postal code	12209

Figure 3. The TDI library makes it easy to create a tabbed document interface.

Typically, the first tab in such an interface is a menu of available forms or some type of home page for the application, such as that shown in **Figure 4**. When the user opens a form, it appears as a new tab. The tab may display an icon and includes an "X" to close the tab. The home button at the right edge of the main window selects the first tab. As more tabs are opened, the tab width automatically narrows as necessary to fit all of the open tabs in the window. Tabs can be rearranged by dragging them to the desired tab position. The first tab is special in that it doesn't display an icon and can't be closed or rearranged.



Welcome to My Application. Select what you want to work with from the items below.

Forms	Reports
Customers	Sales statistics
Orders	Cash flow
Inventory items	Daily calls

Figure 4. The first tab is usually a menu or home page.

Using TDI

Start by adding _TDI.VCX to your project. When you build your project, 29 PRGs, all starting with "_api" and using the same technique the PRGs used by TBZ do, are automatically added. Only 10 of the PRGS are the same as those in TBZ so if you use both libraries in an application, you'll have 93 of these PRGS in your project.

The next step is to create the main window for the application. This form hosts the forms in your application rather than _SCREEN so it'll be a top-level form. To hide _SCREEN, create a CONFIG.FPW file containing at least SCREEN=OFF (you'll likely want other lines as well, such as RESOURCE=OFF) and add it to the Text Files section of your project. Create a form and set ShowWindow to 2-As Top-Level form. Drop a TDIMain object on the form and add code to its TDIInit method to run another form, the one that'll display the content for the first tab. For example, Main.SCX (shown in **Figure 5**), the form that acts as the main window in the samples for this article, just has one line of code in TDIInit:

```
do form Main
```

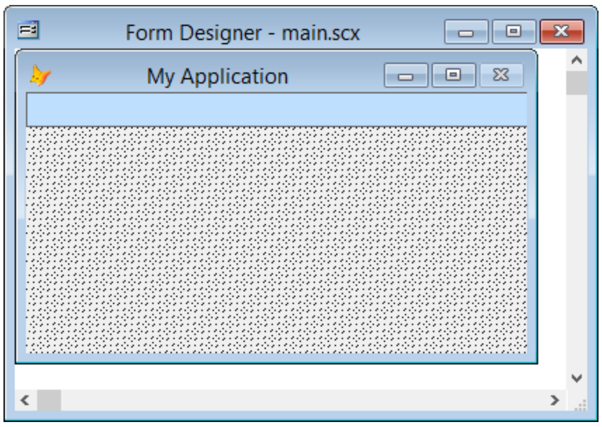


Figure 5. The form for the main window just consists of an instance of TDIMain.

There are only a few properties of TDIMain you may wish to change:

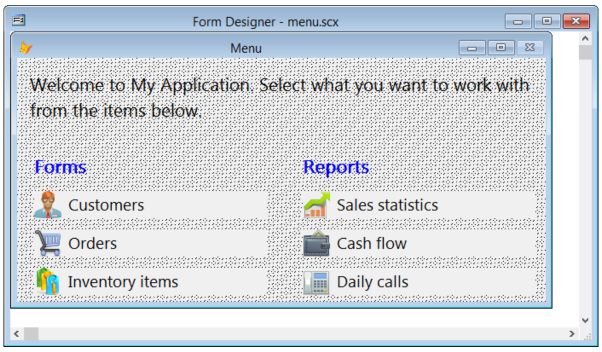
- TabFirstWidth: the width of the first tab. If you leave this at the default of 0, the tab is automatically sized.

- TabMaxWidth: the maximum width of a tab. The default is 200.

- TDICenterDialogs: set this to .T. (the default) to center system dialog windows, such as MESSAGEBOX(), on the form or .F. to center them on the Windows desktop.

- TDIGoToFirstTabOnClose: set this to .T. to automatically select the first tab when a tab is closed or .F. (the default) to select the next logical tab.

Now create a form that contains the contents of the first tab. Set ShowWindow to 1-In Top-Level Form and Caption to the text displayed on the first tab. Add whatever controls to the form you wish. In the case of Menu.SCX (shown in **Figure 6**), which is the form used for the first tab in the samples for this article, I added some labels and some instances of a custom class consisting of an image and label to act as big buttons (the "Customers," "Orders," and other buttons shown in Figure 6). The Click method of each "button" runs another form using DO FORM *SomeFormName* or displays a message box.



Welcome to My Application. Select what you want to work with from the items below.

Figure 6. The form for the main tab usually contains buttons to launch other forms.

Finally, create the other forms used by the application, being sure to set ShowWindow to 1-In Top-Level Form, Caption to the text displayed on the tab, and Icon to the desired icon for the tab. One thing you'll find is that you might want to move the controls a little to the right and down from the top from where you might normally put them. For example, I usually start controls 10 pixels from the left and 10 from the top but that puts them too close to the edges when used with TDI. I suggest starting them at least 15 pixels from the left and top edges.

To display the main window for the application, use DO FORM *MainWindowName* or instantiate and call the Show method of the main window class you created.

How it works

Carlos has a brief description of how TDI does its magic on the TDI web page. Basically, one of his classes monitors VFP window creation and takes over how the form is displayed inside the top-level form.

Summary

Carlos Alloatti has created a couple of cool new libraries to add new features to your application's user interface. The TBZ library allows you to replace the VFP Toolbar with something that has all of the benefits of toolbars but none of the pitfalls. The TDI library provides a tabbed document interface for your application. Not all applications will benefit from this type of interface but it's certainly an interesting concept for form-centric applications.

Doug Hennig is a partner with Stonefield Software Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT); the award-winning Stonefield Query; the MemberData Editor, Anchor Editor, and CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro; and the My namespace and updated Upsizing Wizard in Sedna.

Doug is co-author of "Making Sense of Sedna and SP2," the "What's New in Visual FoxPro" series (the latest being "What's New in Nine"), "Visual FoxPro Best Practices For The Next Ten Years," and "The Hacker's Guide to Visual FoxPro 7.0." He was the technical editor of "The Hacker's Guide to Visual FoxPro 6.0" and "The Fundamentals." All of these books are from Hentzenwerke Publishing (http://www.hentzenwerke.com). He wrote over 100 articles in 10 years for FoxTalk and has written numerous articles in FoxPro Advisor, Advisor Guide to Visual FoxPro, and CoDe. He currently writes for FoxRockX (http://www.foxrockx.com).

Doug spoke at every Microsoft FoxPro Developers Conference (DevCon) starting in 1997 and at user groups and developer conferences all over the world. He is one of the organizers of the annual Southwest Fox conference (http://www.swfox.net). He is one of the administrators for the VFPX VFP community extensions Web site (http://vfpx.codeplex.com). He was a Microsoft Most Valuable Professional (MVP) from 1996 to 2011. Doug was awarded the 2006 FoxPro Community Lifetime Achievement Award (http://tinyurl.com/ygnk73h).