White Paper

Leveraging the Power of IntelliSense

By Travis Vandersypen, Senior Software Developer, EPS Software Corporation

Abstract

This whitepaper explores the IntelliSense capabilities of Visual FoxPro 7 and how you can extend the functionality in the event additional functionality is required or may be helpful. We will discuss what the IntelliSense features of Visual FoxPro 7 offer to us as well as how to configure the native features of IntelliSense to more of what we need or desire.

Contents

What Does IntelliSense Offer?

In short, IntelliSense provides a means by which a developer can become more productive. This productivity can be increased through the following features:

- Keyword Completion
- Syntax Tips
- Member Lists
- Value Lists
- Most Recently Used (MRU) Lists

Keyword completion comes in particularly useful by allowing a developer to enter some minimum amount of unique characters, “subs” for “SubStr()” as an example, and have Visual FoxPro automatically complete typing the rest as soon as the developer enters a “(“ in this case. The “(“ triggers the IntelliSense features of Visual FoxPro 7 and will automatically replace, for our example, “subs” with “SubStr.” However, the case can also be manipulated and determined beforehand, but we’ll get into that later. So, in effect, Keyword Completion means a developer shouldn’t have to type as much to create the same amount of code that he or she created in earlier versions of Visual FoxPro.

Syntax Tips, one of my favorite features of IntelliSense, provide a quick “tip” on the syntax of a given command. Visual FoxPro has such a robust function library that it is difficult for anyone to remember the exact syntax of every function. Syntax tips “prompt” our memory for us, keeping us from having to constantly go look up the syntax of a command in the help file.

Member Lists are a particularly useful feature to many of us. Since Visual FoxPro 7 is an object-oriented programming language, we deal with objects and classes on a regular basis. However, it can often times be difficult to remember every property, event, and method on every object or class. That’s where member lists come in. When you enter the name of an object or the name of a variable which is an object reference and type the “.”, Visual FoxPro will bring up a list showing you all the properties, events, and methods on that object. This doesn’t only work for objects created from Visual FoxPro classes, but it will also work on COM objects as well. This means we don’t have to worry about possibly mistyping the name of a property, event, or method on an object any longer.

Value Lists help out when a property has a “pre-defined” set of values that can be assigned to it, or when a function can accept certain values as a parameter. Let’s examine the Sys() function as an example. When we enter “Sys(“, Visual FoxPro brings up a list of all the possible values that could be passed into the Sys() function as well as a brief description of what those values do. This makes it much easier than having to open the help file to figure out which number to pass to the Sys() function to get a certain behavior.

Most Recently Used Lists are nothing more than a recent history for certain commands within Visual FoxPro. For instance, if you enter “Modi Form Test”, after you save your changes, if you

enter “Modi Form,“ Visual FoxPro displays a list of all your most recently used forms for you, at the top of which will be “Test.”

But these features are common to every Microsoft programming language that has IntelliSense. There are several things, however, about the Visual FoxPro version that makes it special. First, it’s extensible. If it doesn’t do what you need it to, then you can write a process that will and extend it that way. Second, it’s configurable. You can specify the case of the IntelliSense items, whether or not you want IntelliSense to work automatically, manually, or not at all, etc. And last, but not least, Visual FoxPro’s version of IntelliSense expands beyond the “()”, unlike some other programming languages that we won’t mention (like Visual Basic).

Configuring IntelliSense

One of the greatest features about the Visual FoxPro implementation of IntelliSense is the IntelliSense Manager. This is a form that enables you to customize how IntelliSense works and provides a visual wrapper around the FoxCode.dbf table which is the table that drives the IntelliSense features of Visual FoxPro 7. You can access this form either through the **Tools** menu, or by typing “Do (_CodeSense)” in the Command Window. In either case, Visual FoxPro will start the IntelliSense Manager application shown in **Figure 1** below.

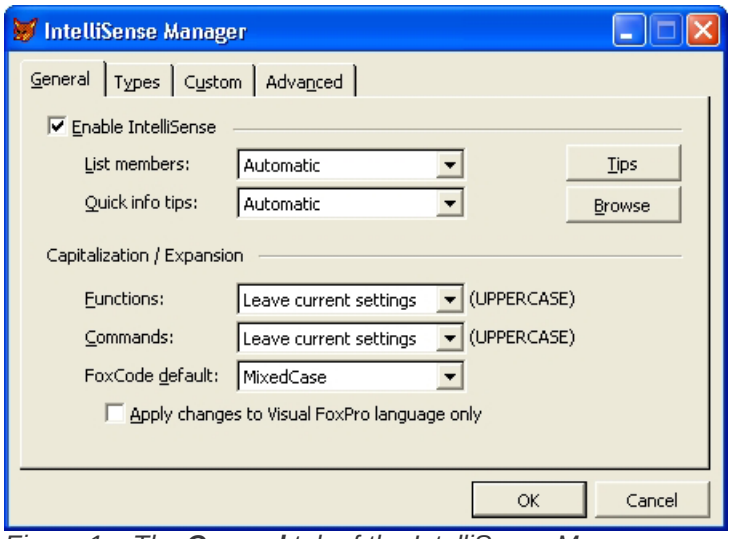

*Figure 1 – The **General** tab of the IntelliSense Manager.*

General Options

We can see from **Figure 1**, that many of the “generic” features of IntelliSense can be managed from the **General** tab of the IntelliSense Manager. We can specify whether or not we even wish to have IntelliSense turned “on” by checking or un-checking the check box labeled “Enable IntelliSense”. We can also specify whether we want the Member Lists and Syntax Tips to show up automatically, manually or not at all by selecting a value of “Automatic,” “Manual” or “Disabled” from the combo boxes labeled “List members” and “Quick info tips.” In addition, we

have the ability to specify the case of functions and commands by selecting a value from the combo boxes labeled “Functions” and “Commands.”

Types Options

The second tab of the IntelliSense Manager, labeled **Types**, contains a list of variable types that may be assigned to variables for which IntelliSense will automatically include in the list that shows up when you type something like “LOCAL loObject As” as shown in **Figure 2**.

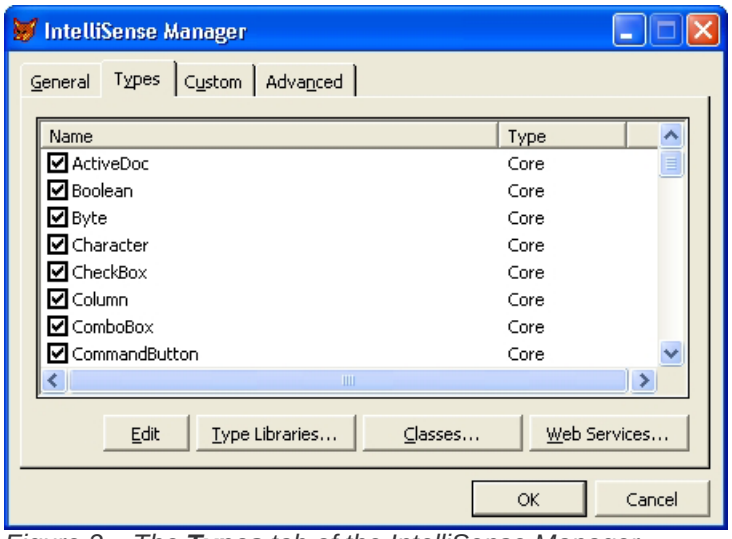

*Figure 2 – The **Types** tab of the IntelliSense Manager.*

One important item to point out here is that you can add additional types as needed. However, these types can be COM objects, Web Services or even your own Visual FoxPro classes. It’s really up to you. The nice thing about registering COM object and Web Services as types is that Visual FoxPro’s IntelliSense will automatically provide the Member Lists for you for those variables.

Custom Options

The **Custom** tab of the IntelliSense Manager, shown in **Figure 3**, enables you to add your own command abbreviations that can be expanded out. For instance, looking at **Figure 3**, we can see that when we type “MC,” the “MC” will be expanded out to “Modify Command,” which saves on the amount of typing we would otherwise have to perform. However, for more complex tasks, you can also create scripts that are to be executed rather than being replaced with something, but we’ll get into that later.

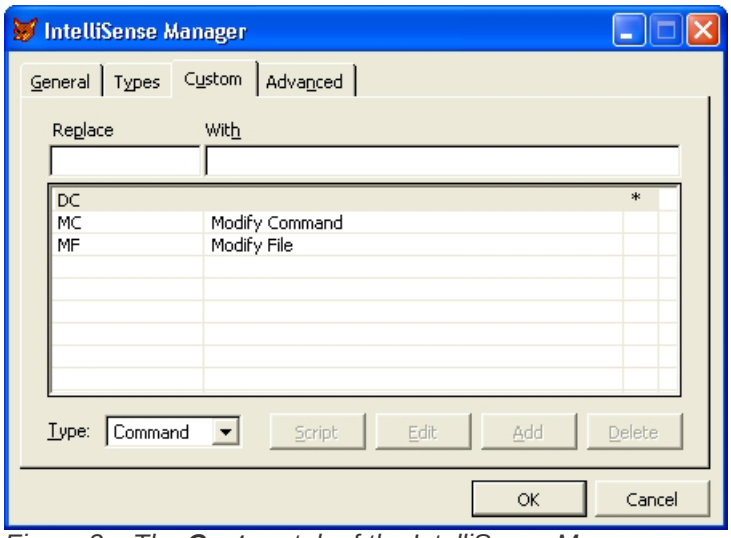

*Figure 3 – The **Custom** tab of the IntelliSense Manager.*

Advanced Options

Finally, the last tab of the IntelliSense Manager, labeled **Advanced** and shown in **Figure 4** below, provides an entry-point into two other areas of the IntelliSense features within Visual FoxPro 7. One is influencing the IntelliSense behavior in a more refined sense, and the other is maintenance.

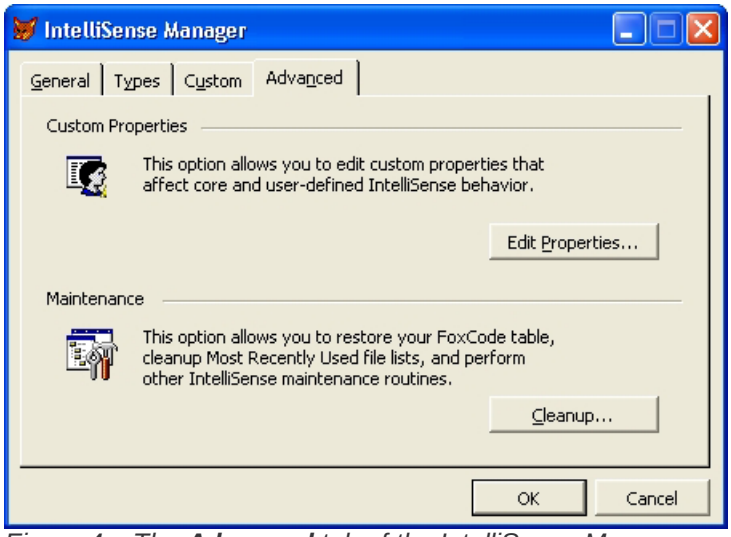

*Figure 4 – The **Advanced** tab of the IntelliSense Manager.*

Clicking **Edit Properties…** displays the Custom Properties Dialog, shown in **Figure 5**, while clicking **Cleanup…**displays the Cleanup and Maintenance Dialog, shown in **Figure 6**.

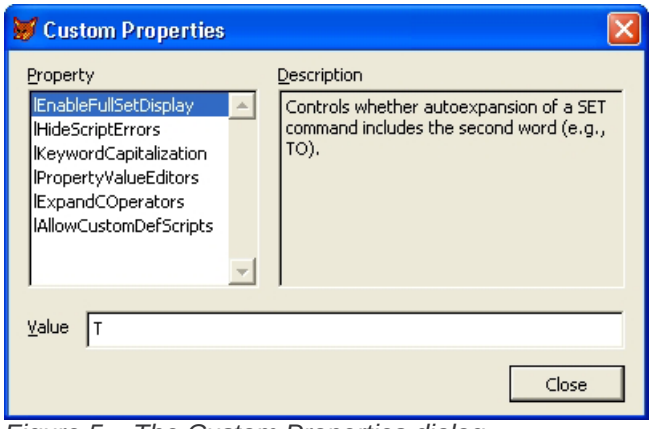

Figure 5 – The Custom Properties dialog.

From **Figure 5**, you can see a whole set of options available to us. The first, "lEnableFullSetDisplay" has control over whether Visual FoxPro's IntelliSense automatically expands out the full SET command in cases like "SET PATH TO." If this property is set to True, Visual FoxPro automatically expands "SET PATH" to "SET PATH TO".

Other notable options include the "lPropertyValueEditors" and "lExpandCOperators" properties. The "lPropertyValueEditors" property, when set to True, displays various dialogs like the Color Dialog when attempting to programmatically assign a color value to a color property, like BackColor. The "lExpandCOperators" property will expand statements like "i++" to "i = i + 1".

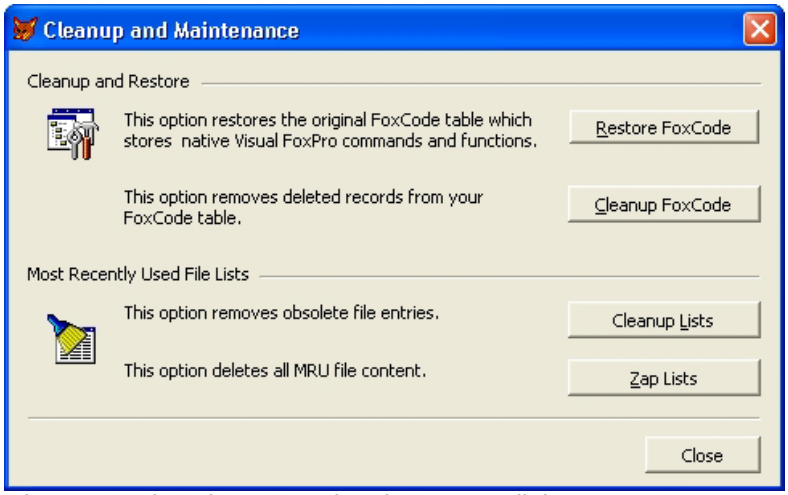

Figure 6 – The Cleanup and Maintenance dialog.

The Cleanup and Maintenance Dialog, shown in **Figure 6**, provides the maintenance features necessary to keep the IntelliSense features in working order. It will clean up any entries you may have made to the FoxCode.dbf table and well as cleaning up or clearing your MRU lists.

Extending Visual FoxPro's IntelliSense Capabilities

As with most other things in Visual FoxPro 7, you can extend the functionality of the IntelliSense. This can be done through many means: you can add your own types to the Types list, you can add custom commands to expand out and you can even write your own scripts to execute. In this section, we will concentrate on writing your own custom scripts, as those will probably be of the most interest to you.

Writing Custom Scripts

To create a new script, open the IntelliSense Manager, go to the "Custom" tab and type in an abbreviation that you want Visual FoxPro to look for to tell it when to execute your script. As an example, we will develop 2 scripts for documenting code changes. So, enter "BC" into the text box labeled "Replace". Now, click the "Add" button. You should now see a new item in the grid showing up as "BC". Now, click the button labeled "Script". This will bring up a text editor. You should already see a line in there that says "LPARAMETERS oFoxCode". You'll need to change the text to read like the code shown below.

```
LPARAMETERS oFoxCode

If oFoxcode.Location = 0
   Return "BC"
EndIf

oFoxcode.ValueType = "V"

LOCAL lcReturn

lcReturn = "*{ Begin changes made by Travis " + Transform(DateTime())

Return lcReturn
```

I know what you're saying: "What is oFoxCode?" Well, oFoxCode is a parameter that is passed in to every script that gives you certain information such as which editor you're in, what the previous characters are, etc. The structure for the FoxCode object, taken from the Visual FoxPro help file, is listed in the table below.

Property	Description
Abbrev	Contents of the Abbrev field
Case	Contents of the Case field
Cmd	Contents of the Cmd field

Property	Description
CursorLocChar	This is a special character denoting the location to place the cursor after script (Default is "~" character)
Data	Contents of the Data field
DefaultCase	Default Case setting of FoxCode as derived from the Version record (Type = "V")
Expanded	Contents of the Expanded field
Filename	Name of file being edited
FullLine	Full text from the line being typed currently
Icon	Icon for use with items array
Items	Array for use in populating a dropdown list to be displayed as post script action. Requires ValueType = "L" Items[1,1] – text to display in list Items[1,2] – value tip for item The only required element is the first element for each row in the Items array. By default the array is sorted ascended to allow for incremental seeks. Users can use the ItemSort property to turn this off and use a natural sort of array.
ItemScript	Script for use with items array
ItemSort	Whether to sort items array (default = .T.)
Location	Type of editor being edited: 0 – Command Window 1 – Program 8 – Menu Snippet

Property	Description
	10 – Code Snippet 12 – Stored Procedure
Menuitem	The menu item selected if user is running script with ValueType="L". Can be used in follow up script.
ParamNum	Parameter number of the function for script call made within a function
Save	Contents of the Save field
Source	Contents of the Source field
Timestamp	Contents of the Timestamp field
Tip	Contents of the Tip field
Type	Contents of the Type field
UniqueId	Contents of the UniqueId field
User	Contents of the User field
UserTyped	Text the user typed. Does not include triggerkey (use FullLine instead for this).
ValueTip	Quick Info tip to display when ValueType = "T"
ValueType	Handler for post script action L – displays a drop-down list populated from the Items array V – displays Value T – displays a Quick Info tip from ValueTip

Now then, if we go back to the code listing and examine it, we can see that we set the ValueType property of the oFoxCode object to “V” to that Visual FoxPro will replace the “BC” that we enter with “*{ Begin changes made by …”. Also, notice the “If” statement checking the

Location property of the oFoxCode object. This test is making sure we are in some form of editor other than the Command Window, where it doesn't make sense to have comments for documentation.

To test this script, save it and then click **OK** in the IntelliSense Manager dialog. In the Command Window, type *BC* and then a space. Nothing should happen and that's the behavior we want. Now, if we type *Modi Comm* and press Enter in the Command Window, we should see a Program Editor. Now, type *BC* and a space. If everything worked correctly, "BC" should have been replaced with something like "*{ Begin changes made by Travis 3/22/2002 1:20:49 PM".

Now then, knowing when and where changes begin is great, but how about knowing when and where changes end? Let's add another script called "EC." Follow the exact same steps outlined above, except that when you get to the text editor; enter the text shown below.

```
LPARAMETERS oFoxCode

If oFoxcode.Location = 0
   Return "EC"
EndIf

oFoxcode.ValueType = "V"

LOCAL lcReturn

lcReturn = "*{ End changes made by Travis " + Transform(DateTime())

Return lcReturn
```

Summary

We have seen that although it may have taken the Visual FoxPro team a little while to implement IntelliSense into Visual FoxPro 7, they have, as always, done a magnificent job in providing us with a flexible solution that will meet our needs. Not only does it support the standard IntelliSense features common to every Microsoft programming language, it also goes above and beyond those features, offering us a great realm of possibilities. We have complete control over the IntelliSense features within Visual FoxPro 7 and if something isn't there that we need, we can simply add it. After having studied the features of Visual FoxPro's IntelliSense, I can honestly say that, in my opinion, this makes the purchase of Visual FoxPro 7 worthwhile by itself.