Some special remarks for macros that should work with the X# macro compiler.  
Normal expressions should ‘just work’, including function calls, constructor calls, normal operators etc.   
The following contains some problems that I have seen with VO code or some remarks about special handing of macros.

|  |  |  |
| --- | --- | --- |
|  | Macro | Remarks |
| 1 | local cMacro as string  local oBlock as \_codeblock  cMacro := "{|a| .not.a}"  oBlock := &(cMacro)  ? Eval(oBlock,true) *// false*  ? Eval(oBlock,false) *// true* | VO allows .not., .or., .xor. without whitespace between Can we do that ? |
| 2 | local cMacro as STRING  DBUSEAREA(true,,"Customer")  cMacro := "LastName + FirstName"  ? &cMacro  wait | This appears a lot as index expressions etc. This needs to be translated to \_\_VarGet (“LASTNAME”) + \_\_VarGet (“FIRSTNAME”). This is NEW. Vulcan does not allow this. But LASTNAME and/or FIRSTNAME can also be memory variables. \_\_VarGet will look for a field in the current workarea first and when that does not exist then it will look for a memvar with that name. When both do not exist then a runtime error will be generated by \_\_VarGet() The “plumbing” for the memvars is not ready but we need to support this. |
| 3 | local cMacro as STRING  DBUSEAREA(true,,"Customer")  cMacro := "\_FIELD->LastName + \_FIELD->FirstName"  ? &cMacro  wait | This needs to be translated to \_\_FieldGet(“LASTNAME”) + \_\_FieldGet(“FIRSTNAME”) |
| 4 | local cMacro as STRING  DBUSEAREA(true,," Customer")  cMacro := "CUSTOMER->LastName + CUSTOMER->FirstName"  ? &cMacro  wait | Needs to be translated to  \_\_FieldGetWa("CUSTOMER ", " LASTNAME ")+ \_\_FieldGetWa("CUSTOMER ", " FIRSTNAME ") |
| 5 | local macro as string  memvar number  number := 10  macro := "number += 10"  ? &macro *// 20*  ? number *// 20*  wait | Needs to be translated to \_\_VarPut(“NUMBER”, \_\_VarGet(“NUMBER”)+10. See remarks for #2 |
| 6 | \_FIELD->FIRSTNAME := “Nikos” | Needs to be translated to \_\_FieldPut(“FIRSTNAME”,”Nikos”) |
| 7 | CUSTOMER-> FIRSTNAME:= “Robert | \_\_FieldPutWa(“CUSTOMER”, “FIRSTNAME”,”Robert”) |
| 8 | For resolving function calls it should respect the ClassLibraryAttribute in the assembly headers like Vulcan does | This attribute declares the static class where the functions are defined. |
| 9 | The VO macro compiler ‘allows’ missing closing parens, such as in  local cMacro as string  cMacro := "Left('abc',1"  ? &cMacro  VO Allso allows missing end curly for literal arrays. | Can we and do we want to support this? When not then we need to generate a clear error message because people will have used that (without knowing?) |
| 10 | The VO macro compiler also allows to create memvars ‘on the fly’ in the macro. Vulcan does not support that. local cMacro as string  local oBlock as \_codeblock  cMacro := "{|a,b| c := a +b, b := b + c, b}"  oBlock := &(cMacro)  ? Eval(oBlock,10,11) *// 32* | Inside the codeblock \_\_VarPut(“c”, a+b) is called.  This will allocate a memvar with the lifetime of the  codeblock. This does not work in Vulcan. For Gruen I have created a macro preprocessor that detects this and automatically adds a parameter **c** to the list of parameters, so there is a place on the stack for the value. |
| 11 | VO allows empty TRUE or FALSE blocks for IIF() local cMacro as string  local oBlock as \_codeblock  cMacro := "{|a,b| iif(a, b,)}"  oBlock := &(cMacro)  ? Eval(oBlock,true,1) *// 1*  ? Eval(oBlock,false,1) *// NIL* | Should not be too difficult to implement I think. When not then we need to generate a clear error message because people will have used that (without knowing?) |