Extending the Toolbox

The Toolbox was designed to allow various kinds of extensions. Among other things, you can add your own item types and menu items.

Tamar E. Granor, Ph.D.

Several issues back, I wrote about the Toolbox, a cool tool added in VFP 8 that makes designing forms and classes much easier. Like many other VFP tools, the Toolbox was written with VFP and was designed with extension in mind. This month, I want to show you how simple some changes to the Toolbox are, so you can customize it for your needs.

I'll start with a look at the Toolbox's infrastructure; not surprisingly, it uses some tables. Then, I'll look at two kinds of changes you can make without touching the core Toolbox code.

Under the hood of the Toolbox

The Toolbox has five components: user interface, engine, tool classes, data and metadata. The user interface, the engine and the tool classes are written in VFP. You have the source code. If you haven't already done so, unzip XSource.ZIP in the Tools\XSource folder of your VFP installation. After you do so, you'll find the Toolbox code in the VFPSource\Toolbox folder.

The Toolbox engine code is in ToolboxEngine.PRG. If you take a look at this code, you'll find a variety of methods, some corresponding to the various actions you can take in the Toolbox, others more fundamental.

The Toolbox UI code is in a number of classes. The main Toolbox form is defined by the ToolboxForm class in the FoxToolbox library. The same classlib contains classes for many other UI elements, as does toolboxctrls.vcx.

The Toolbox sees the world in terms of categories and tools; everything in the Toolbox is one or the other. Each type of category or tool is implemented by a class. All of the implementation classes are based on the _root class, contained in _toolbox.vcx. The _root class contains methods that provide the basic functionality for any item in the Toolbox; some of the methods are abstract here, others call on the Toolbox engine, while others work with a collection of properties for the particular tool.

The same class library contains many subclasses of _root. Some are abstract, like _root, while others implement a particular tool type. The File tool type is actually implemented by a number of different classes, representing different kinds of files.

The _toolbox.vcx class library is excluded from the Toolbox project. A copy is installed in the Toolbox folder of your VFP installation. Excluding it makes it possible to make changes and additions without having to rebuild the Toolbox application.

Later in this article, I'll show you how to define a new tool type by creating a new subclass of _root.

The Toolbox uses two tables. The first is Toolbox.DBF, stored in the VFP folder of the Documents and Settings hierarchy (the folder referenced by HOME(7)). This table contains one record for each tool in the Toolbox, that is, one for each category and one for each item. Figure 1 shows part of my Toolbox table; these entries are among those supplied with the tool.

Toolbox

Uniqueid	Showtype	Tooltypeid	Tooltype	Parentid	Toolname
microsoft.favorites	F	CATEGORY.FAVORITES	Memo		Favorites
microsoft.textscraps	C	CATEGORY.TEXTSCRAPS	Memo		Text Scraps
microsoft.foundation	C	CATEGORY.GENERAL	Memo		My Base Classes
microsoft.webservice	C	CATEGORY.WEBSERVICE	Memo		My XML Web Serv
microsoft.baseclasses	C	CATEGORY.GENERAL	Memo		VFP Base Classes
baseclass.checkbox	T	BASECLASS	Memo	microsoft.baseclasses	Check Box
baseclass.collection	T	BASECLASS	Memo	microsoft.baseclasses	Collection
baseclass.gridcolumn	T	BASECLASS	Memo	microsoft.baseclasses	Grid Column
baseclass.combobox	T	BASECLASS	Memo	microsoft.baseclasses	Combo Box
baseclass.commandgroup	T	BASECLASS	Memo	microsoft.baseclasses	Command Group
baseclass.container	T	BASECLASS	Memo	microsoft.baseclasses	Container
baseclass.control	T	BASECLASS	Memo	microsoft.baseclasses	Control
baseclass.cursor	T	BASECLASS	Memo	microsoft.baseclasses	Cursor
baseclass.custom	T	BASECLASS	Memo	microsoft.baseclasses	Custom
baseclass.dataenvironment	T	BASECLASS	Memo	microsoft.baseclasses	Data Environmen
baseclass.editbox	T	BASECLASS	Memo	microsoft.baseclasses	Edit Box
baseclass.exception	T	BASECLASS	Memo	microsoft.baseclasses	Exception
baseclass.form	T	BASECLASS	Memo	microsoft.baseclasses	Form
baseclass.grid	T	BASECLASS	Memo	microsoft.baseclasses	Grid
baseclass.columnheader	T	BASECLASS	Memo	microsoft.baseclasses	Column Header
baseclass.hyperlink	T	BASECLASS	Memo	microsoft.baseclasses	HyperLink
baseclass.image	T	BASECLASS	Memo	microsoft.baseclasses	Image

Figure 1. The Toolbox table contains one record for each thing in the Toolbox, whether category or item.

The Toolbox table also contains data for filters you create and for add-ins, which include custom menu items. I'll show you how to create custom menu items later in this article.

Tooltype

Uniqueid	Showtype	Tooltype	Classname	Classlib	Fi
CATEGORY.GENERAL	C	General category	Memo	memo	m
CATEGORY.FOLDER	C	Dynamic folder category	Memo	memo	m
CATEGORY.ACTIVEX	C	Registered ActiveX controls	Memo	memo	m
CLASS	T	Class	Memo	memo	m
FILE	T	File	Memo	memo	m
SCRIPT	T	Script	Memo	memo	m
ACTIVEX	T	ActiveX control	Mem	emo	m
TEXTSCRAP	T	Text scrap	Memo	memo	m
SCX	T	Form	Memo	memo	M
PRG	T	Program	Memo	memo	M
DBF	T	Table	Memo	memo	M
DBC	T	Database	Memo	memo	M
FRX	T	Report	Memo	memo	M
LBX	T	Label	Memo	memo	M
IMAGE	T	Image	Memo	memo	M
APP	T	Application	Memo	memo	M
BASECLASS	T	Base class	Memo	memo	m

_scripttool

Figure 2. The ToolType table maps tools to the subclasses of _root that implement them.

The other table is ToolType.DBF, stored in the Toolbox folder of your VFP installation; it maps tools to the classes that implement them. Each record represents one type or category, tool or file. For example, Figure 2 shows that the _SCRIPT tool is implemented by the _ScriptTool class.

To modify Toolbox behavior, you can make changes to any of the components. What's great is that you can make a number of changes without having to touch the UI or engine code. The rest of this article looks at two such changes.

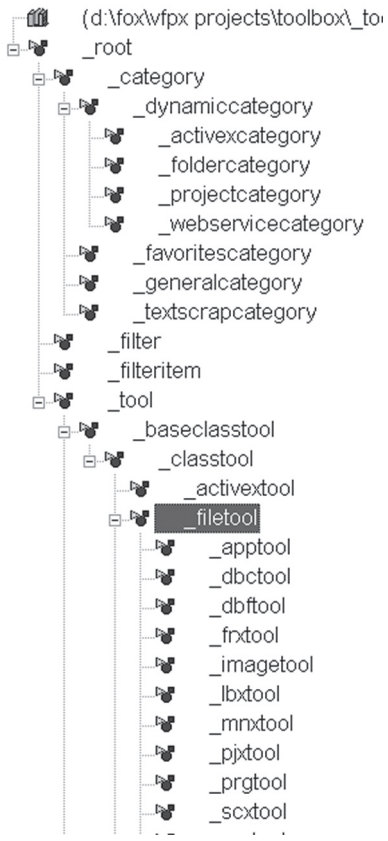

Figure 3. The tool classes used in the Toolbox all derive from a conmon ancestor, _root.

Adding a tool type

Although the Toolbox includes five types of items, and the file type is pretty broad, you might have other kinds of things you want to include in the Toolbox with their own behaviors. It turns out that adding new item types is fairly simple.

One of the projects I'm working on uses a Wiki and an online bug tracking system. While I have shortcuts in my browser for each of those sites, having them accessible from within VFP seems pretty useful. You can use a File type item for a link, but getting it set up right is a little tricky. So I decide to add a special Link type.

To add a new item type, you need to add a record to ToolType and create a class to handle it. Because _toolbox.vcx is excluded from the Toolbox project, you can add the new class in this library without otherwise affecting the Toolbox code.

Since the Toolbox already knows what to do if you specify a link for a File item (it opens the page in the default browser), base the new class on the one used for files (called _FileTool). As Figure 3 shows, _FileTool is several subclasses below _root in the hierarchy. It handles all the functionality common to the File data item and is further subclassed for functionality specific to a type of file.

The only thing the new class has to do differently is provide a way to add links easily. That comes down to modifying the Item Properties form, so that it provides a textbox to enter the URL rather than prompting for a filename.

It took some digging into the Toolbox code to figure out what had to change. The Item Properties dialog bases its contents on a collection called oDataCollection. The OnCreateDataValues method of _FileTool adds an item to the collection to handle the filename, specifying that it should use a control class called cfoxfilename. The line of code from _FileTool.OnCreateDataValues that sets this up is shown in Listing 1.

Listing 1. This line adds the controls for the filename to the Item Properties dialog for a File item.

```
THIS.AddDataValue("filename", '', ;
                  DATAVALUE_FILENAME_LOC, ;
                  '', .F., "cfoxfilename", '')
```

For the link-handling class, you want an ordinary textbox instead of the textbox and button combination that cfoxfilename specifies. That class comes from ToolboxCtrls.VCX, which also contains a regular textbox class called cfoxtextbox. Use that one instead for our link class.

So, create a subclass of _FileTool and call it _LinkTool. Then, in its OnCreateDataValues

methods, replace the cfoxfilename control with a cfoxtextbox. Listing 2 shows the code that does the trick. Rather than overriding the code in _FileTool.OnCreateDataValues (which does some other things as well), modify the member of oDataCollection to have the value you want. This code also modifies the caption used next to the text. You'll also need to add the line in Listing 3 to the Toolbox.H file.

Listing 2. The new _LinkTool class has code in only one method, OnCreateDataValues.

```
#include "toolbox.h"

DODEFAULT()

* custom to the _linktool, modify the info
* for the filename item
WITH This.oDatacollection("FILENAME")
   .DataCaption = DATAVALUE_LINK_LOC
   .EditClass = "cFoxTextBox"
ENDWITH
```

Listing 3. Add this line to Toolbox.H to make the code in Listing 2 work.

```
#DEFINE DATAVALUE_LINK_LOC
"Link"
```

With the code done, all you have to do is add a record to ToolType.DBF to make the whole thing work. Figure 4 shows data for this record. Classname points to the new _LinkTool class. Because it's in the _Toolbox.VCX classlib, you don't need to fill in the Classlib field; that's the default. Setting ShowNew to .T. ensures that this type will show up in the Add Item dialog. Setting PropSheet to .T. causes the Item Properties dialog to appear as part of the process of adding an item.

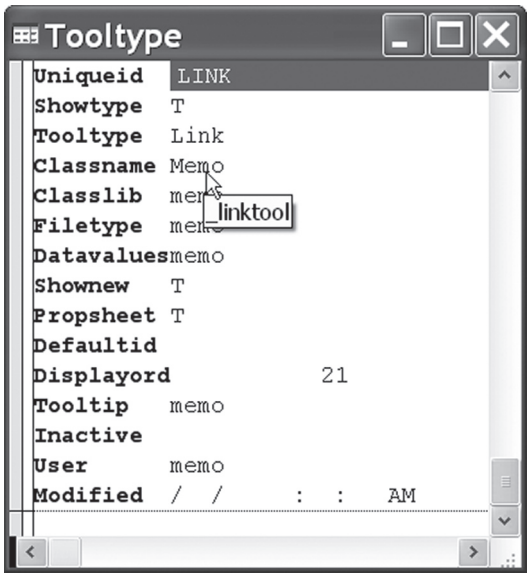

Figure 4. Add this record to the ToolType table to set up the Link tool.

Once you've made these changes, when you choose Add Item from the Customize Toolbox dialog, the Add Item dialog includes the Link type (Figure 5). When you choose the Link type, the Item Properties dialog appears, with a textbox for entering the link (Figure 6). When you've added a link, you can click on it to open the page in your browser; the Toolbox is even smart enough to use a link icon for it.

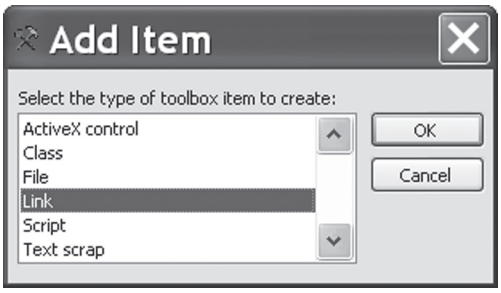

Figure 5. After adding the record in Figure 4 to the ToolType table, the Add Item dialog includes the Link type.

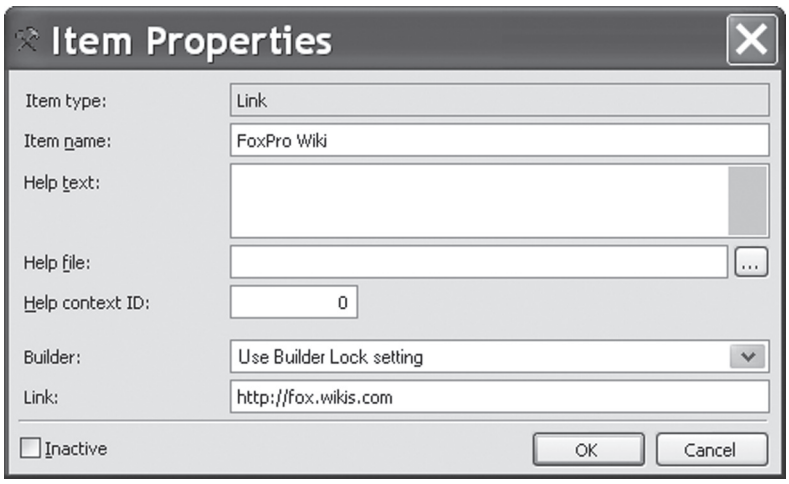

Figure 6. The Item Properties dialog for the new Link type includes a textbox to enter the URL.

Add items to the context menu

I've never quite understood why you can add class libraries to the Toolbox from the context menu, but have to open the Customize Toolbox dialog to add other kinds of items. Once I'd created the Link tool type, I decided to figure out how to access it without using the Customize Toolbox dialog. It turns out that the Toolbox supports an add-in mechanism that lets you add items to the shortcut menus.

Add-ins are stored in the main Toolbox data table, Toolbox.DBF. By default, there are no add-ins, but you can set them up.

An add-in for a shortcut menu item needs only a few fields of the Toolbox table filled in: UniqueID, ShowType, ToolName and ToolData.

UniqueID is the item's primary key. The Toolbox uses a two-part naming scheme for UniqueID. All categories have UniqueID values in the form "Microsoft.CategoryName." The built-in items have UniqueID values in the form "CategoryName.ItemName." Items you add through the Toolbox's interface have UniqueID values in the form "User.SYS2015value." I recommend that when you add items manually, you use a similar format with the first part identifying you (or your company) and the second part identifying the item. So, for the Add link menu item, I'll use "TSLLC.AddLink" as the UniqueID.

ShowType indicates the type of item the record represents. It contains "C" for category, "T" for "tool item," and so forth. For an add-in, use "A".

ToolName is the name used for the item in the Toolbox. For a menu item, put the text you want in the menu. For the Add link menu item, put "Add link" of course.

Finally, ToolData is the field that provides functionality. For an add-in, you put code to run when the add-in is chosen. For the Add link menu item, I modified code I found in the CreateToolItem method of the ToolboxEngine class; the modified code is shown in Listing 4.

Listing 4. This code implements the Add link menu item. It's stored in the ToolData memo field of the add-in's record in Toolbox.DBF.

```
LPARAMETERS oCurrentItem

LOCAL oEngine, oToolItem, oCategory, ;
      oToolType, lSuccess

oEngine = oCurrentItem.oEngine

oCategory = oEngine.CurrentCategory

oToolType = oEngine.GetTooltypeRec("LINK")
IF NOT ISNULL(m.oToolType)
   m.lShowPropertySheet = ;
      m.oToolType.PropSheet
   m.cClassName = m.oToolType.ClassName
   m.cClassLib  = m.oToolType.ClassLib
   IF EMPTY(m.cClassLib)
      m.cClassLib = oEngine.DefaultClassLib
   ENDIF

   TRY
      m.oToolItem = ;
          NEWOBJECT(m.cClassName, m.cClassLib)
   CATCH TO oException
      MESSAGEBOX(oException.Message + ;
                 CHR(10) + CHR(10) + ;
                 m.cClassName + ;
                 "(" + m.cClassLib + ")", ;
                 MB_ICONEXCLAMATION, ;
                 TOOLBOX_LOC)
   ENDTRY

   IF VARTYPE(m.oToolItem) == 'O'
      WITH m.oToolItem
         .oEngine    = m.oEngine
         .ToolTypeID = m.oToolType.UniqueID
         .ToolType   = m.oToolType.ToolType
         .ClassName  = m.cClassName
         .ClassLib   = ;
            IIF(m.cClassLib == ;
                oEngine.DefaultClassLib, ;
                '', m.cClassLib)
      ENDWITH

      IF m.lShowPropertySheet
         IF !oToolItem.Properties(.T.)
            m.oToolItem = .NULL.
         ENDIF
      ENDIF

      IF !ISNULL(m.oToolItem)
         WITH m.oToolItem
            .ParentID = oCategory.UniqueID
         ENDWITH

         lSuccess = ;
            oEngine.NewItem(m.oToolItem)
      ENDIF

   ENDIF

ENDIF

RETURN m.lSuccess
```

Once you've added the record to the Toolbox table and reopened the Toolbox, you can see the Add Link menu item when you right-click over an item or category. Unfortunately, it doesn't appear when you right-click over an empty space; changing that behavior would require changes to the core Toolbox code.

If you'd like to have a shortcut menu item for adding text scraps, you can do it the same way. Just change the UniqueId and ToolName fields, and modify the code for ToolData to use the string "TEXTSCRAP" instead of "LINK" in the call to GetToolTypeRec.

Adding a shortcut menu item to add files is different, though. When you add a file, you want the GetFile() dialog to appear so you can point to the file rather than the Item Properties dialog. I found the right code in the AddTool method of the form ToolboxCustomize and modified it as in Listing 5.

Listing 5. Adding a file is actually easier than adding other items. Put this code in ToolData for a shortcut menu item to add files.

```
LPARAMETERS oCurrentItem

LOCAL oEngine, oToolItem, oCategory, ;
      oToolType, lSuccess

oEngine = oCurrentItem.oEngine

oCategory = oEngine.CurrentCategory

m.cFilename = GETFILE()
IF !EMPTY(m.cFilename)
   m.lSuccess = ;
      oEngine.CreateToolsFromFile( ;
         oCategory.UniqueID, m.cFilename, .T.)
ENDIF

RETURN m.lSuccess
```

Although it's not relevant for these menu items, you can specify that a particular shortcut menu item appears only for a particular item type. To do so, put the item type in the ToolTypeID column.

Try it yourself

I hope the examples here inspire you to experiment with behavior changes you'd like in the Toolbox. They turn out to be surprisingly easy to make. The hardest part, in my experience, is finding code in the Toolbox that gives you a place to start.

Like the other Xbase tools, the Toolbox is included in VFPX, so please consider sharing your modifications with the VFP community.

This month's downloads include the updated _Toolbox.VCX class library with the _LinkTool class, the updated ToolType.DBF table containing a record for the new Link type, and AddMenuItems.PRG, a program that adds three new items (Add Link, Add Text Scrap and Add File) to the Toolbox context menu.

Author Profile

Tamar E. Granor, Ph.D. is the owner of Tomorrow's Solutions, LLC. She has developed and enhanced numerous Visual FoxPro applications for businesses and other organizations. She currently focuses on working with other developers through consulting and subcontracting. Tamar is author or co-author of ten books including the award winning Hacker's Guide to Visual FoxPro, Microsoft Office Automation with VisualFoxPro and Taming Visual FoxPro's SQL . Her latest collaboration is Making Sense of Sedna and SP2, coming out this year. Her books are available from Hentzenwerke Publishing (www.hentzenwerke.com). Tamar is a Microsoft Support Most Valuable Professional. In 2007, Tamar received the Visual FoxPro Community Lifetime Achievement Award. You can reach her at tamar@thegranors.com or through www.tomorrowssolutionsllc.com.