User-defined commands

(Tom Rettig vs R. Russell Freeland)

User-Defined Commands

Russ:

- Ashton-Tate was the first company to introduce the concept of user-defined commands (UDCs) when they demonstrated their dBASE IV compiler in 1988. However, despite Clipper 5.0’s release problems and departure from the dBase language, Nantucket’s likely to be the first company to implement UDCs.

Clipper originally introduced user-defined functions (UDFs) which proved so useful that they are now found within all popular dBase dialects. UDFs added functionality to the language by letting users execute user-written routines from within an expression. Before UDFs, they were limited to writing procedures executed by the DO command. Thus, UDFs have added genuine functionality which wasn’t available previously.

UDCs add no such new functionality. Whatever you can do with a UDC, you can do with a DO procedure. The advantage to the traditional method is that when you see a DO command, you know right away that a user-written procedure is being executed. But, suppose you are called upon to modify a program using the PROMPT command. What does it do? You look it up in the software documentation, but it isn’t there. Scan through the inadequately documented header file and you’ll see it right after the FOO command and dozens of others you never heard of. Reading, understanding, and modifying the program are all made more time-consuming and costly.

Commands or reserved-word instructions are the foundation of a language. The fact that they’re limited in number and always perform the same operation provides a standard upon which all programmers in a language can depend. The widespread availability of programmers proficient in one or more dialects of the dBase language is one of its primary attractions. If your programmer gets hit by a truck, you can hire another who’s already trained and experienced. But, if your programmer or a third-party library defines its own commands, you won’t be able to easily replace either.

Tom Rettig heads Tom Rettig Associates, in Beverly Hills, a supplier of software tools and custom programming. He is the author of Tom Rettig’s FoxPro Handbook and the soon-to-be-published Tom Rettig’s Clipper 5.0 Handbook, both by Bantam. He can be reached at (213) 272-3784.

- Tom, user-defined commands are the best thing that has ever happened to Schmeebase. Let’s go over your objections:

  - First, you say, “Whatever you can do with a UDC, you can accomplish with a DO procedure.” I’m not thoroughly familiar with the Clipper implementation yet, but in the Ashton-Tate version of UDCs, you can do much more with UDCs than with UDFs. UDCs take advantage of scopes, field-lists, etc., as if they were built-in commands. Let’s look at an example of how to optimize some code using one command with a scope:

First, the code before optimization:

SELECT 2
USE master
ORDER 44 or INDEX
SELECT 1
USE transaction
DO WHILE .NOT. EOF()
  SEEK <...>
  REPLACE quantity WITH ;
  transaction->quantity
  SELECT transaction
  SKIP 1
ENDDO

Now, with optimization:

SELECT 2
USE master
ORDER 44 or INDEX
SELECT 1
USE transaction
SET RELATION TO <...> ;
INTO master
REPLACE ALL master->quantity;
WITH master->quantity =;
quantity
REPLACE ALL works immensely faster than individual REPLACEs within a DO loop. As implemented in A-T’s Professional Compiler (whenever it gets finished), UDCs have as much access to scopes, field-lists (including calculated fields), etc., as do the "default extensions" that implement the dBASE IV language itself. That makes it possible for "black box" UDCs to act on sets of fields and records without using "*os" to slow them to a crawl.

Second, you say, "Reading, understanding, and modifying the program are all made more time-consuming and costly." I have no idea what you mean by this, so I’ll list the reasons why I think the exact opposite is true:

A. User-defined commands’ syntax must be listed either in the source module in which they’re used, or in a header file specifically "included" in that module. On the other hand, UDFs and procedures can be called with no prototype at all, and can be in any module linked with the source (even modules you may not have been given!).

B. If a module uses "variant" calling sequences where the number or type of parameters determines the actions taken, the UDC’s syntax definition makes the variant behaviors much clearer.

C. Schmeebase language syntax is designed to be readable—that’s one of the main reasons for using it. UDCs by nature are much more readable. Of these syntaxes, which seems more lucid to you?

SCROLL(1,1,20,78,15)

or SCROLL SCREEN FROM 1,1 ; TO 20, 78 COLOR W+/B

Third, you say, "If ... your programmer or a third-party library defines its own commands, you won’t be able to easily replace either." Tom, are you arguing against the feasibility of using the Tom Rettig Library for Clipper? The language will be extended. The language has been extended. Now the question is whether our current extensions with UDFs are implemented in the best way, and my answer is no.
Using UDCs I can implement FoxPro syntax in dBase IV programs or dBASE IV syntax in Clipper programs. This solves some of "Tower of Babel" problem caused by differences in vendors' extensions, without inviting lawsuits. I don't have to make a hard choice between language vendors. Finally, when I prototype a UDC's syntax, that gives the compiler the opportunity to warn me when I use incorrect syntax to implement an extension. This alone makes UDCs the only way to go for me.

I can only assume you haven't used UDCs much, or these points would be obvious. Or perhaps you like Fox's SYS(2002[,1]) syntax as compared to SET CURSOR ON--OFF?

R. Russell Freeland is a freelance, free-spirited consultant and programmer in Fort Lauderdale, Florida. His company, Synergy Corp., publishes SCRIMAGE, a screen and menu design tool for dBase language programmers. He also claims to play a mean jazz trombone.