‘Mates8’ usage in VB.Net

In version v8.3.0 there have been several changes.

New ‘Config’ class allows private input and output configuration: for rounding; imaginary symbol, decimal or other numeric base output; fractions; input case sensitivity; etcetera. Now there is no shared assignments to ‘MathGlobal8’ members, but to an instance of ‘Config’ class.

Method ‘matrixParser.parse’ is no more a shared method so, an instance is needed to invoke the method.

Output methods like ‘toStringExpr(cfg as Config)’ or ‘toStringComplex(cfg as Config)’ employ the Config instance parameter to know how to format the output data.

Please find an explanation in the following code’s instructions and comments:

```
1. Imports mates8
2.
3. Module Module1
4.
5.     Sub Main()
6.         testMates8()
7.     End Sub
8.     Sub testMates8()
9.         Try
10.             Dim N As Int32 = 1
11.             Dim mP As New matrixParser
12.             Dim sResult As String = ""
13.             Dim strVarsAndFns As String = ""
14.             Dim cfg As New Config
15.
16.             ' Set configuration for input data:
17.             cfg.bEngNotation = True ' exponents multiples of 3 (10.3e3, 1.2e6, ...)
18.             cfg.bIgnoreSpaces = True
```

```
19.             cfg.bCaseSensitive = True
20.             cfg.degreesType = MathGlobal8.degreesType.radians
21.
22.             ' the following are examined only when
23.             ' 'ToString" type methods are invoked:
24.             cfg.bFractions = True
25.             cfg.bRounding = True ' round to 3 decimals
26.             cfg.bDetail = False ' no detailed info
27.             cfg.Base = MathGlobal8.outputBase.decimal
28.
29.             ' set config. into mP, mostly for
30.             ' input data as in toStringXXXX() methods
31.             ' cfg is needed to be passed as parameter (except for matrixParser
32.             ' mP.toString()). These are toStringExpr(cfg), toStringMatrixParser(cfg),
33.             ' toStringMtx(cfg), toStringPoly(cfg), toStringComplex(cfg),
34.             ' for expressions, matrices, polynomials and complex numbers,
35.             ' respectively.
36.             mP.setConfig(cfg)
37.
38.             '     INPUT for matrixParser.Parse(strQuery,strVarsAndFns,oVars,cfg)
39.             '     =============================
40.             '  (Only the first string parameter strQuery is mandatory, the other
41.             '  three may be omitted).
42.             '  1) strQuery = the math expression to parse,
43.             '     for example: strQuery="2*2", "2*x+3*x", "∫(cos(x))dx", "roots(x^16-1)"
44.             '                  or a matrix expression with columns delimited by
45.             '                  semicolons and rows by vbCrLf as "A^-1"
46.             '  2) strVarsAndFns = "" or eventualy variables values or functions
47.             '                     for ex. "x=-1" or  "A=2;3" + vbCrLf + "-1;2"
48.             '                     Dim oVars As VarsAndFns = Nothing
49.             '  3) oVars is a VarsAndFns object. Its finality is to store all the
50.             '    variables or custom functions contained in strQuery and/or strVarsAndFns.
51.             '    Each variable will have only one entry in oVars. For example, if
52.             '    strQuery is "z+x^2+3*x+y" then oVars.getNamesList will return a string
53.             '    array = {"z","x","y"}. Below you may find an example where oVars.setValue is
54.             '    used to evaluate an expression ("2x^2+5*y") with 2 variables.
55.             '   4) An instance, 'cfg', of Config class in order to determine how the input
56.             '    data will be parsed.
57.
58.             '     OUTPUT:
59.             '     =======
60.             '     1) mP.toString returns the result as a string.
```

```
61.             '     2) mP.retCjo() returns a complex or, eventually, an array of complex.
62.             '     3) When the result is a matrix
63.             '       xmP.ret.exprMtx.getExpr(row, column) returns the expression
64.             '       contained at a row and column ((0,0) is the first row and columns)
65.
66.             ' mP.ret.exprMtx.getExpr(row, column).IsReal will tell
67.             ' if the element's content is a real number and
68.             ' mP.ret.exprMtx.getExpr(row, column).toDouble its value.
69.             ' mP.ret.exprMtx.rows gives the number of rows in the matrix
70.             ' mP.ret.exprMtx.cols gives the # of columns
71.
72.             ' As an exxmple, if we want the roots of x^16-1
73.             ' we equal strQuery="roots(x^16-1)", execute
74.             ' mP = matrixParser.parse(strQuery,"", nothing)
75.             ' and, at the output, the roots will be in mP.retCjo(); first, the real
76.             ' roots (if any) and then the complex (if any):
77.             '
78.             ' root1:  mp.retCjo(0)  ' = -1 (real)
79.             ' root2:  mp.retCjo(1)  ' =  1 (real)
80.             ' root2:  mp.retCjo(2)  ' = -i (complex)
81.             ' ...
82.             ' root16: mp.retCjo(15) ' = (0.923879532511287 -i*0.38268343236509) (complex)
83.
84.             ' Real roots, in mP.retCjo(), are ordered from most negative to most positive.
85.             ' If a root is real and not complex, i.e. the imaginary value is zero,
86.             ' mP.retCjo(0).IsReal will be True.
87.
88.
89.             Dim strQuery As String = "2*2/3"
90.             Dim oVars As VarsAndFns = Nothing
91.             Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery) ' 4/3 (if mathglobal8.fractions=False => =1.333)
92.             mP.parse(strQuery)
93.             If mP.errMsg.Length Then
94.                 Throw New Exception(mP.errMsg)
95.             End If
96.             Console.WriteLine("Result: " + mP.ToString) ' Result: 4
97.             Console.WriteLine("")
98.             N += 1
99.
100.             strQuery = "2*x+3*x"
101.             Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery) ' 2*x+3*x
102.             mP.parse(strQuery)
```

```
103.            If mP.errMsg.Length Then
104.                Throw New Exception(mP.errMsg)
105.            End If
106.            Console.WriteLine("Result: " + mP.ToString) ' Result: 5*x
107.            Console.WriteLine("")
108.            N += 1
109.
110.            ' Prefix for hexa numbers &h, for octal &o, binary prefixed by &b
111.            ' 255 as hexadecimal (&hFF),  logical optor. AND, 15 as hexa.(&hF)
112.            ' logical operators valid are "and", "or", "xor", "not", "nand", "nor"
113.            strQuery = "(&hff xor &hf)+3" ' ...and add 3 (decimal)
114.            Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery)
115.            mP.parse(strQuery)
116.            If mP.errMsg.Length Then
117.                Throw New Exception(mP.errMsg)
118.            End If
119.            Console.WriteLine("Result: " + mP.ToString) ' Result: 18
120.            Console.WriteLine("")
121.            N += 1
122.
123.            strQuery = "∫(cos(x))dx"
124.            Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery) ' integral(cos(x))dx
125.            mP.parse(strQuery)
126.            If mP.errMsg.Length Then
127.                Throw New Exception(mP.errMsg)
128.            End If
129.            Console.WriteLine("Result: " + mP.ToString) ' Result: sin(x) + _constant
130.            Console.WriteLine("")
131.            N += 1
132.
133.            strQuery = "f(3)-f(2)"
134.            strVarsAndFns = "f(x)=x^2-1"
135.            Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery + _
136.                    " where " + strVarsAndFns)
137.            mP.parse(strQuery, strVarsAndFns)
138.            If mP.errMsg.Length Then
139.                Throw New Exception(mP.errMsg)
140.            End If
141.            Console.WriteLine("Result: " + mP.ToString) ' Result: 3^2-1-(2^2-1)=8-(3)=5
142.            Console.WriteLine("")
143.            N += 1
144.
```

```
145.            cfg.bIgnoreSpaces = False ' carrige return (vbCrLf) must be considered
146.            strQuery = "A"
147.            ' semicolons delimit columns, vbCrLf delimits rows:
148.            strVarsAndFns = "A=2;3" + vbCrLf + "-1;-2" ' watch for semicolons and vbCrLf
149.            Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery) ' integral(cos(x))dx
150.            mP.parse(strQuery, strVarsAndFns)
151.            If mP.errMsg.Length Then
152.                Throw New Exception(mP.errMsg)
153.            End If
154.            Console.WriteLine("Result: " + vbCrLf + mP.ToString) ' Result: matrix A=2;3|-1;-2
155.            Console.WriteLine("element at row 2, column 2 is: {0}", _
156.                              mP.ret.exprMtx.getExpr(1, 1).toDouble)
157.            Console.WriteLine("")
158.            N += 1
159.
160.            strQuery = "A*A^-1"
161.            strVarsAndFns = "A=2;3|-1;-2" ' "|" is also valid for row delimiter
162.            Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery + " where " + strVarsAndFns)
163.            mP.parse(strQuery, strVarsAndFns)
164.            If mP.errMsg.Length Then
165.                Throw New Exception(mP.errMsg)
166.            End If
167.            Console.WriteLine("Result: " + vbCrLf + mP.ToString) ' Result: identity matrix
168.            Console.WriteLine("element at row 2, column 2 is:  {0}", _
169.                              mP.ret.exprMtx.getExpr(1, 1).ToStringExpr(cfg))
170.            Console.WriteLine("")
171.            N += 1
172.
173.            strQuery = "A^-1"
174.            strVarsAndFns = "A=z;x|-2;3" + vbCrLf + "x=y+4"
175.            Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery + " where " + strVarsAndFns)
176.            mP.parse(strQuery, strVarsAndFns)
177.            If mP.errMsg.Length Then
178.                Throw New Exception(mP.errMsg)
179.            End If
180.            Console.WriteLine("Result: " + vbCrLf + mP.ToString) ' Result: identity matrix
181.            Console.WriteLine("element at row 2, column 2 is:  {0}", _
182.                              mP.ret.exprMtx.getExpr(1, 1).ToStringExpr(cfg))
183.            Console.WriteLine("")
184.            N += 1
185.
186.
```

```
187.            strQuery = "A*A^-1"
188.            strVarsAndFns = "A=z;x|-2;3" + vbCrLf + "x=y+4"
189.            Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery + " where " + strVarsAndFns)
190.            mP.parse(strQuery, strVarsAndFns)
191.            If mP.errMsg.Length Then
192.                Throw New Exception(mP.errMsg)
193.            End If
194.            Console.WriteLine("Result: " + vbCrLf + mP.ToString) ' Result: identity matrix
195.            Console.WriteLine("element at row 2, column 2 is:  {0}", _
196.                              mP.ret.exprMtx.getExpr(1, 1).ToStringExpr(cfg))
197.            Console.WriteLine("")
198.            N += 1
199.
200.            strQuery = "A^-1*A"
201.            strVarsAndFns = "A=z;x|-2;3" + vbCrLf + "x=y+4"
202.            Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery + " where " + strVarsAndFns)
203.            mP.parse(strQuery, strVarsAndFns)
204.            If mP.errMsg.Length Then
205.                Throw New Exception(mP.errMsg)
206.            End If
207.            Console.WriteLine("Result: " + vbCrLf + mP.ToString) ' Result: identity matrix
208.            Console.WriteLine("element at row 2, column 2 is:  {0}", _
209.                              mP.ret.exprMtx.getExpr(1, 1).ToStringExpr(cfg))
210.            Console.WriteLine("")
211.            N += 1
212.
213.
214.            strQuery = "roots(x^16-1)"
215.            Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery) ' roots(x^16-1)
216.            mP.parse(strQuery)
217.            If mP.errMsg.Length Then
218.                Throw New Exception(mP.errMsg)
219.            End If
220.            Console.WriteLine("Result: " + vbCrLf + mP.ToString) ' Result: -1, 1, i, -i, ....
221.            Console.WriteLine("The 3rd. root is " + mP.retCjo(2).toStringComplex(cfg))
222.            Console.WriteLine("")
223.            N += 1
224.
225.            ' Example 2:
226.            ' Now we want to evaluate "2x^2+5" for x=-1, x=2, x=3 and x=-i
227.            strQuery = "2x^2+5"
228.            Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery)
```

```
229.            ' 1) Call parse() method:
230.            mP.parse(strQuery)
231.            If mP.errMsg.Length Then
232.                Throw New Exception(mP.errMsg)
233.            End If
234.
235.            ' 2) An AST tree has been created; call the evalExpression() method
236.            '    for each value of x:
237.            Console.Write("  Evaluating " + strQuery + ", for x=-1")
238.            Dim cmplx As Complex = mP.ret.curExpr.evalExpression(New Complex(-1))
239.            Console.WriteLine(": " + cmplx.toStringComplex(cfg)) ' = 7
240.
241.            Console.Write("  Evaluating " + strQuery + ", for x=2")
242.            cmplx = mP.ret.curExpr.evalExpression(New Complex(2))
243.            Console.WriteLine(": " + cmplx.toStringComplex(cfg)) ' = 13
244.
245.            Console.Write("  Evaluating " + strQuery + ", for x=3")
246.            cmplx = mP.ret.curExpr.evalExpression(New Complex(3))
247.            Console.WriteLine(": " + cmplx.toStringComplex(cfg)) ' = 23
248.
249.            Console.Write("  Evaluating " + strQuery + ", for x=-i")
250.            ' note x= -i => real part= 0, imaginary= -1:
251.            cmplx = mP.ret.curExpr.evalExpression(New Complex(0, -1))
252.            Console.WriteLine(": " + cmplx.toStringComplex(cfg)) ' = 3
253.            N += 1
254.
255.            strQuery = "2x^2+5*y"
256.            Console.WriteLine(N.ToString + ". Parsing and evaluating: " + strQuery)
257.            ' 1) Call parse() method:
258.            mP.parse(strQuery, "", oVars)
259.            If mP.errMsg.Length Then
260.                Throw New Exception(mP.errMsg)
261.            End If
262.
263.            Console.Write("  Evaluating " + strQuery + ", for x=1 and y=2")
264.            ' note x= -i => real part= 0, imaginary= -1:
265.            oVars.setValue(oVars.getVarIDByName("x"), _
266.                           New ExprMatrix(New Complex(1)))
267.            oVars.setValue(oVars.getVarIDByName("y"), _
268.                           New ExprMatrix(New Complex(2)))
269.            cmplx = mP.ret.curExpr.evalExpression(Nothing, oVars)
270.            Console.WriteLine(": " + cmplx.toStringComplex(cfg)) ' 2+10=12
```

```
271.             Console.WriteLine("")
272.             N += 1
273.
274.             Console.WriteLine(" Multiply previous expression " + strQuery + " by 'Pi:'")
275.             Dim product As Expression = _
276.                 (New Expression(Math.PI)) * mP.ret.curExpr
277.             Console.WriteLine("...* Pi = " + product.ToStringExpr(cfg)) '
278.             Console.WriteLine("")
279.             N += 1
280.
281.             Console.WriteLine(" ... and multiply same expression " + strQuery + " by 'i*Pi:'")
282.             product = _
283.                 (New Expression(New Complex(0, Math.PI))) * mP.ret.curExpr
284.             Console.WriteLine("...* i * Pi = " + product.ToStringExpr(cfg)) '
285.             Console.WriteLine("")
286.             N += 1
287.
288.         Catch ex As Exception
289.             Console.WriteLine(ex.ToString)
290.         End Try
291.         Console.WriteLine("Press 'Enter' to exit.")
292.         Console.ReadLine()
293.     End Sub
294.
295. End Module
```

In order to know the list of possible functions to invoke and their syntax, as trigonometric 'sin(x)', hyperbolic 'sinh(x)', or real part Re(z) and so on, please take a look at http://xrjunque.nom.es/precis/polycalc.aspx under "Constants/Functions". Also the box besides titled "Examples" can possibly give you a tip.