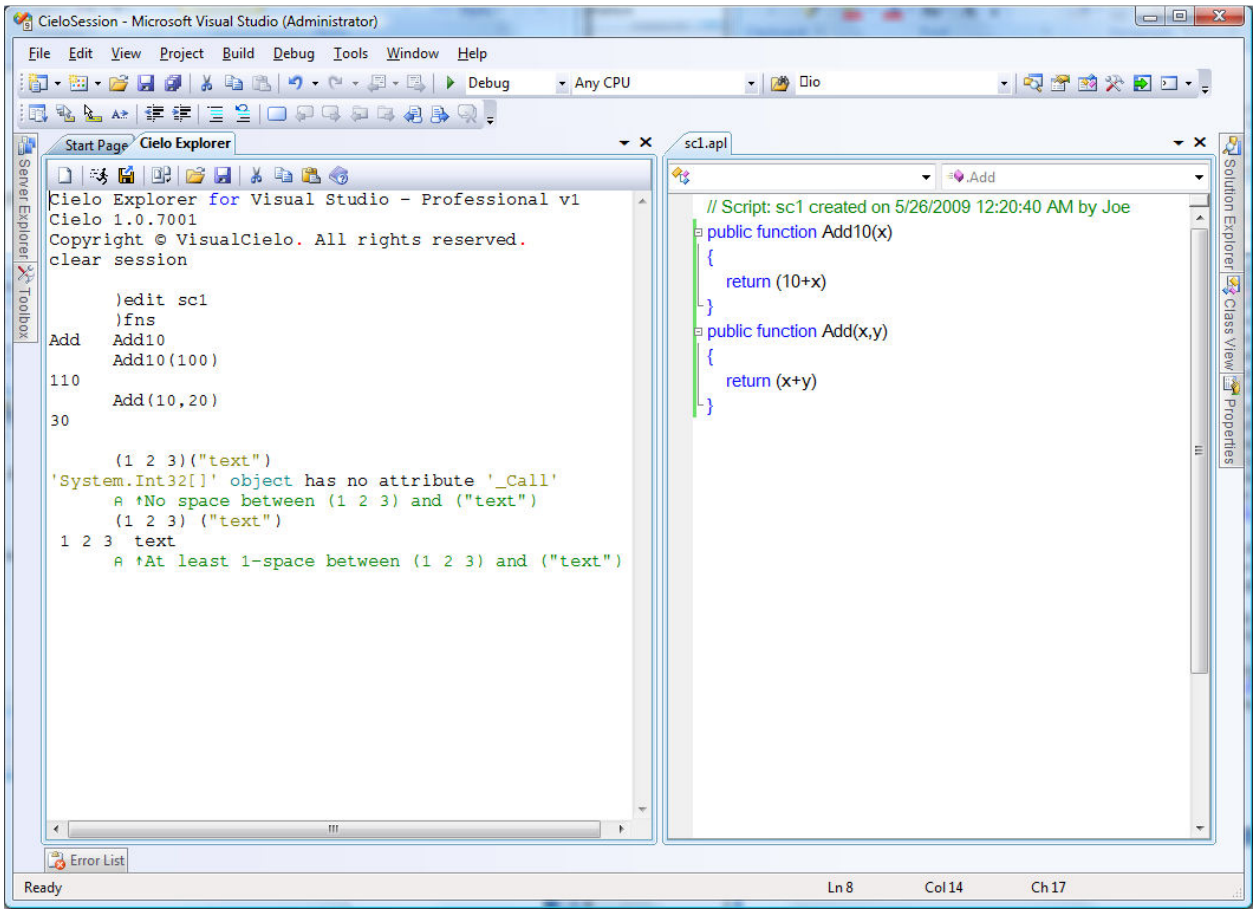
Calling Functions with the “method” [.Net Standard] Function Signature

When a VisualAPL function uses the **method** function signature syntax, the function header encloses the function arguments in parentheses “(…)” which immediately follow the function name and separates the arguments with commas “arg1, arg2”. The **method** function signature is one of the most significant elements of the inter-operability of .Net languages, including C#, VB.Net and VisualAPL.

The Microsoft JIT (just-in-time) compiler is used to compile the programming language source code to CLR (common language runtime) code. The JIT complier interprets the **method** function signature syntax to locate where function calls are occurring in the source code.

VisualAPL does not employ a proprietary interpreter component like legacy APL language implementations, but instead uses the Microsoft JIT compiler to compile VisualAPL source code to CLR code. Therefore whenever the **method** function signature syntax occurs in VisualAPL source code, the JIT compiler will interpret that VisualAPL code segment as a function call.

In the following Cielo Explorer session, the “sc1” script contains the definition of the “Add10” and “Add” functions. Both of these functions use the **method** function signature syntax. Thus in the Cielo Explorer session they are called by immediately following the function name with the parentheses-enclosed function argument list.



The remainder of the Cielo Explorer session illustrates that because the Microsoft JIT compiler is used to compile VisualAPL code to CLR code, careful spacing is necessary when typing the elements of a VisualAPL nested array which involves “)(“. At least one space must be included between elements of a VisualAPL nested array, when those elements are (possibly superfluously) enclosed in parentheses.

If this spacing requirement is not met by the VisualAPL programmer’s typing, the Microsoft JIT compiler will interpret the VisualAPL code statement as a function call, resulting in the illustrated error message “… object has no attribute ‘_Call’”. Thus “(…)(…”, without the space, is interpreted by the Microsoft JIT compiler as a function call with function name “(…)” and “(…)” may not actually be a function, whereas ”(…) (…”, with the space, is a VisualAPL nested array representation.

The nature of the Microsoft JIT compiler is such that only at run-time can the ‘functionality’ of “(…)” be determined. It is tempting to think that overriding the Microsoft JIT compiler would be an alternative to this evidently non-traditional APL behavior. However this would mean the retrograde step of incorporating a proprietary interpreter into the VisualAPL product with the attendant maintenance effort, cost to the developers and licensees, deployment issues and inevitable delays to update such a proprietary VisualAPL interpreter when Microsoft modified .Net in any significant way.

So when typing VisualAPL statements incorporating parentheses, remember to add the space between “) (“ when appropriate to avoid having the Microsoft JIT compiler improperly interpret “)(“ as a function call.