Tools for and by Geeks and Gurus

Doug Hennig

HackCX Professional and ViewEditor Professional are two tools from Geeks and Gurus that can make your life easier. HackCX allows you to edit SCX and VCX files in ways the Form and Class Designers can't, and ViewEditor is more convenient and safer to use than VFP's View Designer.

As professional developers, we rely on the tools VFP provides to create the applications our clients pay us to build. However, sometimes we run into problems VFP's tools can't handle. When that happens, our choices are to do things the hard way, to create our own tools, or to buy third-party tools. The first option is often error-prone and takes too long and the second option doesn't usually make a lot of sense—we get paid to build solutions for our clients, not for ourselves. Examples of third-party tools include the FixDBC utility I presented in the November 2003 issue ("Database, Heal Thyself"), the various table repair utilities available (such as FoxFix from XiTech and Recover from Abri Technologies), and HTML Help Builder from West Wind Technologies.

This month, we'll look at a couple of relatively new tools from Geeks and Gurus Inc.: HackCX Professional and ViewEditor Professional.

HackCX Professional

Have you ever edited a form created by a co-worker only to discover that they used VFP base classes rather than your company's base classes? Maybe you've created a subclass but then realized it has the wrong parent class. These aren't the kinds of things you can change in the VFP Form and Class Designers; once you've chosen a particular object or class, the only way to change it is to delete the object or class and start over.

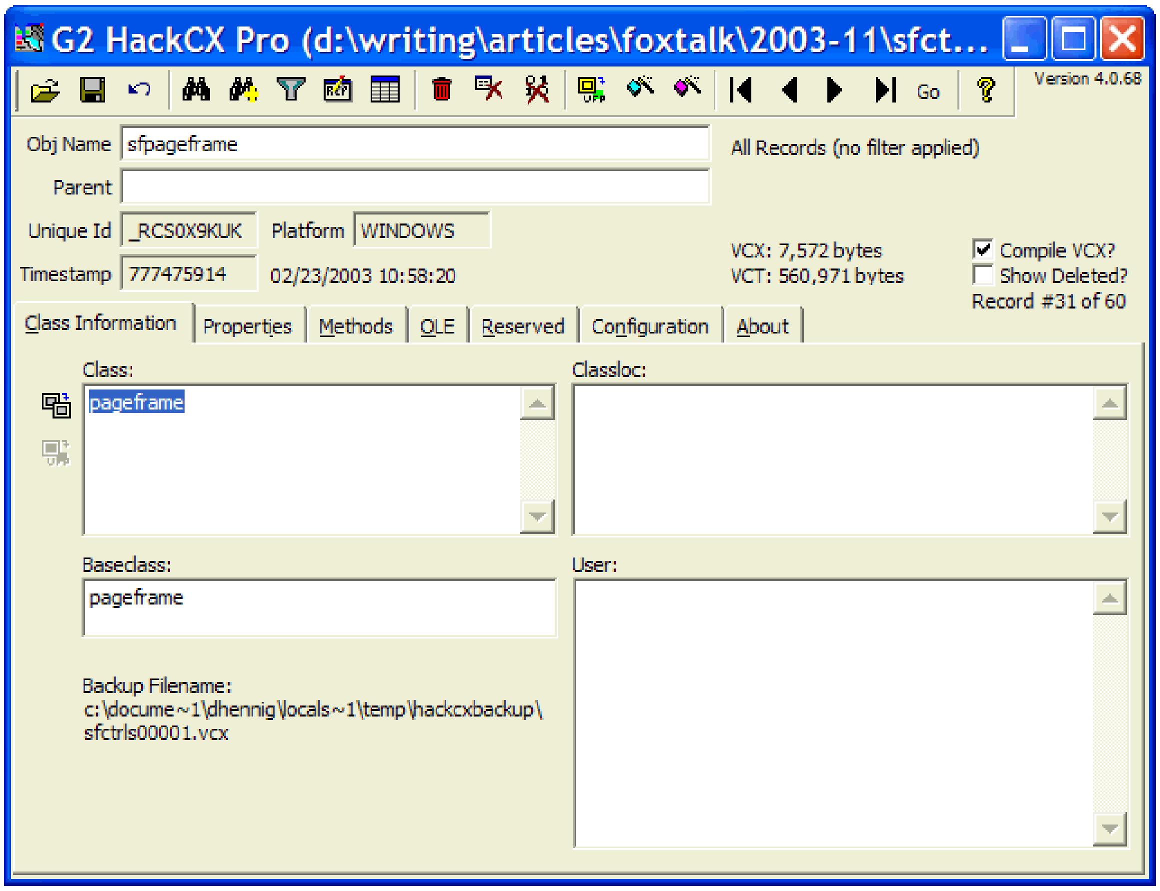
Your first thought after discovering this problem might be, "Oh, man, all that work down the drain". Then you remember that VCX and SCX files are really VFP tables with unconventional extensions, so you USE WHATEVER.SCX and browse it, only to see field names like UNIQUEID, TIMESTAMP, PLATFORM, USER, and a slew of field names starting with RESERVED. Where to begin?

Fortunately, there's a tool that can make these types of changes to SCX and VCX files fast and easy: HackCX Professional. HackCX, written by well-known VFP guru Rick Schummer, allows you to do the following:

- Redefine the class and class library of the parent class or redefine the base class of objects. Although the Class Browser also has this feature, it doesn't allow you to redefine the class of objects inside containers, only the container itself.
- Search for or filter on values in any of the SCX or VCX columns.
- Safely edit anything about any object or class.
- Quickly change the scope (public, protected, or hidden) of properties and methods.
- Recompile all code in the entire class library or form.
- Add properties and methods, and set the default value for new properties.
- Edit the code in the SCX or VCX User column.

To run HackCX, do HACKCX4.EXE. It prompts you for the name of an SCX or VCX file to edit. HackCX is smart enough to do a backup of the VCX/VCT or SCX/SCT files before you begin to make any changes (you get to decide where the backups are made in the Configuration page), and all changes are made in a table-buffered mode so you can safely reverse them if necessary. Figure 1 shows HackCX with SFCTRLS.VCX open and positioned to the record for the SFPageFrame class.

Figure 1. HackCX Professional makes it easy to hack a VCX or SCX file to make changes the Class or Form Designer won't allow.



Obj Name: sfpageframe

All Records (no filter applied)

Parent:

Unique Id: _RCS0X9KUK

Platform: WINDOWS

Timestamp: 777475914 02/23/2003 10:58:20

VCX: 7,572 bytes

VCT: 560,971 bytes

[✓] Compile VCX?

[ ] Show Deleted?

Record #31 of 60

Class: pageframe

Classloc:

Baseclass: pageframe

User:

Backup Filename:
c:\docume~1\dhennig\locals~1\temp\hackcxbackup\sfctrls00001.vcx

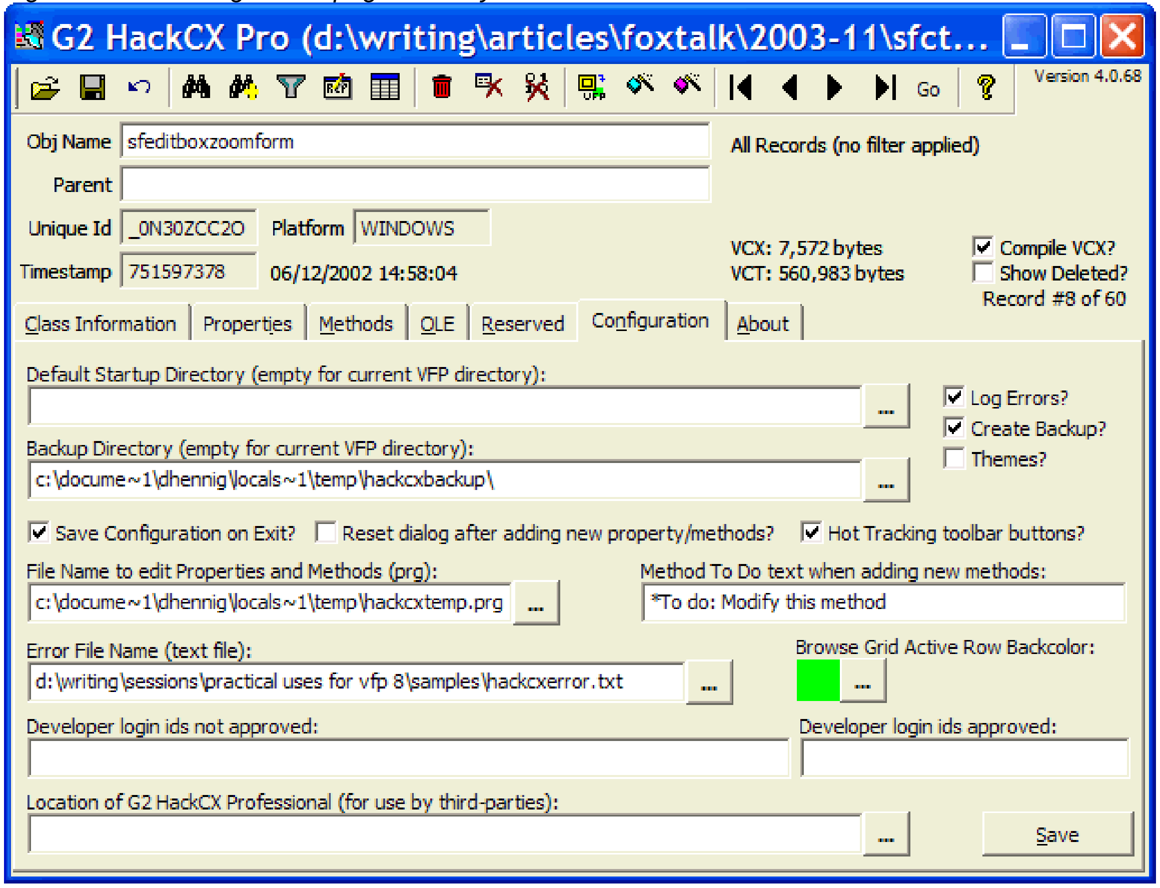
Before we discuss the HackCX form, let's discuss the layout of an SCX or VCX file. Each object in a form has its own record in an SCX file, as does each class and each object in a class in a VCX file. For example, in SFCTRLS.VCX, SFPageFrame consists of just one object (the page frame subclass), so there's just one record for it in the VCX (actually, there's two—a Comment record is also present, but we aren't really interested in Comment records). SFEditBoxZoomForm, however, has three records, because it's a form class with a command button and an edit box. You can tell that the command button and edit box records belong to the SFEditBoxZoomForm class because their Parent properties contain "sfeditboxzoomform". Every record has a UniqueID (its primary key in the table), Platform (always "Windows"), Timestamp (the date and time the record was modified, stored as an encoded value but expanded in HackCX to a readable format), and ObjName (the name of the object). Class and ClassLoc contain the class the object is based on and the class library where that class is defined (empty for VFP base classes), while BaseClass contains the VFP base class. Properties are stored in the Properties memo with each one on a separate line (only custom properties and those with changed values appear). Protected properties and methods are listed in the Protected memo. Source code for methods is found in the Methods memo, while ObjCode contains the compiled source. If the object is an ActiveX control, information about it is contained in Ole and Ole2. Eight Reserved fields (Reserved1 through 8) contain a variety of information, such as the include file for the class, the class description, and the icon for the Form Controls tool bar. User-defined information can be placed in the User memo.

The HackCX form consists of a toolbar, an area showing information about the SCX or VCX file and about the current record in the file, and a page frame with more information about the current record or the HackCX configuration. In Figure 1, we can see that SFPageFrame is a class (Parent is empty) based on the VFP PageFrame base class (from the values in Class and ClassLoc) and was last modified on 02/23/2003. We can also see the size of the VCX and VCT files, the record number we're on, and the name of the backup file HackCX created for us.

The Class Information page shows information about the class: the Class, ClassLoc, BaseClass, and User properties. There are also command buttons to redefine the class so it's based on another class or on a

VFP base class. You could do this yourself by typing the appropriate values into the Class and ClassLoc columns, but since ClassLoc is a relative path from the current SCX or VCX to the class library for the parent class, it's more convenient to let HackCX figure out the proper path for you. The Properties page shows the contents of the Properties and Protected memos, while the Methods page shows the contents of Methods. The ObjCode memo isn't shown in this page, since it wouldn't be readable and isn't hand-editable. However, there are functions in HackCX that allow you to clear the contents of ObjCode (handy if it become corrupted, for example) or to force a compile of the code in Methods so ObjCode is refreshed. The Ole page shows the contents of the Ole and Ole2 memos for ActiveX controls. Ole is read-only because it contains binary information that would certainly become corrupted if you tried to edit it, while Ole2, which contains the path for the OCX file, is editable. The Reserved page shows the contents of the eight Reserved fields, with an explanation of the purpose of each one. The last two pages aren't about the current record, but contain customization information (see Figure 2) and information about HackCX.

Figure 2. The Configuration page allows you to customize HackCX.


G2 HackCX Pro (d:\writing\articles\foxtalk\2003-11\sfct...

Version 4.0.68

Obj Name: sfeditboxzoomform

Parent:

All Records (no filter applied)

Unique Id: _0N30ZCC2O

Platform: WINDOWS

Timestamp: 751597378 06/12/2002 14:58:04

VCX: 7,572 bytes
VCT: 560,983 bytes

[✓] Compile VCX?
[ ] Show Deleted?
Record #8 of 60

Class Information | Properties | Methods | OLE | Reserved | Configuration | About

Default Startup Directory (empty for current VFP directory):

Backup Directory (empty for current VFP directory): c:\docume~1\dhennig\locals~1\temp\hackcxbackup\

[✓] Log Errors?
[✓] Create Backup?
[ ] Themes?

[✓] Save Configuration on Exit? [ ] Reset dialog after adding new property/methods? [✓] Hot Tracking toolbar buttons?

File Name to edit Properties and Methods (prg): c:\docume~1\dhennig\locals~1\temp\hackcxtemp.prg

Method To Do text when adding new methods: *To do: Modify this method

Error File Name (text file): d:\writing\sessions\practical uses for vfp 8\samples\hackcxerror.txt

Browse Grid Active Row Backcolor: 🟩

Developer login ids not approved:

Developer login ids approved:

Location of G2 HackCX Professional (for use by third-parties):

The toolbar and shortcut menu has functions to:

- open an SCX or VCX file
- save or revert changes
- navigate in the file (the usual first, last, next, and previous functions, as well as a Go function to move to a particular record number)
- find a record (a dialog allows you to select the field to search in and the value to search for) or find the next record
- filter the file on a certain condition (for example, BaseClass = "commandbutton" would only show objects based on CommandButton)

- display a “command” window in which you can enter commands to execute (such as a REPLACE ALL command to perform some global change in the file)
- browse the file
- delete the current record
- pack the file
- clear the contents of ObjCode
- redefine all classes to VFP base classes (probably only useful if you’ve created a form or class based on classes in your framework and want to send it to someone who doesn’t have your framework)
- create a new property or method
- display help (the help file is quite complete and, well, helpful)

Here’s an example of how I actually used HackCX recently. SFEditBoxZoomForm has a command button that, when clicked, displays the GetFont dialog so the user can specify the font and size for the edit box. However, I realized that I already had a class to do that (SFGetFont in SFBUTTONS.VCX) but hadn’t used it in SFEditBoxZoomForm, so if I wanted to change the appearance or behavior of font buttons, I’d have to change it in two places. So, I opened SFCTRLS.VCX, navigated to the cmdFont record in the SFEditBoxZoomForm class (I simply used the Next function until I came across the record, but could have used the Find function instead), clicked on the Redefine Class button, and selected SFGetFont in SFBUTTONS.VCX. HackCX notified me that there were changes pending by showing both “File Changed!!” and “Obj Changed!!” in the upper right corner of the form (moving to a different record clears the “Obj Changed!!” indicator but not “File Changed!!”). I then clicked on the Save button. Total effort: about 15 seconds.

HackCX is now my preferred mechanism for hacking SCX and VCX files. It’s so useful that I’ve added an item to my VFP Tools menu (there’s a topic in the help file that tells you how to do that) to pop it up whenever I need it. Hacking SCX and VCX files isn’t something you do every day (hopefully!), but when you need to do it, you really need to do it, and HackCX makes it quick, easy, and safe (well, you can still mess things up if you don’t know what you’re doing, but at least it gives you a backup and the option to revert your changes).

ViewEditor Professional

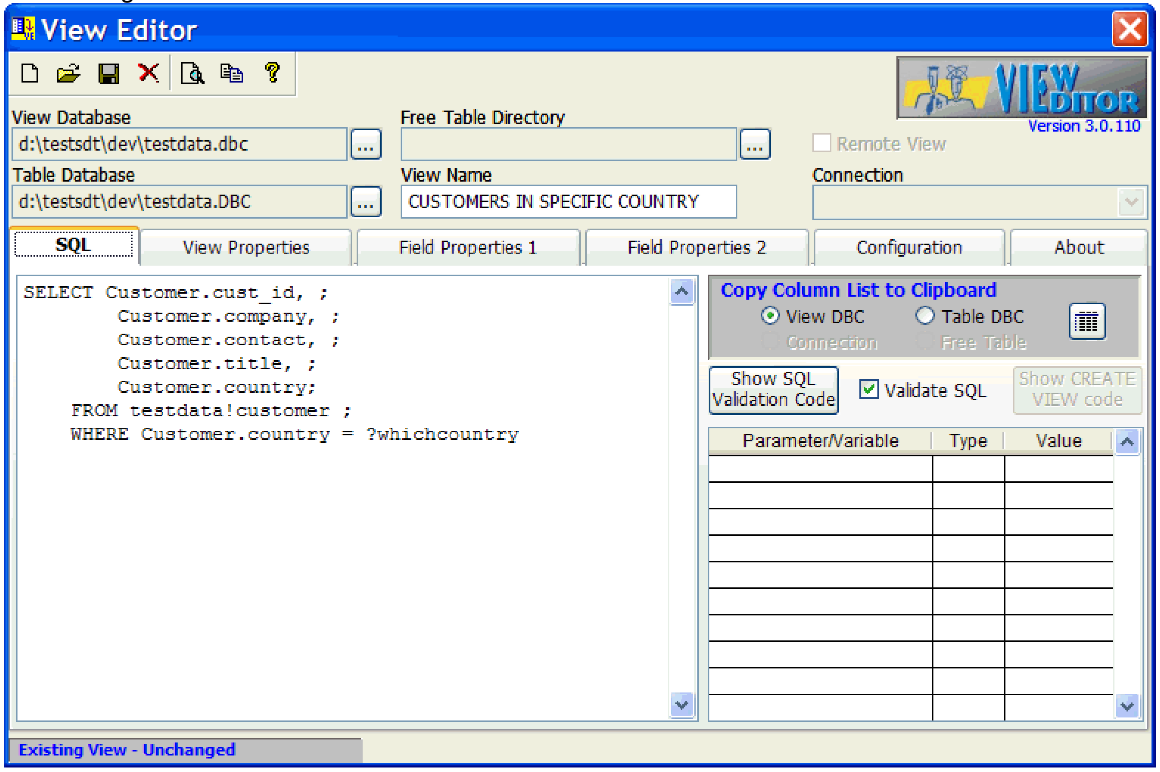
Until recently, the View Designer had a deservedly bad reputation (some people called it the “View Mangler”) because it couldn’t handle views of any complexity. For example, a three-table view was certain to become corrupted if edited in the View Designer. That’s changed in VFP 8; Microsoft did a great job not only fixing the problems in the View Designer, but adding some much needed features, such as an editable SQL window. However, if you’re still working in earlier versions of VFP and need to create or maintain views, or if you want a tool that has a cleaner, easier-to-user interface than the View Designer, ViewEditor Professional is the tool for you.

ViewEditor, written by another well-known VFP guru, Steve Sawyer, has several features that make it easier to use than the View Designer, including:

- You can write your own SQL SELECT statement, so you don’t have to struggle with a visual interface to get exactly the result set you want.
- You can set view and view field properties quickly and easily in the same dialog as the view is edited.
- You can “inherit” table field properties, such as Caption and DefaultValue, with a single click.
- It supports views in a different database than the tables the view is based on, and allows you to specify a table directory when creating views from free tables.

To run ViewEditor, do VE3.EXE. You can optionally pass the names of a view and the database container the view is defined it to automatically open that view. If you don't pass these names, you can open up to two databases (one for the view and optionally one for the source tables) and a view using the appropriate functions. ViewEditor maintains a "most recently used" list of databases, so you can quickly open a previously selected database by right-clicking on the open database button and selecting the desired database from the context menu. Figure 3 show ViewEditor working with the Customers in Specific Country view in the TESTDATA database.

Figure 3. ViewEditor is easier to use and doesn't suffer from the "view mangler" problems of the View Designer.



View Database: d:\testsdt\dev\testdata.dbc
Table Database: d:\testsdt\dev\testdata.DBC
View Name: CUSTOMERS IN SPECIFIC COUNTRY
Copy Column List to Clipboard: [✓ View DBC] Table DBC
[✓ Validate SQL]

```
SELECT Customer.cust_id, ;
       Customer.company, ;
       Customer.contact, ;
       Customer.title, ;
       Customer.country;
   FROM testdata!customer ;
   WHERE Customer.country = ?whichcountry
```

Like HackCX, the ViewEditor form consists of a toolbar, an area showing information about the view, and a page frame with properties for the view and the fields in the view. There's also a status bar showing whether the open view is new or existing and whether it's been changed or not. The toolbar has function to create or open a view, save or delete the current view, preview the view (that is, open and browse the result set), copy the view code (including code to set the view and field properties), and display the help file.

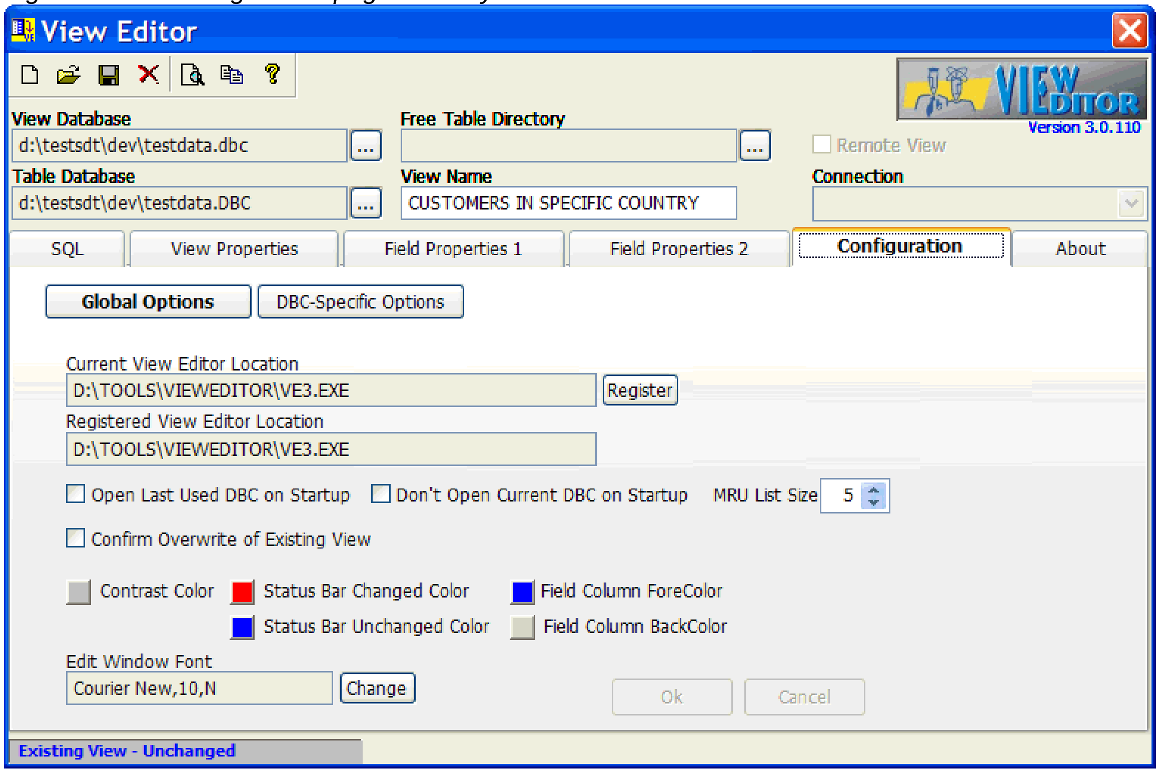
The SQL page has an edit box in which you can edit the SQL SELECT statement for the view. Double-clicking in the edit box displays a code window with syntax coloring and IntelliSense. Because typing the list of fields for a view can be quite tedious and error-prone, ViewEditor provides a feature that allows you to put all of the fields for a specific table onto the Clipboard so you can paste it into the edit box. You can enter the list of parameters for the view, their data types, and values used for testing the view into a grid (much handier than the modal dialog the VFP View Designer uses for this). ViewEditor automatically validates a view (although you can turn this off) to ensure it's correct, and a couple of command buttons allow you to see the code used for testing the view and creating the view.

The View Properties page allow you to set the properties of the view, such as the update type, where type, and tables to update. ViewEditor does a good job of ensuring that the Tables property has the database name properly specified and doesn't list any tables that aren't supposed to be updated, a failing of the View Designer that will cause you grief at the least opportune moment. Some of the properties, such as Rule Text and Rule Expression, aren't even exposed in the View Designer; you have to set them in code if you don't use ViewEditor.

The Field Properties 1 page combines the features of the View Designer's Update Criteria page and View Field Properties dialog (which is, of course, annoyingly modal). You can specify which fields are updatable, which is the key field, and the update name and data type for each field. There's also a button to ensure the update name property for each field is correct, with both the database and alias name specified. The Field Properties 2 page covers the rest of the field properties: caption, default value, rule text, rule expression, and comment. It also provides a button that copies properties from the source table field to the view field so you don't have to manually set these properties.

The last two pages allow you to customize ViewEditor's appearance and behavior (see Figure 4) and see information about ViewEditor. The DBC-Specific Options settings are especially useful, because you can define default values for new views, such as update type, where type, and the location of the source tables (table database or free table directory).

Figure 4. The Configuration page allows you to customize ViewEditor.



View Editor

Version 3.0.110

View Database: d:\testsdt\dev\testdata.dbc

Free Table Directory:

Remote View

Table Database: d:\testsdt\dev\testdata.DBC

View Name: CUSTOMERS IN SPECIFIC COUNTRY

Connection:

Current View Editor Location: D:\TOOLS\VIEWEDITOR\VE3.EXE

Registered View Editor Location: D:\TOOLS\VIEWEDITOR\VE3.EXE

☐ Open Last Used DBC on Startup ☐ Don't Open Current DBC on Startup MRU List Size 5

☐ Confirm Overwrite of Existing View

Contrast Color, Status Bar Changed Color, Field Column ForeColor

Status Bar Unchanged Color, Field Column BackColor

Edit Window Font: Courier New,10,N

Existing View - Unchanged

Here's an example showing how I used ViewEditor to create a new view. I opened the TESTDATA database that comes with VFP and ran ViewEditor. I clicked on the Create New View button, entered the view name, and typed the following SQL SELECT statement:

```
select * FROM customer ;
  where country = ?pcCountry
```

In the Parameters grid, I added pcCountry and specified C (for Character) as the data type. On the View Properties page, I turned on the Send Updates properties to make the view updatable. On the Field Properties 1 page, ViewEditor properly identified that CUST_ID was the key field and flagged the rest of the fields as updatable, so I didn't have to do anything. I saved the view, then tested it by clicking on the Preview button.

Okay, that was a pretty simple view. Let's try a more complex one. I want a view showing information about orders. I started with the View Designer (using VFP 7), creating a new view and selecting the Customer, Orders, Employee, Orditems, and Products tables. I selected several fields from each table, then saved the view. Here's the SQL SELECT statement the View Designer generated:

```
SELECT Customer.company, Orders.order_id,;
  Orders.order_date, Orders.order_amt,;
  Orders.order_dsc, Orders.order_net,;
  Employee.first_name, Employee.last_name,;
  Orditems.line_no, Orditems.unit_price,;
  Orditems.quantity, Products.eng_name;
 FROM testdata!customer INNER JOIN testdata!orders;
    INNER JOIN testdata!orditems;
    INNER JOIN testdata!products;
    INNER JOIN testdata!orders;
    INNER JOIN testdata!employee ;
   ON  Employee.emp_id = Orders.emp_id ;
   ON  Employee.emp_id = Orders.emp_id ;
   ON  Products.product_id = Orditems.product_id ;
   ON  Orders.order_id = Orditems.order_id ;
   ON  Customer.cust_id = Orders.cust_id
```

Notice the nested JOIN syntax, which I find confusing and never use. When I reopened the view in the View Designer, the Employee table was missing in the graphical view and the SQL SELECT statement (although its fields were still there), so the View Designer lived up to its "View Mangler" reputation. (To be fair, this didn't happen when I did the same thing in VFP 8.)

I deleted the view and ran ViewEditor. I created a new view, entered a name for the view, and typed the SQL SELECT statement using the syntax I prefer:

```
SELECT Customer.company, Orders.order_id, ;
    Orders.order_date, Orders.order_amt, ;
    Orders.order_dsc, Orders.order_net, ;
    Employee.first_name, Employee.last_name, ;
    Orditems.line_no, Orditems.unit_price, ;
    Orditems.quantity, Products.eng_name ;
  FROM testdata!customer ;
    INNER JOIN testdata!orders ;
      ON Customer.cust_id = Orders.cust_id ;
    INNER JOIN testdata!employee ;
      ON Employee.emp_id = Orders.emp_id ;
    INNER JOIN testdata!orditems ;
      ON Orders.order_id = Orditems.order_id ;
    INNER JOIN testdata!products ;
      ON Products.product_id = Orditems.product_id
```

Now, so long as I don't edit this view in the View Designer, it'll be fine. Thus, not only is ViewEditor more convenient to use (everything in once place without reliance on various modal dialogs), it's also safer.

Summary

HackCX and ViewEditor are licensed per developer. HackCX is $50 and ViewEditor is $75; a bundle with both tools is $100. If you'd like to read the documentation for these tools before purchasing, you can download the help files from http://www.g2tools.com. You can also purchase these tools online (via PayPal) from this Web site.

Rick and Steve have created a couple of must-have tools in HackCX and ViewEditor. While you may not use them every day, they are the type of tools that can make short work of the tasks you must perform when you do need them, and at the price, it's a no-brainer purchase decision. Check these tools out, and I'm sure you'll find as I have that they pay for themselves the first time you use them.

Doug Hennig is a partner with Stonefield Systems Group Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT) and Stonefield Query, author of the CursorAdapter and DataEnvironment builders that come with VFP 8, and co-author of "What's New in Visual FoxPro 8.0", "What's New in Visual FoxPro 7.0", and "The Hacker's Guide to Visual FoxPro 7.0", from Hentzenwerke Publishing. Doug has spoken at every Microsoft FoxPro Developers Conference (DevCon) since 1997 and at user groups and developer conferences all over North America. He is a Microsoft Most Valuable Professional (MVP) and Certified Professional (MCP). Web: www.stonefield.com and www.stonefieldquery.com Email: dhennig@stonefield.com