December, 2006

Advisor Answers

Sedna, VFPX and VFPy

VFP 9

Q: What is Sedna? Is that the code name for VFP 10? When will it ship?

A: While Sedna is the code name for the next release of Visual FoxPro, it's almost certainly not going to be called VFP 10. After evolving for about 20 years, VFP offers a very mature language and development environment. While many of us would like to see various changes, it's hard to think of major enhancements for the core of the product; that is, there's nothing out there that justifies a complete version upgrade.

VFP has always had a fairly open architecture compared to other products. Many of the native tools store their data in tables, and quite a few of the tools are written in VFP code. The latter are generally referred to as the Xbase tools, and VFP comes with the source code for almost of them.

Over the last few versions, Microsoft has made the product even more open. For example, while the core IntelliSense functionality is built in, IntelliSense was designed from the start to be extensible. VFP 9 introduced the _MENUDESIGNER variable that lets you substitute a different menu designer tool. Of course, the entire new reporting system, while rooted in the core product, is open and extensible; you can add your own report listeners and previewers, and even replace the report builder application that's used at design-time. VFP 9 also added the ability to replace items on the native VFP menus with your own code and provides significant customization options for the Property Sheet.

At the same time, recent versions have included a number of new Xbase tools. VFP 7 gave us the Object Brower (and the much less useful Task List). VFP 8 introduced the Toolbox, Task Pane Manager and Code References tools, and VFP 9 added the Data Explorer. VFP 9 Service Pack 1 also included a new license for the Xbase tools to make it easier to distribute enhanced and extended versions.

The result of all these changes is that it's easier than ever for us to change VFP's behavior without Microsoft's intervention.

Given the completeness and the openness of the product, Microsoft has decided to minimize changes to VFP's core (VFP.EXE) for the time being. While they plan another service pack to fix bugs, future enhancements to VFP will almost all come as add-on DLLs or updates to Xbase components.

Sedna is the first effort in this direction. It has a few major goals including making it easier to use VFP 9 with .NET and SQL Server 2005; improving the use of VFP 9 with the latest release of Windows, Windows Vista; and enhancing the new reporting system. In addition, it includes improvements to some of the Xbase tools. You can see Microsoft's plan for Sedna at http://msdn.microsoft.com/vfoxpro/roadmap/.

Part of the plan is to work more transparently. Rather than waiting until the product is almost done and then offering a preview beta to the community, the Fox team has been providing periodic Community Technology Previews (CTPs). Unlike the public betas for VFP 8 and VFP 9, the CTPs aren't all-inclusive. They highlight what the team has been working on over a particular period of time. You can find the CTPs at the Visual FoxPro website, http://msdn.microsoft.com/vfoxpro/.

Microsoft expects to release Sedna in the first half of 2007. Compatibility with Windows Vista is one of its goals. Microsoft has also promised a second Service Pack for VFP 9 at the same time. (So if you've found any bugs in the product, be sure to submit them using the online bug reporting tool available at http://connect.microsoft.com/. VFP is part of the "Visual Studio and .NET Framework" connection.)

FoxPro developers have always been known for their community spirit. In that vein and given the open nature of VFP, there are two public projects to create and enhance open source add-ons for VFP in the Sedna timeframe.

VFPX (originally called SednaX—the "X" stands for "eXtensions") can be found at http://www.codeplex.com/Wiki/View.aspx?ProjectName=VFPX. It includes the updated New Property/Method dialog I mentioned in the September issue, as well as a number of other tools. All are available for download, and you're welcome to jump in and help, too.

VFPy (as far as I know, the "y" indicates the next one after "x") is located at http://www.codeplex.com/Wiki/View.aspx?ProjectName=VFPy. The

star offering there is ActiveVFP, a web development framework. Again, you can simply use what's offered or choose to help.

Microsoft hasn't given any clues yet as to what will come after Sedna, but it's clear that the VFP community will continue to offer new and better tools.

–Tamar

Handling duplicates

VFP 9/8/7

Q: My users often create duplicate records with minor variations in spelling or punctuation. How can I get rid of those duplicates?

A: What's known as "de-duping" is a long-standing database problem. The database aspect actually isn't that hard. What's difficult is deciding what constitutes a duplicate record. In fact, that's not a programming problem, but a business problem.

The first step in handling duplicates is to determine what might constitute a duplicate record. For example, do two student records with the same name indicate duplication? Maybe, but frequently not. After all, some names are quite common and even uncommon names are sometimes legitimately duplicated. (One of my sons went to school with two girls who had the same name down to the middle initial, even though neither the first nor the last name was particularly common.)

How about two patient records with the same name and address? More likely, but they could be parent and child. (The ultimate example of this, of course, is the boxer George Foreman, who named all five of his sons George.)

Even something like identical social security numbers may indicate a data entry error rather than duplication. So, the first lesson is that in most cases, code can't decide whether two records are duplicates or not; a human has to make that decision. The best code can do is suggest that two records might be duplicates and present them for inspection.

The second problem is that typically, the reason for duplicates isn't exact matches, but near matches. Misspellings, changes in punctuation, using different forms of a name ("Michael" vs. "Mike," for example), and other such variations won't turn up when searching for

exact matches, unless you can identify fields to match that aren't subject to such problems.

VFP has some tools to help in such situations. For example, you can reduce a string to only alphabetic characters and spaces by applying CHRTRAN() twice, like this:

```
? CHRTRAN( cString, ;
           CHRTRAN(LOWER(cString), ;
           "abcdefghijklmnopqrstuvwxyz ", ""), "")
```

If you want to keep digits as well, add them to the second parameter of the inner call:

```
? CHRTRAN( cString, ;
           CHRTRAN(LOWER(cString), ;
           "abcdefghijklmnopqrstuvwxyz 0123456789", ""), "")
```

VFP includes two functions that help somewhat in de-duping, though they're not as useful as they initially seem. SOUNDEX() takes a string and returns a four-character code (a letter followed by three digits) that represents the sound of the original string. It's not bad at matching varying spellings, especially of names, but the letter is always the first letter of the original string, so homonyms like "Kathy" and "Cathy" return different values ("K300" and "C300", respectively). In addition, SOUNDEX() only encodes the first four distinct consonant sounds, so strings that vary near the end may have the same value. For details on the algorithm used by SOUNDEX(), see http://en.wikipedia.org/wiki/Soundex. You can find several alternatives to SOUNDEX() on the FoxPro wiki: http://fox.wikis.com and search for Soundex.

DIFFERENCE() takes two strings and returns a value between 1 and 4 to indicate how similar the strings are. The higher the number, the closer the match. Like SOUNDEX() however, the first letter counts more, so identical-sounding strings that have different first letters never rank higher than 3.

Neither SOUNDEX() nor any of its variants will handle the nickname problem. To deal with that, you'll have to write some code. One option is to create a translation table where each record has a string you might find and the string to substitute for matching purposes. Then, write a function that takes the name you have and looks it up in the table. One problem with this strategy is that a particular nickname might actually be used for more than one name. For example, the

nickname "Ted" sometimes stands for "Theodore," but sometimes stands for "Edward."

Since the nickname problem applies only to first names, not to surnames, another option is to use only the first initial when looking for duplicates. Again, this isn't an ideal solution because some nicknames have a different initial than the name they stand for (such as "Tony" for "Anthony"). Unfortunately, there is no perfect solution.

A variant of the nickname problem that applies mostly to company and organization names is easier to solve. Some words tend to be written differently at different times. The most common is probably "and," which gets written variously as "and," "&", and "+." Similarly, "company" is sometimes spelled out and sometimes written as "Co." A substitution table is far more effective in this case than with people's names.

Another problem related to the nickname problem also applies to names of organizations and businesses. These names tend to have multiple words and some of them may be omitted at times. For example, you might find "The Philadelphia Zoo," "Philadelphia Zoo" and even "The Philadelphia Zoological Gardens." It's not hard to eliminate common words like "the"; just use STRTRAN():

```
? ALLTRIM(STRTRAN(" " + cString + " ", " the ", "", -1, -1, 1))
```

Note the spaces before and after the word "the" to ensure we don't remove the word "the" from the middle of another word (like "other"). Since "the" might appear at the beginning of the string we're checking, I've added a space in front of that string. Similarly, a space afterwards ensures that we can find the desired word at the end (though it's unlikely "the" would appear at the end of the string to be checked). The result is wrapped in ALLTRIM() to remove those extra spaces, if necessary. The function call also uses the optional Flags parameter of STRTRAN() to make the search case-insensitive.

Putting all this together, the transformations you might want to apply to a given field before looking for duplicates are:

- Remove punctuation
- Eliminate spelling differences (whether via SOUNDEX() or something else)
- Make nicknames and abbreviations uniform
- Remove helper words

For a given situation, you can choose the appropriate transformations and write a function that handles them. For a more general solution, you could data-drive the whole process, with a table to list the fields of interest and the transformations to apply.

Oddly, once you've solved the business problem of what constitutes a potential duplicate, identifying them is easy. A single query (or prior to VFP 9, two queries) for each combination of expressions that identifies potential duplicates does the trick.

To demonstrate, I created a table based on the Customer table from the example Northwind database. My table, called Cust, contains a subset of the records from the original. It also contains duplicates of several records, with modifications to the CompanyName field. For this example, let's look for duplicates with similar company names in the same country. Here's the query that finds the suspects:

```
SELECT * ;
   FROM Cust ;
     JOIN (SELECT SOUNDEX(companynam) as ccode , ;
                  Country ;
             FROM Cust ;
             GROUP BY 1,2 ;
             HAVING CNT(*)>1) Dups ;
        ON SOUNDEX(CompanyNam) = cCode ;
           AND Cust.Country = Dups.Country ;
   ORDER BY CompanyName ;
   INTO CURSOR PotentialDups
```

The query uses a derived table, that is, a subquery in the FROM clause, which finds all the unique combinations of SOUNDEX(CompanyNam) and Country and retains only those combinations for which there's more than one record. The main query then retrieves all the records in the table that have those particular combinations. In VFP 8 and earlier, put the code for the derived table into a separate query that saves the results in a cursor, then use that cursor in the main query. (The Cust table and the query above are included on this month's Professional Resource CD.)

If there are several ways to specify potential duplicates, you'll need to execute several queries and then combine the results. Once you have the list of potential duplicates, you can show it to users and let them decide which records to keep and which to discard or consolidate.

–Tamar