Chapter 1 IntelliSense

One thing VFP developers have long envied in other Microsoft development environments is IntelliSense. In VFP 7, this great productivity enhancement is finally available.

One of the most obvious differences between VFP 6 and other Microsoft development environments is IntelliSense. Type a statement such as “Dim SomeVariable As” in Visual Basic and a list of all of the types a variable can be defined as (data types as well as objects) immediately pops up. This is a great productivity booster for several reasons: You can quickly pick the one you want from the list, it avoids spelling mistakes, and you don’t have to always head for the manual or Help file to figure out the syntax for a little-used command. VFP developers have felt shortchanged ever since IntelliSense was introduced.

Fret no more! VFP 7 introduces IntelliSense to VFP developers, and makes up for the long wait by providing a better version of IntelliSense than other products have. Once you start appreciating how much more productive IntelliSense makes you, you’ll wonder how you ever coded without it.

What IntelliSense offers

IntelliSense is only one word, but it encompasses several behaviors, including:

- Automatic keyword completion
- Command and function syntax tips
- Lists of members, values, and most recently used files

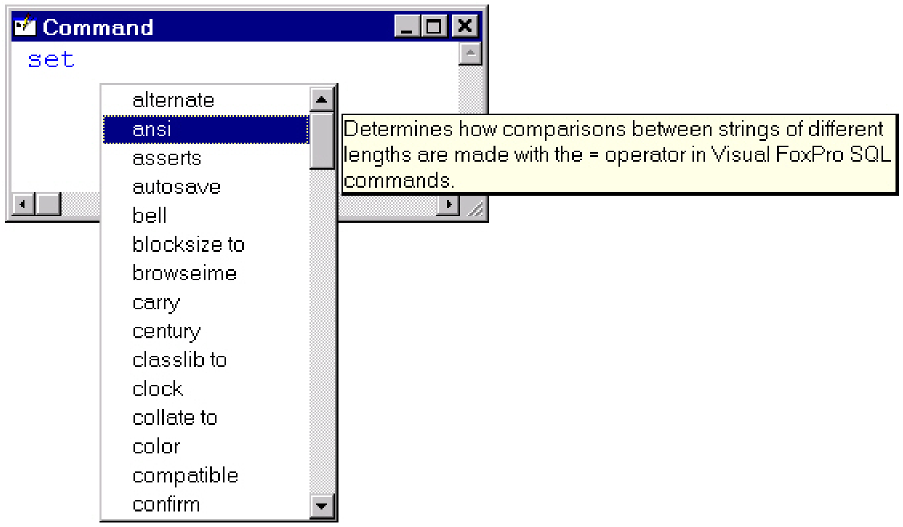
Automatic keyword completion

Xbase descendents such as FoxPro have always supported two ways to enter keywords, such as command and functions: using the full, official keyword (such as Browse) or using the first four or more characters (such as Brow). However, veteran developers will tell you that while it’s fine to use “Repl” in the Command Window, you really should use the full “Replace” in code for clarity.

VFP 7 provides the best of both worlds: You can now type just enough of a keyword to make it distinct, and then press Space or Enter in the case of a command or “(” in the case of a function, and VFP completes the keyword for you. For example, if you type “modi” and press Space, VFP replaces it with “MODIFY”. Because some keywords start with the same set of characters, be sure to type enough to distinguish the keyword from any others. If you want to use the MessageBox() function, you can’t just type “mess(”; VFP expands that to “MESSAGE(”.

Some commands actually consist of more than one keyword; examples include Alter Table, Report Form, and Open Database. Since the first keyword must always be followed by the second, VFP automatically adds that keyword as well. This can take a little getting used to; for a while, you'll probably find yourself typing "OPEN DATABASE DATABASE MyData" because you didn't notice that VFP automatically inserted "DATABASE" as soon as you pressed Space after typing "open".

"Set" is the first keyword in a long list of Set commands. When you type "set" and press Space, IntelliSense displays a list of each of the keywords that can follow "set", such as "deleted" and "exact" (see **Figure 1**). Some commands include the word "to", such as Set Classlib To; as you'll see later in this chapter, you can choose whether IntelliSense automatically inserts the "to" or not.



Determines how comparisons between strings of different lengths are made with the = operator in Visual FoxPro SQL commands.

***Figure 1**. IntelliSense displays a list of every possible Set command.*

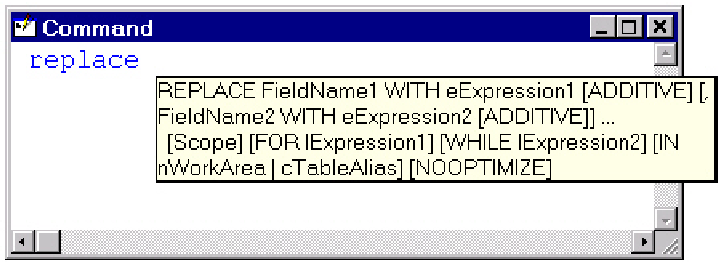
VFP doesn't complete keywords in the middle of a statement. For example, if you type the While clause of a Replace command as "whil", that's how it stays; VFP won't expand it to "WHILE".

Notice the case VFP uses; even if you type the entire word "modify", VFP replaces it with "MODIFY". This probably won't bother long-time Xbase developers, who expect keywords to be in uppercase, but this is very disconcerting to those who prefer lowercase or even "camel" case (such as "TableUpdate()"). Fortunately, you can control the case used for keyword expansion through the IntelliSense Manager, which is discussed in the "Configuring IntelliSense" section of this chapter.

There may be times when you don't want VFP to expand a keyword. Pressing Ctrl-Space rather than Space at the end of a command suppresses expansion for the command. To undo an expansion, press Ctrl-Z twice (the first removes the replacement and the second restores the original keyword, leaving it selected), and then press End to move the cursor to the end of the keyword.

Command and function syntax tips

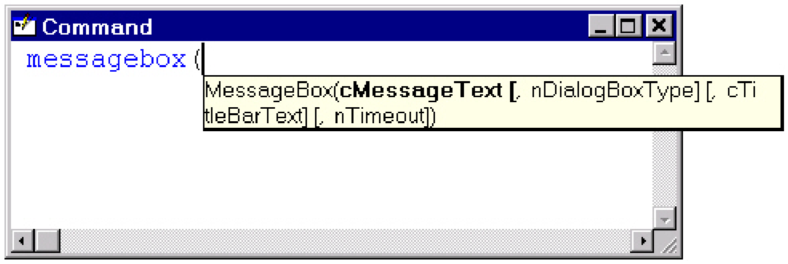
After typing “repl” and pressing Space, you may notice something besides the keyword completion: a tip window showing the complete syntax of the Replace command (see **Figure 2**). This feature, which Microsoft calls “Quick Info,” is a great productivity booster; how many times do you find yourself bringing up the VFP Help file because you can’t quite remember the exact syntax for Alter Table or whether the string to search is the first or second parameter in At()?



```
replace
REPLACE FieldName1 WITH eExpression1 [ADDITIVE] [,
FieldName2 WITH eExpression2 [ADDITIVE]] ...
 [Scope] [FOR lExpression1] [WHILE lExpression2] [IN
nWorkArea | cTableAlias] [NOOPTIMIZE]
```

***Figure 2**. Quick Info saves you a trip to the Help file by displaying the complete syntax for commands and functions.*

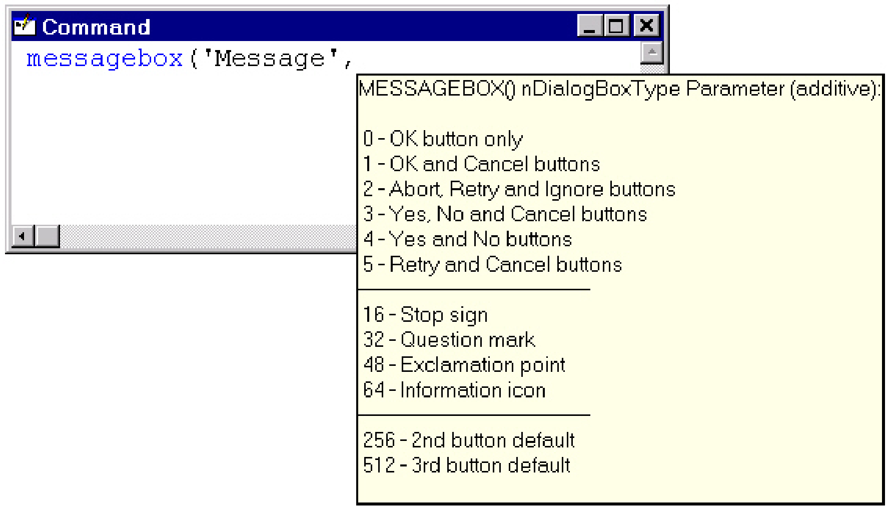
The tip window stays on-screen as you continue to type the rest of a command or enter the parameters of a function, method, or event, only disappearing when you complete the function with “)”, move to another line (such as pressing Enter), or press Backspace (in that case, it reappears when you move the cursor to the next element of the command). You can manually hide it by pressing Esc and manually display it with Ctrl-I or the Edit | Quick Info menu item. It’s especially useful for functions, methods, and events: The parameter you’re currently typing appears in bold (see **Figure 3**).



```
messagebox (
MessageBox(cMessageText [, nDialogBoxType] [, cTi
tleBarText] [, nTimeout])
```

***Figure 3**. The parameter you’re currently typing appears in bold in the Quick Info tip window.*

For some functions, the tip window displays information about the values for a specific parameter. For example, the second parameter for MessageBox() is an additive value for the buttons and icon for the dialog. As you can see in **Figure 4**, IntelliSense shows the complete list of valid values for this parameter. Other functions accept only one of a list of predefined values for some parameters. For instance, the second parameter in DBGetProp() specifies the type of data object (connection, database, field, table, or view), and the list of values for the third parameter, the property, varies with the type of object (for example, DefaultValue is only available for fields). For the type parameter, IntelliSense displays a list of the object types; choose the desired type from the list and IntelliSense inserts it, complete with quotes, in the command line. The list of values displayed for the property parameter includes only those applicable to the selected type; again, you can choose the property from the list to add it to the command line.



```
messagebox('Message',
```

MESSAGEBOX() nDialogBoxType Parameter (additive):

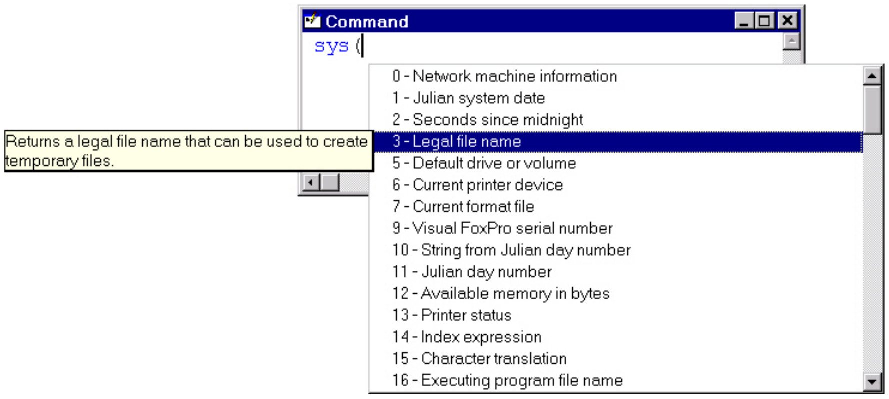
0 - OK button only
1 - OK and Cancel buttons
2 - Abort, Retry and Ignore buttons
3 - Yes, No and Cancel buttons
4 - Yes and No buttons
5 - Retry and Cancel buttons

16 - Stop sign
32 - Question mark
48 - Exclamation point
64 - Information icon

256 - 2nd button default
512 - 3rd button default

***Figure 4**. The tip window for some functions shows detailed information about the current parameter.*

The SYS() function is a special case of this. Although there's only one keyword, it really consists of a large number of functions, with the specific function determined by the first parameter. As **Figure 5** shows, IntelliSense displays a list for this parameter showing not only the numeric values but also the meaning for each one. Once you've selected which one you want, the tip window shows the syntax for the rest of the function call.



```
sys(
```

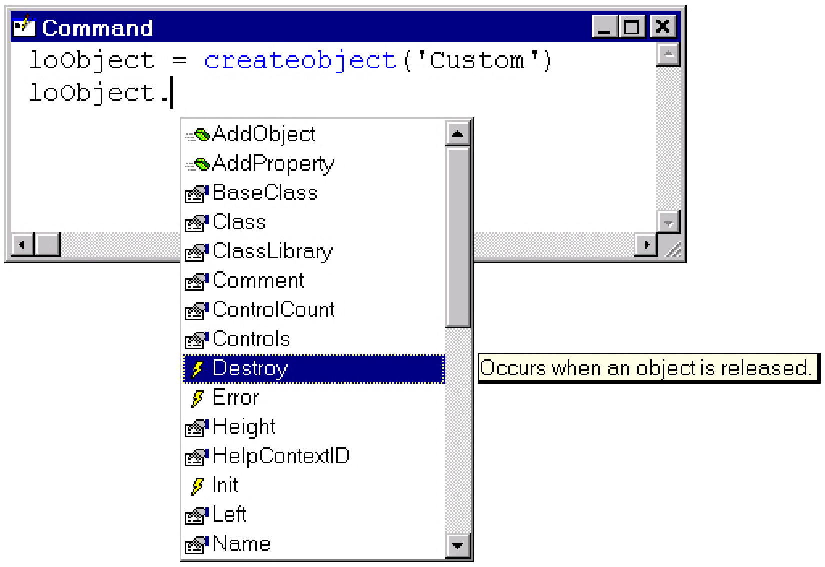
Returns a legal file name that can be used to create temporary files.

0 - Network machine information
1 - Julian system date
2 - Seconds since midnight
3 - Legal file name
5 - Default drive or volume
6 - Current printer device
7 - Current format file
9 - Visual FoxPro serial number
10 - String from Julian day number
11 - Julian day number
12 - Available memory in bytes
13 - Printer status
14 - Index expression
15 - Character translation
16 - Executing program file name

***Figure 5**. IntelliSense makes it easy to figure out which value to pass SYS() for which function.*

List of members

In today's object-oriented code, the IntelliSense List Members feature does more to save you typing than any other. When you enter the name of an object and press ".", VFP displays a list of all members (properties, methods, events, and contained members) of the object; an icon indicates what type of member it is. **Figure 6** shows an example, displaying the members of a newly instantiated Custom object.



```
loObject = createobject('Custom')
loObject.
```

Occurs when an object is released.

***Figure 6**. IntelliSense's List Members feature displays the members of an object.*

You can navigate through the list using the up and down arrow keys, the Page Up and Page Down keys, or using incremental searching by typing the first few letters of a member's name (the letters you type also appear after the period in the command line). As you do so, a ToolTip box shows the description of each member, once again saving you a trip to the Help file.

IntelliSense adds the selected member's name to the command line and hides the list if you press Tab (the cursor appears right behind the member name), Enter (the cursor moves to the next line), or some non-alphabetical character such as Space, ".", "=", or "(" (the character is added at the end of the member name). Pressing Home, End, Esc, or the left or right arrow (if they move the cursor before the period or past the end of what's already typed) hides the list without adding the member's name to the command line. You can manually display the list by pressing Ctrl-J or choosing the Edit | List Member menu item.

List Members works in object hierarchies too. Imagine how much you'd have to type to programmatically change the Caption of the header in the third column of a grid on page 2 of a page frame in a form:

```
Thisform.PageFrame1.Page2.Grid.Column3.Header.Caption = 'some value'
```

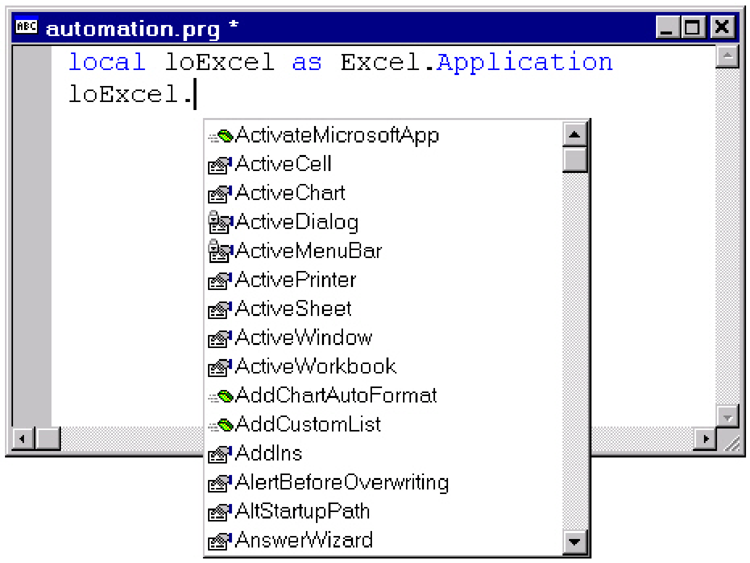
Oops, the name of the grid is actually grdCustomers, so when you run this code, it'll bomb. List Members saves you from both making a spelling mistake and having to type all that code; after you type the period following Thisform, you can select PageFrame1 from the member list and press "." to display the member list for the page frame, choose Page2 and press "." to display the member list for that page, and so on until finally you choose Caption for the header and press "=" to close the list.

List Members doesn't just work with native VFP objects; it works with any instantiated object, such as a TreeView ActiveX control you've added to a form or a COM object you've instantiated with CreateObject(). Unlike Visual Basic's IntelliSense, VFP's version supports member lists for objects in collections (such as Excel's Cells collection) rather than stopping at the collection itself. Even better, it can work with objects you haven't instantiated yet. That leads to the next topic.

Early binding

VFP is a late-binding client of COM objects. That means during development (in the editor and at compile time), VFP has no idea what kind of object you're instantiating in a call to CreateObject(). Only when that code is actually executed does VFP figure out what the properties, events, and methods (PEMs) are for the object. It does that by reading the type library for the object; for a discussion of type libraries, see Chapter 12, "Building World-Class COM Servers in VFP 7." Early-binding clients, such as Visual Basic, read the type library at development time instead. This provides several benefits, but the one related to IntelliSense is the ability to display the members of the COM object in the editor. Say, that sounds just like the List Members feature of VFP 7. The problem is that since VFP doesn't know you've instantiated an Excel object in your code, how can it display Excel's members?

To solve this problem, Microsoft added the As clause to the Local and Public commands (it's also available in Parameters, LParameters, Function, and Procedure declarations, but that's for a different purpose; see the "Strong typing" section of Chapter 12, "Building World-Class COM Servers in VFP 7"). This clause allows you to specify the class of object a variable will contain at run time. When you type the variable name followed by a period in the VFP editor, IntelliSense sees that the variable has been "typed" as a class, gets the members for that class (reading the type library in the case of a COM object), and displays them in the List Members list. **Figure 7** shows the members of the variable loExcel, which has been declared as being an Excel.Application object. Notice there's no CreateObject() statement in this code, so it won't actually work at run time. The As clause doesn't instantiate the object—you still have to do that in code—it just tells IntelliSense how to treat a variable in the editor.



```
local loExcel as Excel.Application
loExcel.
```

***Figure 7**. IntelliSense can show COM members if you declare a variable "as" the COM object.*

After you type the As keyword and press Space, a list of types appears. This list includes VFP data types (such as Integer and Numeric), base classes (including Form and Custom), registered type libraries (for example, ADODB and Excel), your own classes, and Web Services. As with many things in VFP, the list of types is defined in a table, so you can add type libraries, classes, Web Services, and other types. This is discussed in detail later in this chapter in the "Configuring IntelliSense" section.

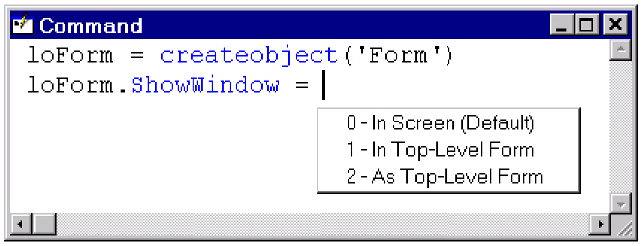
There's one slight consequence of this cool new feature: Spaces are no longer supported between variable names in Local and Public statements, such as:

```
local MyVar1 MyVar2
```

Since most people didn't know you could separate variables with spaces (the Help file has always shown the correct syntax using commas), this likely won't break too much code.

Values list

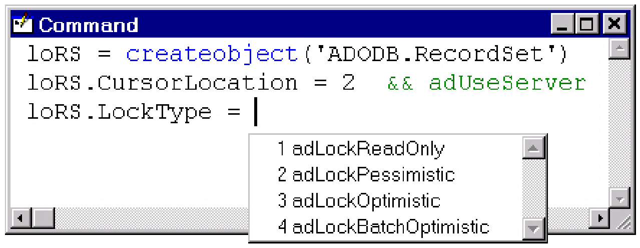
Some object properties accept only a small range of values. For example, Form.AutoCenter is Logical, so the only acceptable values are True and False. Object.BackColor accepts a fairly wide range of numeric values, but these values each represent a color, of which there is a relatively small number of commonly used ones. Unfortunately, figuring out which number to use for the desired color isn't easy. IntelliSense makes it easy to select the correct value for properties like these. When you type "=" after some property names, IntelliSense displays a list of the valid values for those properties; see **Figure 8** for an example.



```
loForm = createobject('Form')
loForm.ShowWindow = 
```

***Figure 8**. The List Values feature makes it easy to select the correct value for a property.*

Logical properties like AutoCenter and AlwaysOnTop have a list with True and False displayed. Those numeric properties that have an enumerated list of values, like BorderStyle, ScrollBars, and WindowType, display the possible values and their meanings. For properties with more complex values, IntelliSense displays an appropriate dialog, such as a color picker for properties representing colors (such as BackColor, FillColor, and ForeColor) and an Open Picture dialog for properties containing image file names (such as Icon and Picture). As with the List Members feature, List Values supports COM objects as well as native ones; IntelliSense displays a list of values for those properties with enumerations defined in the type library. **Figure 9** shows how useful this feature is with an ADO RecordSet, which uses enumerated values for many properties. Notice the line of code above the one showing the list; IntelliSense not only inserted the value but a comment showing the enumerator name for the value, making your code much easier to follow.



```
loRS = createobject('ADODB.RecordSet')
loRS.CursorLocation = 2  && adUseServer
loRS.LockType = 
```

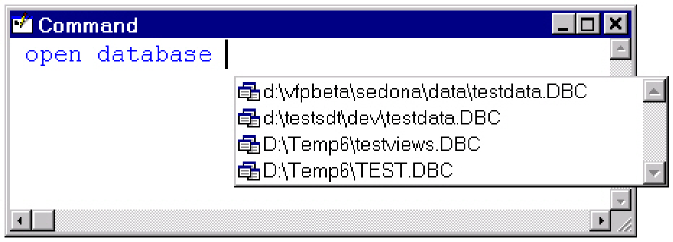
***Figure 9**. List Values even works for enumerated properties of COM objects.*

Some properties that you may think should have a values list or dialog don't. For example, although they contain image file names, IntelliSense doesn't display an Open Picture dialog for DragIcon, MouseIcon, and OLEDragPicture. You may be surprised to not see a font dialog for the FontName property. You'll see later in this chapter how you can script IntelliSense so you could add this functionality if you wish.

The List Values feature doesn't have its own menu item or hot key; the List Members item and hot key (Ctrl-J) serve the same function.

Most recently used file list

Some VFP commands open or process files. Examples include all the Modify commands (such as Modify Program), Open Database, and Report Form. IntelliSense presents a most recently used (MRU) list for these commands, making it easy to use a file you previously worked with. **Figure 10** shows this list in action. You also get an MRU list of directories with the CD command. Note that this feature is only supported in the Command Window. See the “Auto MRU (Most Recently Used) Files” topic in the VFP Help file for a complete list of commands with MRU lists.



```
open database
```

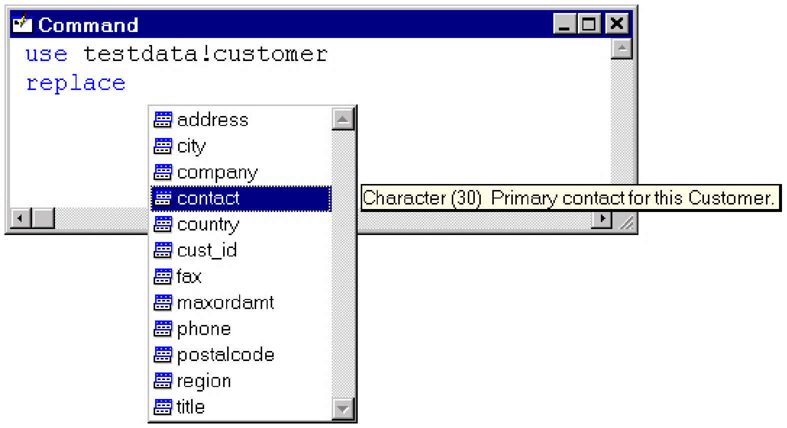
d:\vfpbeta\sedona\data\testdata.DBC
d:\testsdt\dev\testdata.DBC
D:\Temp6\testviews.DBC
D:\Temp6\TEST.DBC

Figure 10*. The MRU list IntelliSense displays makes it easy to select a file you used before.*

The View page of VFP’s Options dialog has a Most Recently Used list contains option that gives you control over how many items IntelliSense will display for you.

Table, field, and variable lists

IntelliSense extends the MRU list for the Use command: In addition to tables you’ve opened before, it lists the tables and views in the current database. The list has a value tip window showing the comment for the selected table or view. IntelliSense also extends the values list for the Replace, Modify Memo, and Modify General commands: If a cursor is open in the current work area, it displays a list of fields in that cursor. A value tip window shows the data type, size, caption, and comment for the selected field; see **Figure 11** for an example. If you type “m.” in the Command Window, IntelliSense displays a list of declared variables. The value tip for each variable shows its current value. As with the MRU list, none of these features works in an editor window.



```
use testdata!customer
replace
```

address
city
company
contact
country
cust_id
fax
maxordamt
phone
postalcode
region
title

Character (30) Primary contact for this Customer.

Figure 11*. IntelliSense displays a list of fields in the current cursor for some commands.*