From Type to Type

Tamar E. Granor, Ph.D.

Changing data from one type to another is a common requirement. In older versions of VFP, you had to know a veritable cornucopia of functions to handle the various possibilities. A few recent additions make type conversion easier.

Data needs to be converted between types for all kinds of reasons, including formatting it for output, moving it from one database to another, and combining it with other data, for example, for use in an index.

Changing to character

Perhaps the most common conversion is from other types to character. Sometimes, data needs to be character for use in text output (whether it's some kind of report or an export file); at other times, it's in order to create a compound index.

In earlier versions, converting from other types to character involved a number of different functions including STR(), DTOC(), TTOC(), PADL() and PADR(). The right choice depended on the original data type and the desired result format. Today, most conversions can be done with either TRANSFORM() or CAST().

Converting numbers

When I started with FoxBase+, it wasn't hard to decide how to convert a number to a string; you used STR(). That was the only choice.

STR() accepts up to three parameters: the number to convert, the length, and the number of decimals to show. To be sure not to lose precision, you need to include the second and third parameters and make sure they're large enough. When converting a numeric field, that's not hard, just pass the field width and decimals. For non-field values, figuring out the necessary size was always difficult.

Numeric conversion got much easier in VFP 6 when the TRANSFORM() function was enhanced to make its second parameter optional. When you pass a number to TRANSFORM(), you get back a string that includes all the digits in the original. As Listing 1 shows, TRANSFORM() works with the various numeric types.

Listing 1. TRANSFORM() lets you convert numbers to character strings without providing the width and number of decimals.

```
? TRANSFORM(123.45) && displays 123.45
? TRANSFORM(123) && displays 12
? TRANSFORM($123.45) && displays $123.45
```

Converting dates and datetimes

The challenge for dates and datetimes has always been that the results change based on how things are set up. The DTOC() and TTOC() functions respect the current SET DATE and SET CENTURY settings.

When you want a result that's independent of SET DATE and SET CENTURY, you have two alternatives for dates, both of them going back a long, long way. DTOC() accepts an optional second parameter (listed as 1 in Help, but any numeric value will do) that indicates that the result should be in YYYYMMDD format. The DTOS() function, which stands for Date TO System, does the same thing, as Listing 2 demonstrates.

Listing 2. DTOC() depends on SET DATE, unless you pass the optional second parameter. DTOS() also provides a result that's independent of the SET DATE setting.

```
LOCAL dDate

dDate = DATE(2008,11,20)

SET DATE AMERICAN
? DTOC(m.dDate)  && displays 11/20/2008
? DTOC(m.dDate, 1)  && displays 20081120
? DTOS(m.dDate)  && displays 20081120

SET DATE BRITISH
? DTOC(m.dDate)  && displays 20/11/2008
? DTOC(m.dDate, 1)  && displays 20081120
? DTOS(m.dDate)  && displays 20081120

SET DATE JAPAN
? DTOC(m.dDate)  && displays 2008/11/20
? DTOC(m.dDate, 1)  && displays 20081120
? DTOS(m.dDate)  && displays 20081120
```

For datetimes, the same principles apply. TTOC() depends on SET DATE and SET CENTURY, unless you pass the optional second parameter. However, there are some additional variations. If you pass 2 as the second parameter, you get only the time portion of the value. In VFP 9, pass 3 for the second parameter, and get the datetime formatted in the XML standard way. Listing 3 demonstrates.

Listing 3. TTOC() offers four different results, depending on the second parameter.

```
tDateTime = DATETIME(2008, 11, 20, 3, 17, 49)
SET DATE AMERICAN
?TTOC(m.tDateTime)
   && displays 11/20/2008 03:17:49 AM
?TTOC(m.tDateTime,1)
   && displays 20081120031749
?TTOC(m.tDateTime,2)
   && displays 03:17:49 AM
?TTOC(m.tDateTime,3)
   && displays 2008-11-20T03:17:49
```

The right choice for converting dates or datetimes depends on what you want to do with the result. For display purposes, dependence on SET DATE and SET CENTURY is probably what you want. However, for use in compound indexes, the setting-independent format is more appropriate.

Finally, TRANSFORM() works for dates and datetimes. It gives the same results as DTOC() and TTOC() without the second parameter, as indicated by Listing 4.

Listing 4. TRANSFORM() works on dates and datetimes.

```
* Assume SET DATE AMERICAN and
* SET CENTURY ON
? TRANSFORM(m.dDate) && displays 11/20/2008
? TRANSFORM(m.tDateTime)
    && displays 11/20/2008 03:17:49 AM
```

For more control over the result, you can pass the optional second parameter to TRANSFORM(). For dates, there are several options. Pass "@D" to get the current SET DATE format or "@E" to use the British date format. In more recent versions, passing "@YS" or "@YL" formats the date in the current short or long date format, as specified in the Windows Control Panel.

TRANSFORM() almost anything

In addition to handling the various numeric types, date, and datetime, TRANSFORM() works for logical values as well, and doesn't even choke on character data. This makes it a one-stop shop for converting to character.

Padding to desired length

While TRANSFORM() provides a handy general-purpose conversion-to-character function, there is a case where another group of functions is easier to work with. When you need to convert to a character string of a specified length, especially if you want to pad on one side or the other with something other than spaces, use the PADx() functions: PADL(), PADR() and PADC().

The PADx() functions have two really useful features. First, like TRANSFORM(), they accept data of almost any type and do an implicit conversion to character. Second, they have an optional third parameter that specifies the pad character. By default, the result is padded with spaces. However, these functions can pad with anything you want, so you can use them, for example, to build strings of digits with leading zeroes, or to create "trailers" of dots as you'd find in a table of contents or index. As the examples in Listing 5 show, PADL() pads on the left, PADR() pads on the right, and PADC() pads on both sides to put the data provided in the middle of the result.

Listing 5. The PADx() functions not only convert the first parameter to character, but can pad with any character.

```
? PADL(324, 10, "0")
  && results in "0000000324"
? PADR("Chapter 1", 25, ".")
  && results in "Chapter 1................"
? PADC("Centered", 20, "/")
  && results in "//////Centered//////"
```

Converting to other types

Most of the conversions you need for types other than characters involve either converting from character to the other type, or converting between similar types, such as from datetime to date.

As with converting to string, VFP has a whole collection of functions to handle other type conversions. Table 1 lists some of the functions that go from one type to another.

Table 1. VFP has lots of type conversion functions that don't result in character data.

Original type	New type	Function
Character	Date	CTOD()
Character	DateTime	CTOT()
Character	Numeric	VAL()
Currency	Numeric	MTON()
Datetime	Date	TTOD()
Numeric	Currency	NTOM()

VFP 9 introduced the CAST() function, which provides a single way to convert between any compatible data types. CAST() accepts a single parameter with a rather unusual format and returns a value of a specified type. The syntax for CAST()'s parameter is shown in Listing 6.

Listing 6. The CAST() function accepts a single parameter, but its format is unusual.

```
uExpression AS cType [(nSize [, nDecimals] )]
```

In other words, you specify an expression and type, and optionally a size. The format for the type and size is the same as in the CREATE TABLE or CREATE CURSOR command. Listing 7 shows some examples. CAST() can handle a wide range of types, not just the obvious combinations. For example, trying to convert a numeric value to a logical results in 0 being .F. and all other values being .T.; that seems as good a choice as any. CAST() does fail on some conversions such as converting a date to logical or numeric.

Listing 7. With CAST(), you can do all your conversions with a single function; you just have to know the type and size you want.

```
? CAST("123.45" as N(6,2))
  && results in numeric value 123.45
? CAST("$123.45" as Y)
  && results in currency value 123.4500
? CAST("^2008-2-8" as D)
  && results in date value 20080208
? CAST("^2008-2-8 15:01" as T)
  && results in datetime value 20080208150100
```

CAST() is especially useful in SQL-SELECT commands, where it lets you specify the type and size for result columns, as well as create memo fields on the fly.

TRANSFORM() or CAST()?

Both TRANSFORM() and CAST() can convert from other types to character, so which one should you use for that purpose? The answer depends on your needs. When you want a string of a particular size, use CAST(). When you need a string in a specific format, use TRANSFORM(). TRANSFORM() is also best when you don't know how big the result could be.

Creating dates and datetimes

One traditional use of CTOD() is creating dates on the fly. It's not unusual to see code like that in Listing 8 in older applications. This code, of course, depends on the SET DATE setting. There are two alternatives that don't depend on that setting.

Listing 8. In FoxBase and FoxPro, it was common to use CTOD() to build date constants.

```
dStart = CTOD("09/01/2008")
```

The first alternative is the curly brace (using "{" and "}") notation for date constants. Rather than having to convert a string to a date, you can simply write the date between curly braces. Beginning in VFP 5, you can also use the strict date notation to make date constants independent of the SET DATE setting. Strict date notation begins with a caret (^) and then includes the date in YMD format. Listing 9 shows strict date format; Listing 10 shows the example of Listing 8 using a date constant in strict date format.

Listing 9. Curly braces enclose date constants. Use strict date format to avoid dependence on SET DATE.

```
{^ YYYY-MM-DD}
```

Listing 10. Use strict date format for date constants rather than converting character values.

```
dStart = {^ 2008-09-01}
```

While the ability to write unambiguous date constants is very helpful, in VFP 6, we were given the ability to unambiguously specify any date. The DATE() function accepts three optional numeric parameters: the year, the month and the day. When you pass those parameters, the function returns the corresponding date.

Listing 11 shows the same example again, this time using DATE().

Listing 11. You can pass year, month and day to DATE() to construct a date value.

```
dStart = DATE(2008, 9, 1)
```

The two techniques work for datetimes, as well. You can specify a datetime constant in strict date format between curly braces, as in Listing 12. In addition, starting in VFP 6, DATETIME() accepts up to 6 parameters to specify the datetime value to create.

Listing 12. Strict date notation and curly braces work for datetime values as well as for dates.

```
tAppt = {^ 2008-09-01 15:37:29}
```

Listing 13. The DATETIME() function lets you create datetime values by passing appropriate parameters.

```
tAppt = DATETIME(2008, 9, 1, 15, 37, 29)
```

On the whole, the DATE()/DATETIME() approach to creating date and datetime values is a better choice in code, as it offers more flexibility. The parameters can be variables, expressions or fields, as well as constants, so it's easy to create one date, given another. For example, to take a birth date, and get this year's birthday for that person, you can use code like:

Listing 14. The DATE() function makes it easy to turn one date into another.

```
* Assume dBirth contains birth date
dBirthday = DATE(YEAR(DATE()), ;
                 MONTH(dBirth), DAY(dBirth))
```

Choose the right one

As this article indicates, there are a lot of ways to convert data from one type to another in VFP. Some are as old as Xbase itself, while others are newer.

For any given conversion task, there are likely to be several options. Find the one that's easiest to get right in the first place and easiest to maintain, and then use that approach every time you need to do that kind of conversion.