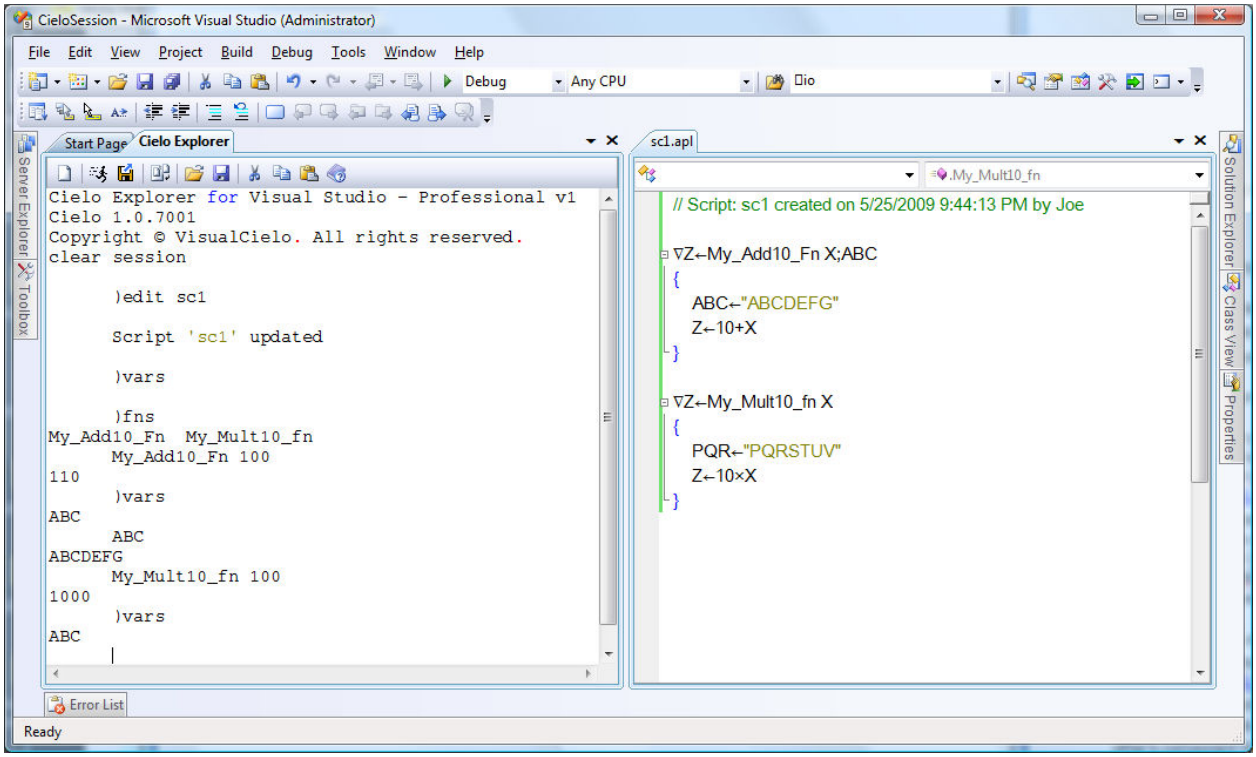
Variable Scoping in VisualAPL Operator Function Signatures

When a VisualAPL function using the ***operator*** (traditional APL) function signature is defined, variables which are named in the function header and separated by semi-colons (“;”), persist after the function call, whereas variables which are not named in the function header are local to the function in scope and do not persist after the function call.

The “My_Add10_Fn” is defined in the “sc1” Cielo Explorer script and uses the operator function signature. This function names the “ABC” variable in the function header, so that after this function is called from the Cielo Explorer session, the “ABC” variable now exists in the Cielo Explorer session.

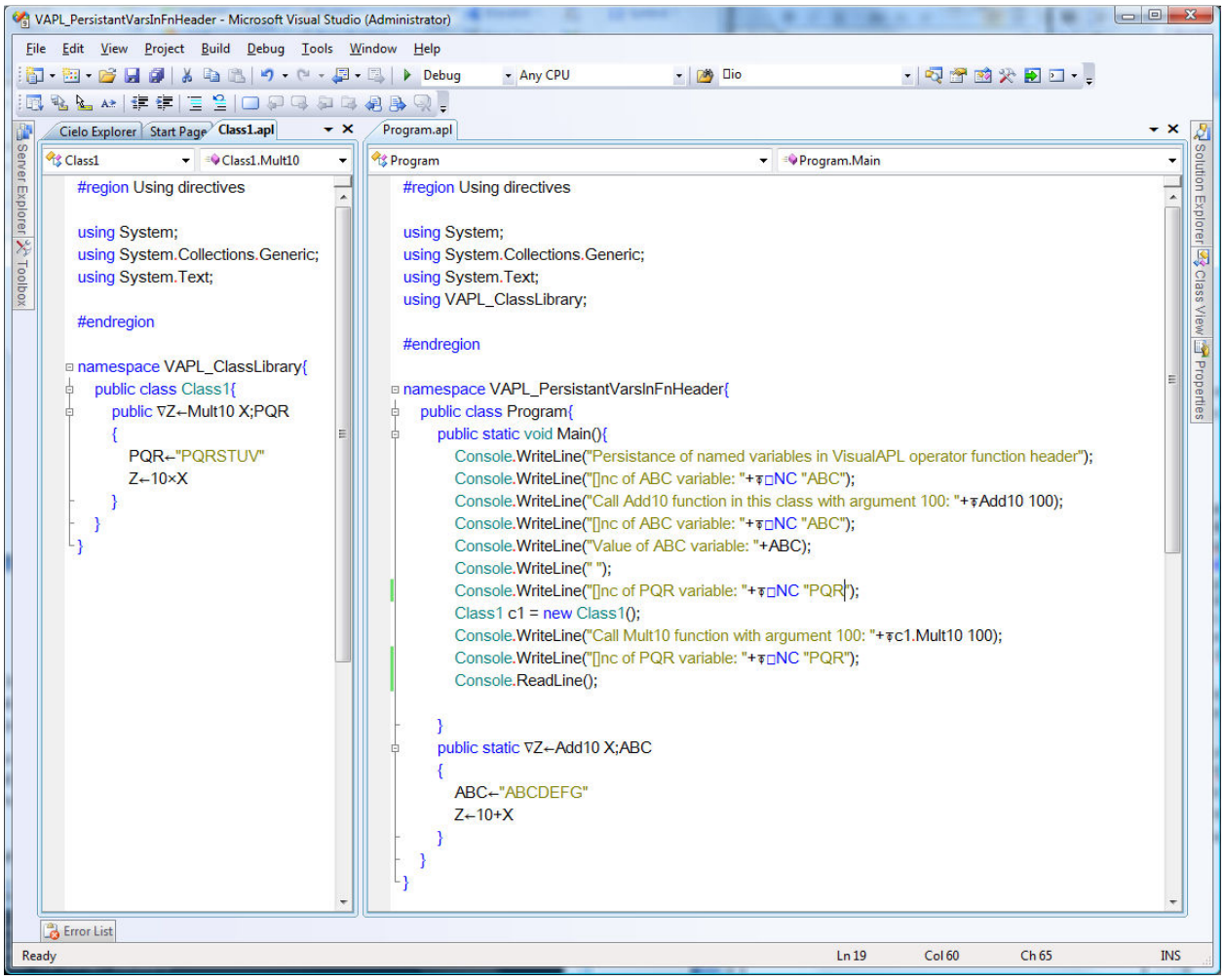
Conversely, the “My_Mult10_fn” is also defined in the “sc1” Cielo Explorer script and also using the operator function signature. This function does not name the “PQR” variable in the function header, so that after this function is called in the Cielo Explorer session, the “PQR” variable does not exist in the Cielo Explorer session.



The OOP (Object-oriented programming) encapsulation principle remains in force with respect to named variables in the function header of a VisualAPL function using the operator function signature. Specifically, only public members of a class are available to another class.

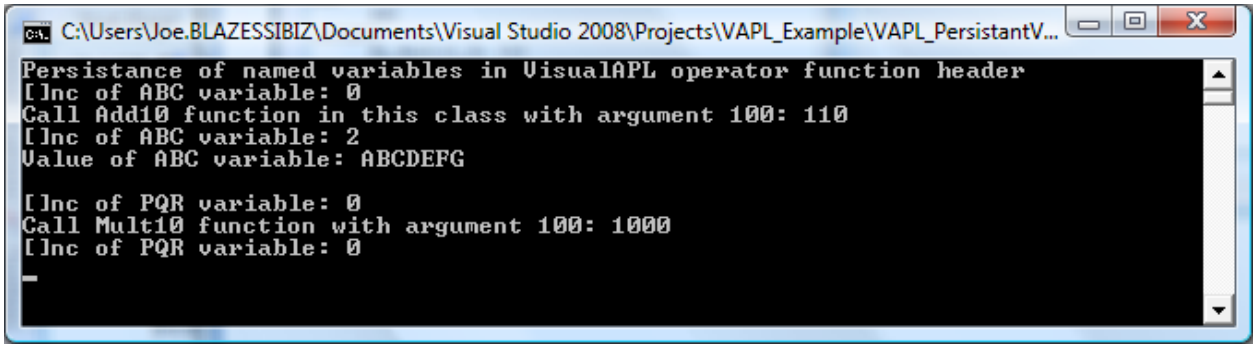
If the function being called is contained in the same class as the calling function, then the named variable in the header of a VisualAPL operator function will persist. However if the function being called is in a different class than the calling function, then the named variable in the header of a VisualAPL operator function will not persist, or even exist, within the calling class.

In the following VisualAPL console project, observe that the Add10 function is contained in the same class as the calling function, Main(), but the Mult10 function is contained in a different class than the calling function Main(). Both the Add10 and the Mult10 functions name a variable in the function header and both use the VisualAPL operator function signature.

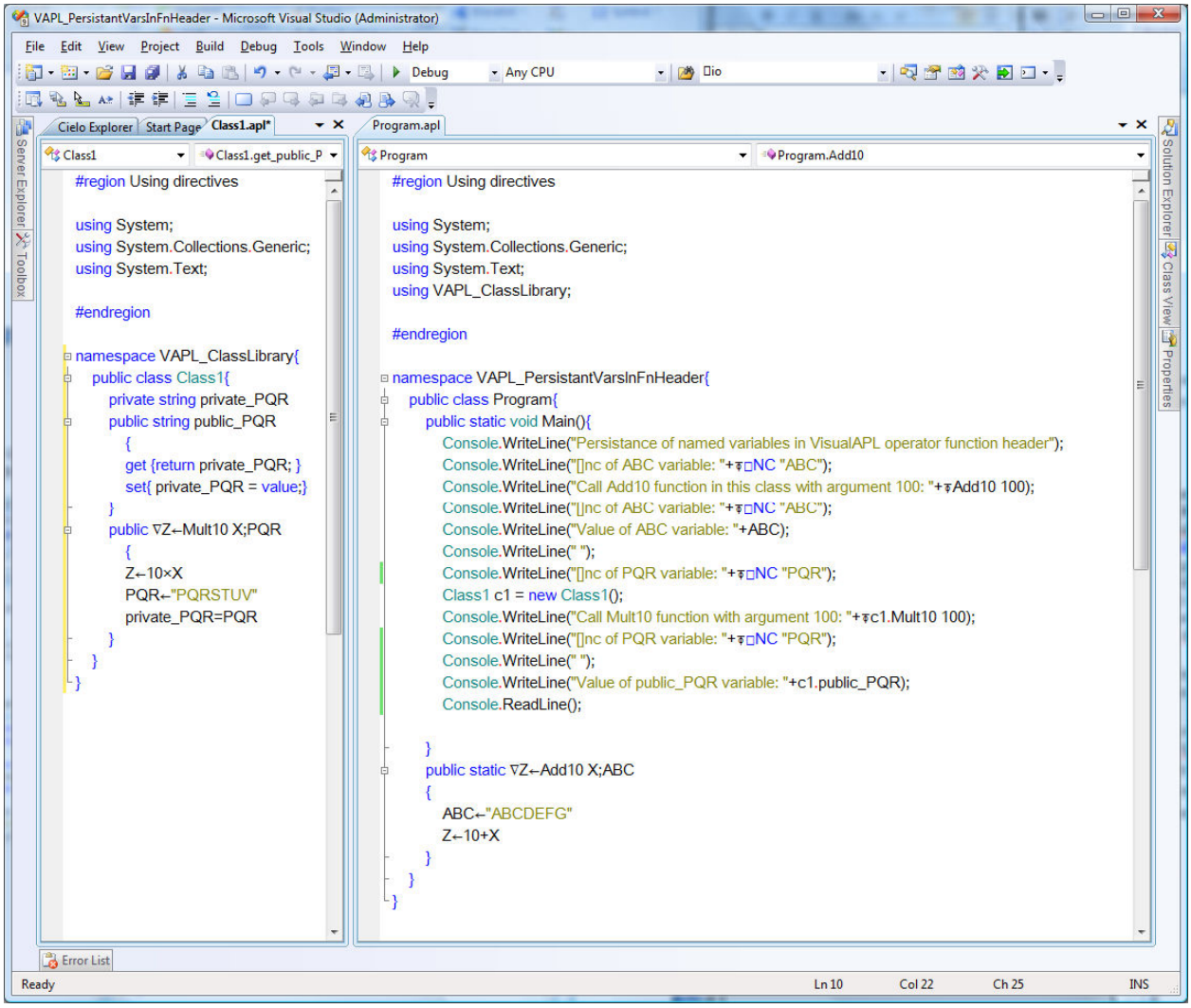


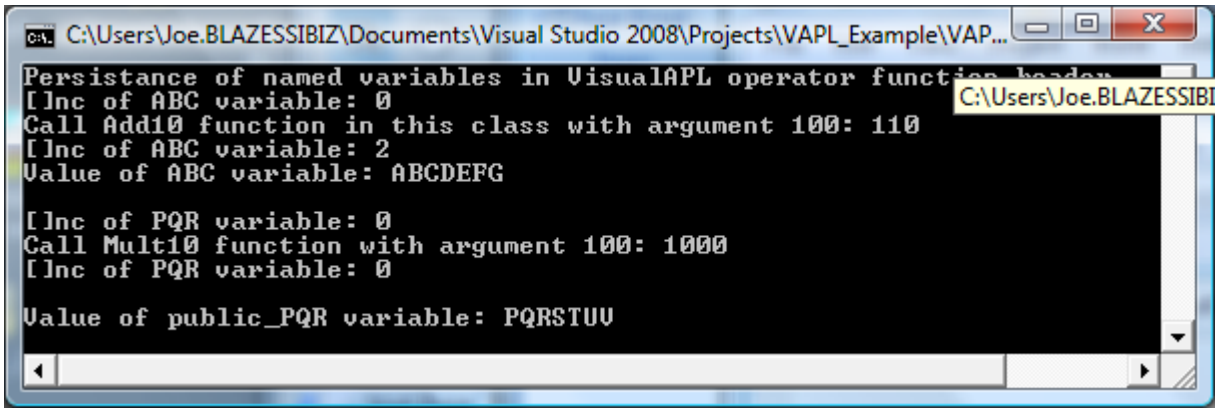
When the console project is run and the Add10 function is called by the Main() function, the named variable in the header of the Add10 function, ABC, persists after the Add10 function call because the Add10 function is in the same class as the Main() function, as verified by the VisualAPL □nc system function result when applied to the variable name “ABC”.

When the Mult10 function is called by the Main() function, the named variable in the header of the Mult10 function, PQR, does not exist in the class containing the Main() function because the Mult10 function is not in the same class as the Main() function.



By adding a public property, public_PQR, to the VAPL_ClassLibrary.Class1 and appropriately setting the underlying Class1.private_PQR variable in the definition of the Mult10 function, the calling class can access the public property value when the Main() function is called:




C:\Users\Joe.BLAZESSIBIZ\Documents\Visual Studio 2008\Projects\VAPL_Example\VAP...
Persistance of named variables in VisualAPL operator function header
[]nc of ABC variable: 0
Call Add10 function in this class with argument 100: 110
[]nc of ABC variable: 2
Value of ABC variable: ABCDEFG
[]nc of PQR variable: 0
Call Mult10 function with argument 100: 1000
[]nc of PQR variable: 0
Value of public_PQR variable: PQRSTUV
C:\Users\Joe.BLAZESSIBI