Calculator Routines and Utilities

This document describes a number of routines written in VB6 code that support calculator functions. The source code for all the routines are appended at the end of this document. The source should be easy to convert to other computer languages.

The main function “Eval” will perform the calculation process for a single Equation or Command.
The function “EvalMulti” can process a series of Equations or Commands with each Equation or Command separated by a line terminator.

Numbers may be presented in either Normal or Scientific/Exponent form nn.nnEnn or nn.nnE±nn format

The function Eval(ExpOrCom As String) As String
The function EvalMulti(ExpsOrComs As String) As String

Example of use :-

```
Dim Reply$
Dim Exp$
        Exp$=“2 + 3 * 4”
        Reply$=Eval(Exp$)               ‘Reply$ will contain the string “14”
```

Note: After a calculation has been processed successfully the result is also placed in the MSM accumulator.

The reply from the Eval function can be :-
1. The result of the Equation
2. The Status of a Command
3. An Error Message

The Eval function will respond to the following MSM Commands

1.	Cell Assignment	“Cell Name” := “New Cell Contents”
2.	Move Accumulator Contents to a Cell	-> “Cell Name”
3.	Move Cell Contents to the Accumulator	<- “Cell Name”
4.	Erase or Delete a Cell	Erase, “Cell Name”
5.	Compact the Storage Structure	Tidy
6.	Initialise or Reset the MSM Structure	Init

Note: With a Cell assignment if the Cell does not exist then it will be created, however, when a move operation is requested if the cell does not exist then an error message will be raised. Likewise an error message will be raised with an Erase request when the cell address is protected “A0” or does not exist.

The response to an MSM command will be either “OK”
or an error message starting with the characters “Err”

Example of use with MSM memory stores being used :-

```
Dim Reply$
Dim Exp$
        Exp$= “a1:= 5.3”
        Reply$=Eval(Exp$)               ‘Reply$ will contain the string “OK”
        Exp$= “1 + [A1] +17e-1”
        Reply$=Eval(Exp$)               ‘Reply$ will contain the string “8” and [A0}=“8”
        Exp$= “[a0] * [A0]”
        Reply$=Eval(Exp$)               ‘Reply$ will contain the string “64” and [A0}=“64”

        Rem An example of a complex equation
        Exp$= "siN(d2R((9-4)^2+5)+D2r((2*6+3)*8/2))"
        Reply$=Eval(Exp$)               ‘Reply$ will contain the string “1”
```

Calculator Routines and Utilities

Possible Eval/EvalMulti Error Messages

1. ErrDiv0 Division by Zero
2. ErrExp Error with Expression (A comment or blank line found)
3. ErrFun Error Unknown/Bad Function/Parameter/s
4. ErrNum Bad Format of a Number
5. Err2DP Bad Number 2 decimal points
6. Err-([Excess Opening Brackets
7. Err-)] Excess Closing Brackets
8. Err-) Conflict of Open/Closing (] Brackets
9. Err-] Conflict of Open/Closing [) Brackets

10. ErrSQM Stack/Queue/Manager Initialise Operation Failed
11. ErrQue Queue storage problem
12. ErrStk Stack storage problem
13. Err[?] Cell Name or Location Invalid
14. ErrCFun Unknown Cell Storage structure Function
15. ErrCMax Cell Store storage area Full

The equation processor supports the following basic operations.

1. + Addition
2. - Subtraction
3. * Multiplication
4. / Division
5. ^ Raise to a power

The equation processor supports the following advanced math's features
Note case of function names is unimportant as all are converted to uppercase.

1.	Pi()	Gives the value of Pi.
2.	Cos(Angle in Radians)	Gives Cosine of Angle
3.	Sin(Angle in Radians)	Gives Sine of Angle
4.	Tan(Angle in Radians)	Gives Tangent of Angle
5.	ACos(Cosine)	Gives Angle in Radians of Inverse Cosine
6.	ASin(Sine)	Gives Angle in Radians of Inverse Sine
7.	ATan(Tangent)	Gives Angle in Radians of Inverse Tangent
8.	Exp(Power)	Raises the base of natural Logarithms to a specified power
9.	Log(Value)	Returns the logarithmic expression for the given Value
10.	Sqr(Value)	Returns the square root of the given Value
11.	D2R(Degrees)	Convert Degrees to Radians
12.	R2D(Radians)	Convert Radians to Degrees

Examples using math's functions and other operations

```
Dim Reply$
Dim Exp$

    Exp$= "R2D(ACOS(COS(D2R(2 * 30))))"
    Reply$=Eval(Exp$)            'Reply$ will contain the string "60"

    Rem An example of a complex equation
    Exp$= "siN(d2R((9-4)^2+5)+D2r((2*6+3)*8/2))"
    Reply$=Eval(Exp$)            'Reply$ will contain the string "1"

    Rem An examples of a  Unary operations
    Exp$= " (- ( 2 ) ) "
    Reply$=Eval(Exp$)            'Reply$ will contain the string "-2"

    Exp$= "1 +- 2 -- 3 -+4 ++5"
    Reply$=Eval(Exp$)            'Reply$ will contain the string "3"
```

```
Attribute VB_Name = "CalcEngine"
Option Explicit
Rem
Rem Utility routines specifically for Calculation Applications Author R. J. Spriggs
Rem These are a set of  utility routine used to perform Expression Calculations.
Rem Programmers: The Visual Basic source code in this document may be copied
Rem and reused for any non commercial use.
Rem The author  disclaims  all  warranties , expressed  or  implied,
Rem including, without limitation, the warranties of merchantability
Rem and of fitness for any purpose.  The author assumes no liability
Rem for damages, direct or consequential , which may result from the
Rem use of this code described in this document.

Rem 29/Apr/2019 RJS  Initial Module Design and Creation
Rem Modification Record
Rem Developing simple calculator features           29/Apr/2019 RSP
Rem Introduced Cell Storage features                30/Jun/2019 RSP
Rem Total redesign of Eval Application              18/Jly/2019 RSP
Rem Redesign of Eval Application completed          31/Jly/2019 RSP
Rem Removal of redundant code                       02/Aug/2019 RSP
Rem Redesign of Memory Store Manager MSM            02/Aug/2019 RSP
Rem Make some Routines and Constants Global         04/Aug/2019 RSP
Rem Include EvalMulti into this suite               20/Aug/2019 RSP
Rem Upgrades to EvalMulti features                  02/Sep/2019 RSP
Rem TBA                                             29/Apr/2019 RSP

Rem Calc Utility Functions and Subroutines.
Rem ======================================
Rem
Rem        External Access Functions and Subroutines located in this Module.
Rem
Rem     Type        Name            Comment
Rem
Rem     Function    Eval            New     Evaluate Expressions or Process Commands
Rem     Function    EvalMutli       New     Evaluate a set of Expressions or Process Commands
Rem     Function    MSM             New     A Memory cell Store Manager
Rem     Function    Queue           New     A Queue Manager
Rem     Function    Stack           New     A Stack Manager
Rem
Rem
Rem ======================================
Rem
Rem        Internal  Functions and Subroutines located in this Module.
Rem
Rem     Type        Name            Comment
Rem
Rem     Function    GetNum          New     Extract a number from a String or Memory Store
Rem     Function    GetHier         New     Will identify calculation hierarchy
Rem     Function    CalcSum         New     Will perform calculations on a pair of values
Rem     Function    BrackTst        New     Will verify Brackets in an equation match
Rem     Function    Fun2Num         New     Will convert Functions to a number
Rem     Function    SignFix         New     Will resolve some Bracket and Unary sign parings
Rem
Rem ======================================
Rem
Rem        Internal  Functions and Subroutines located in other Modules.
Rem     Source code for these routines is located at the end of this document
Rem
Rem     Type        Name            Comment
Rem
Rem     Function    NewLine$                Will Reply with CR LF character sequence
Rem     Function    V2S$                    Will Convert Value to String (no Spaces)
Rem     Function    SmartUC                 Smart Tidy and LC to UC Character Converter

Rem ============================================
Rem Define Specialist and Configuration Constants.
Rem ============================================

Public Const Pi = 3.14159265358979          'High Accuracy PI
'Public Const Pi = 3.141592654               'Low  Accuracy PI

'Const StkMax = 10                           'Calculator Stack Depth  (Test/Default use Setting)
'Const CellMax = 20                          'Number of Cell Stores   (Test/Default use Setting)
'Const QueMax = 10                           'Calculator Queue Length (Test use Only)
'Const QueMax = 30                           'Calculator Queue Length (Test/Default use Setting)



Rem Define below as application requires (Max Value per setting is about 30000)
Global Const StkMax = 10                        'Calculator Stack Depth
Global Const CellMax = 20                       'Number of Cell Stores required
```

```
Global Const QueMax = 30                        'Calculator Queue Length typically Stkmax*3

Rem ================================================
Rem End of Configuration Constants.
Rem ================================================



Rem ================================================
Rem Control Constants for this application.
Rem ================================================

Global Const QSMDebug = 99                  'Produce Debug Print of Stack, Queue or Manager Stores
Global Const QSMInit = 98                   'Initialise the Stack, Queue or Manager Stores
Global Const QSCnt = 97                     'Indicate number of items in Stack or Queue Stores
Global Const QSPeek = 96                    'View first Remove item on Stack or Queue
Global Const QueAdd = 1                     'Add    a value to   a Queue
Global Const QueRem = 2                     'Remove a value from a Queue
Global Const StkPush = 3                    'Push   a value onto the Stack
Global Const StkPop = 4                     'Pop    a value from the Stack

Global Const MSMCmd = 10                    'Interrogate a Command string
Global Const MSMTdy = 11                    'Tidy Cell Structure
Global Const MSMDel = 12                    'Delete Selected Cell
Global Const MSMCre = 20                    'Create/Select   Cell
Global Const MSMSel = 21                    'Select a        Cell
Global Const MSMLdA = 30                    'Load Acumulator Cell
Global Const MSMRdA = 31                    'Read Acumulator Cell
Global Const MSMLdS = 32                    'Load Selected   Cell
Global Const MSMRdS = 33                    'Read Selected   Cell

Rem ================================================
Rem End of Control Constants for this application.
Rem ================================================


Rem Maths Functions and Subroutines.
Rem ================================
Rem
Rem         Functions and Subroutines located in this Module.
Rem
Rem     Type        Name                    Comment
Rem
Rem     Function    ACos                    Maths   Inverse CoSine Routine  (Ready and Now Used)
Rem     Function    ASin                    Maths   Inverse Sine Routine    (Ready and Now Used)
Rem     Function    D2R                     Maths   Converts Degrees to Radians
Rem     Function    R2D                     Maths   Converts Radians to Degrees


Rem
Rem ================================================
Rem Start of
Rem Infix Expression and Command Processing Routines.
Rem ================================================
Rem

Public Function Eval(ExpTxt As String) As String
Rem This routine will Parse the text string and process the calculation
Rem The revised version of the programme uses the "Shunting Yard Algorithm"
Rem which is converts infix notation to RPN.
Rem (http://en.wikipedia.org/wiki/Shunting-yard_algorithm).
Rem The algorithm was invented by Edsger Dijkstra
Rem (https://en.wikipedia.org/wiki/Edsger_Dijkstra)
Rem and named the "shunting yard" algorithm because its operation
Rem resembles that of a railroad shunting yard.

Rem Modification Record
Rem Initial Design                                  29/Apr/2019 RSP
Rem Total redesign of Eval Application              18/Jly/2019 RSP
Rem Revised for Infix-2-RPn source                  27/Jly/2019 RSP
Rem Referenced Infix-2-RPn algorithm                21/Aug/2019 RSP
Rem Cell address can now be a Function Parameter    21/Aug/2019 RSP
Rem Included extra checks for just a signed number  21/Aug/2019 RSP
Rem Restrict Unary checks to + or - only            06/Sep/2019 RSP
Rem Support for Nested Functions                    09/Sep/2019 RSP
Rem Support for both Scientific and Normal numbers  10/Sep/2019 RSP
Rem TBA                                             05/Jun/2019 RSP

Rem
Rem         Use and Programming Notes
Rem
```

```
Rem When the string is invalid an Error Message starting with the
Rem Letters "Err" will be returned that defines the error.
Rem
Rem     Current Error mesages are :-
Rem
Rem ErrSQM                  Stack/Queue/Manager Initialise Operation Failed
Rem Err[?]                  Cell Name or Location Invalid
Rem ErrDiv0                 Division by Zero
Rem ErrExp                  Error in Expression
Rem ErrFun                  Error Unknown/Bad Function/Parameter
Rem ErrNum                  Bad Format of a Number
Rem Err2DP                  Bad Number 2 decimal points
Rem ErrCMax                 Cell Store storage Full
Rem ErrQue                  Queue storage problem
Rem ErrStk                  Stack storage problem
Rem Err-([                  Excess Opening Brackets
Rem Err-)]                  Excess Closing Brackets
Rem Err-)                   Conflict of Open/Closing (] Brackets
Rem Err-]                   Conflict of Open/Closing [) Brackets
Rem

Rem     Define Data structures used by this Routine

Dim UFlg As Boolean                                     'Unary operation flag Marker
Dim Ip As String                                        'Input Data string Temp Store
Dim Reply As String                                     'Reply Data string Temp Store
Dim ATkn As String                                      'Acumulator or Previous Token
Dim FTkn As String                                      'Function Token
Dim Tkn As String                                       'Token Value Data string Temp Store
Dim Tmp As String                                       'Temporary Storage String
Dim Txt As String                                       'Expression String


    If Len(ExpTxt) = 0 Or InStr(ExpTxt, "!") = 1 Then   'Check for Void Expression
        Eval = "ErrExp"                                 'Not an Expression
        Exit Function                                   'Nothing to do
    End If

    Rem Check expresion for Assignments, Command and  Functions
    If MSM(MSMCmd, ExpTxt) = False Or ExpTxt = "OK" Then
        Eval = ExpTxt                                   'Hold Error or Status message
        Exit Function                                   'Cell Processing is complete
    End If

    If Stack(QSMInit, "") = False Or Queue(QSMInit, "") = False Then
        Eval = "ErrSQM"                                 'Indicate Initialise Stack or Queue Failed
        Exit Function
    End If

    Txt = ExpTxt                                        'Hold entry Expression
    Eval = BrackTst(Txt)                                'Check expresion Brackets match and Tidy
    If Len(Eval) <> 0 Then Exit Function                'Bracket or paring Error

    Eval = ModFunID(Txt)                                'Modify expresion with specialist Functions
    If Len(Eval) <> 0 Then Exit Function                'Function conversion  Error

    Ip = SignFix(Txt)                                   'Hold expression after sorting multi signs


    Rem ====================================================
    Rem Convert Infix Notation string to a RP Notation String.
    Rem ====================================================
    Do While Len(Ip) <> 0                               'Process whole of Infix string
        If GetToken(Ip, Tkn) = False Then               'Get a Number,Function or Error Message
            Eval = Tkn                                  'Hold Specified Error Message
            Exit Function                               'All work done
        End If

        If IsNumeric(Tkn) Then                          'When Token is a Number
            If Queue(QueAdd, Tkn) = False Then          'Put Token in the Queue
                Eval = "ErrQue"                         'Error message
                Exit Function                           'Problem with Queue
            End If

        ElseIf Tkn = "(" Then                           'Check Open Bracket
            If Stack(StkPush, Tkn) = False Then         'Put Token on the Stack
                Eval = "ErrStk"                         'Error message
                Exit Function                           'Problem with Stack
            End If

        ElseIf Tkn = ")" Then                           'Check Close Bracket
```

```
            Do                                          'Remove Tokens From the Stack till ( found
                If Stack(StkPop, Tkn) = False Then      'Remove a Token From the Stack
                    Eval = "ErrStk"                     'Error message
                    Exit Function                       'Problem with Stack
                End If
                If Tkn = "(" Then                       'Transfer Complete
                    Exit Do
                Else
                    If Queue(QueAdd, Tkn) = False Then  'Put Token in the Queue
                        Eval = "ErrQue"                 'Error message
                        Exit Function                   'Problem with Queue
                    End If
                End If
 'DoEvents  'Debug Break
            Loop

        ElseIf Mid$(Tkn, 1, 1) = "~" Then               'Check Others (Function Group)
            If Stack(StkPush, Tkn) = False Then         'Put Token on the Stack
                Eval = "ErrStk"                         'Error message
                Exit Function                           'Problem with Stack
            End If

        ElseIf InStr("+-*/^", Tkn) <> 0 Then            'Check for Operator
            Do
                Call Stack(QSCnt, Tmp)                  'Get count of items in Stack
                If Val(Tmp) = 0 Then Exit Do
                Call Stack(QSPeek, ATkn)                'Hold Last/Previous Token
                If GetHier(Tkn, ATkn) <= 0 Then
                    If Stack(StkPop, ATkn) = False Then 'Remove old Token From the Stack
                        Eval = "ErrStk"                 'Error message
                        Exit Function                   'Problem with Stack
                    End If
                    If Queue(QueAdd, ATkn) = False Then 'Put Token in the Queue
                        Eval = "ErrQue"                 'Error message
                        Exit Function                   'Problem with Queue
                    End If
                Else
                    Exit Do
                End If
 'DoEvents  'Debug Break
            Loop
            If Stack(StkPush, Tkn) = False Then         'Put Token on the Stack
                Eval = "ErrStk"                         'Error message
                Exit Function                           'Problem with Stack
            End If
        End If
 'DoEvents  'Debug Break
    Loop

    Do                                                  'Finally transfer Stack contents to Queue
        Call Stack(QSCnt, Tmp)                          'Get count of items left on the Stack
        If Val(Tmp) = 0 Then Exit Do
        If Stack(StkPop, Tkn) = False Then              'Remove a Token From the Stack
            Eval = "ErrStk"                             'Error message
            Exit Function                               'Problem with Stack
        End If
        If Queue(QueAdd, Tkn) = False Then              'Put Token in the Queue
            Eval = "ErrQue"                             'Error message
            Exit Function                               'Problem with Queue
        End If
'DoEvents    'Debug Break
    Loop



    Rem ===========================================
    Rem Convert a RP Notation String to an Expression.
    Rem ===========================================

    Do                                                  'Loop Till Queue is empty
        Call Queue(QSCnt, Tmp)                          'Get count of items in Queue
        If Val(Tmp) = 0 Then Exit Do                    'Exit when Extractions Complete
        Call Queue(QueRem, Tkn)                         'Remove Token from Queue
        If IsNumeric(Tkn) Then                          'Check action to perform
            Rem Case = Numeric
            If Stack(StkPush, Tkn) = False Then         'Put Token on the Stack
                Eval = "ErrStk"                         'Error message
                Exit Function                           'Problem with Stack
            End If

        ElseIf Mid$(Tkn, 1, 1) = "~" Then
```

```
            Rem Case = Function with or without Parameters
            Tkn = Tkn   'debug stop point
            If FunCalc(Tkn, Reply) = False Then
                Eval = Reply                            'Error message
                Exit Function                           'Problem with Function
            Else
                If Stack(StkPush, Reply) = False Then   'Put Reply on the Stack
                    Eval = "ErrStk"                     'Error message
                    Exit Function                       'Problem with Stack
                End If
            End If

        Else
            Rem Case = Operator
            Call Stack(QSCnt, Tmp)                      'Get count of items on Stack
            If Val(Tmp) = 1 Then                        'This is a Unary Start Operator
                If InStr("+-", Tkn) = 0 Then            'Check Unary only + and - allowed
                    Eval = "ErrExp"                     'Error message
                    Exit Function                       'Problem with Unary
                End If
                If Stack(StkPop, Tmp) = False Then      'Remove only value From the Stack
                    Eval = "ErrStk"                     'Error message
                    Exit Function                       'Problem with Stack
                End If
                If Stack(StkPush, "0") = False Then     'Put dummy Value on the Stack
                    Eval = "ErrStk"                     'Error message
                    Exit Function                       'Problem with Stack
                End If
                If Stack(StkPush, Tmp) = False Then     'Put removed value on the Stack
                    Eval = "ErrStk"                     'Error message
                    Exit Function                       'Problem with Stack
                End If
            End If
            Tmp = Tkn                                   'Remember Function to Process
            If Stack(StkPop, Tkn) = False Then          'Remove a Value Token From the Stack
                Eval = "ErrStk"                         'Error message
                Exit Function                           'Problem with Stack
            End If
            If Stack(StkPop, ATkn) = False Then         'Remove Acumulator From the Stack
                Eval = "ErrStk"                         'Error message
                Exit Function                           'Problem with Stack
            End If
            Eval = CalcSum(ATkn, Tkn, Tmp)              'Perform the Calculation
            If Len(Eval) <> 0 Then Exit Function        'Quit if any Errors
            If Stack(StkPush, ATkn) = False Then        'Place Answer on the Stack
                Eval = "ErrStk"                         'Error message
                Exit Function                           'Problem with Stack
            End If
        End If
    Loop

    If Stack(StkPop, Eval) = False Then                 'Remove calculation Value From the Stack
         Eval = "ErrExp"                                'Error message
         Exit Function                                  'Problem with Stack/Equation/Expression
     End If
     Call Stack(QSCnt, Tmp)                             'Get count of items left on Stack
     If Val(Tmp) <> 0 Then                              'Check that the Stack is empty
         Eval = "ErrExp"                                'Error message
         Exit Function                                  'Problem with Expression
     End If

     Eval = V2S$(Val(Eval))                             'Hold Tidy String
     Call MSM(MSMLdA, Eval)                             'Place result in MSM acumulator store

End Function


Public Function EvalMulti(EqCom As String) As String
Rem This routine will Parse a set of text strings and process the calculations
Rem Modification Record
Rem Initial Design                                  17/Aug/2019 RSP
Rem Include EvalMulti into this suite               20/Aug/2019 RSP
Rem Allow Comments and Blank Lines to be ignored    04/Sep/2019 RSP
Rem TBA                                             05/Jun/2019 RSP

Dim Busy As Boolean                                 'General Purpose Marker
Dim Cnt As Integer                                  'General Purpose Counter
Dim Pnt As Integer                                  'General Purpose Pointer
Dim Mess As String                                  'General Purpose Message Store
Dim Eqn As String                                   'General Purpose Equation String
Dim Txt As String                                   'General Purpose Work String
```

```
Dim W$                                              'General Purpose Work String

    Mess = EqCom                                    'Hold Equations/Comments
    If InStr(EqCom, Chr$(13)) = 0 Then              'Check for single statement mode
        Mess = Mess + Chr$(13)                      'Append a terminator
    End If
                                                    'Build a queue of equations
    Busy = True                                     'Assume there is some work to do
    Eqn = ""                                        'Clear Equation store
    Txt = ""                                        'Start with an empty equation set

    For Cnt = 1 To Len(Mess$)                       'Remove all Blank lines and Comments
        W$ = Mid$(Mess, Cnt, 1)                     'Extract one Character at a time
        If W$ = Chr$(10) Then W$ = ""               'Erase any line feed characters
        If W$ = Chr$(13) Then                       'When Line terminator located
            If Len(Eqn) <> 0 Then                   'Do not process a blank lines
                Txt = Txt + Eqn + W$                'Build up Equation lines
            End If
            Eqn = "":  W$ = "": Busy = True         'Prepare for next equation line
        End If

        If Busy And W$ = "!" And Len(Eqn) = 0 Then  'Check for comment start
            Busy = False                            'Indicate this is a comment line
        End If
        If Busy Then Eqn = Eqn + W$                 'Build up equation string
    Next Cnt

    Busy = True                                     'Assume we have line/s to process
    Do While Busy = True                            'Process all Equations/Commands
        Pnt = InStr(Txt, Chr$(13))                  'Look for a Terminator (CR)

        If Pnt > 1 Then
            W$ = Mid$(Txt, 1, Pnt - 1)              'Remove an Equation/Command
             Eqn$ = W$
            If Len(Txt) > Pnt Then
                Txt = Mid$(Txt, Pnt + 1)            'Remove leading Equation/Command
            Else
                Txt = ""                            'Remove last expression
                Busy = False                        'Job Done
            End If

            Mess$ = Eval(W$)                        'Process Equation/Command
            If InStr(Mess$, "Err") = 1 Then
                Mess$ = Mess$ + "  " + Eqn$          'Add Equation that caused the Error
                Exit Do
            End If
        Else
            Mess$ = "ErrExp": Exit Do               'Bad Expression/Command
        End If
 DoEvents  'Debug Break
    Loop                                            'End of Multi-Line Processing
    EvalMulti = Mess$
End Function

Rem
Rem ================================================
Rem End of
Rem Infix Expression and Command Processing Routines.
Rem ================================================
Rem




Rem ============================================
Rem Utility routines used by Eval Routine.
Rem ============================================

Private Function GetToken(Str As String, Token As String) As Boolean
Rem This routine will extract a Number/Cell Contents or a Function from a string
Rem Modification Record
Rem Adapted from Function GetNum then re-written    25/Jly/2019 RSP
Rem Modified to also accept Scientific number       10/Sep/2019 RSP

Dim CellMode As Boolean                             'Currently processing a cell address
Dim CellLoc As String                               'Name Address of cell being processed
Dim Parsing As Boolean                              'Parsing in Progress Indicator
Dim SciNum As Boolean                               'Parsing Scientific style number Indicator
Dim DP As Boolean                                   'Fraction point marker
Dim Pnt As Integer                                  'General Purpose pointer
```

```
Dim Char As String                                  'A temporary Characters store
Dim Num As String                                   'A temporary number Characters store

Const DigCodes = "0123456789."                      'Valid Number Symbols
'Const FunCodes = "+-*/^()"                        'Calculation Function Symbols
Const FunCodes = "+-*/^(),"                         'Calculation Function Symbols and Pmark
Const FunOpC = "["                                  'Open  Cell     [
Const FunClC = "]"                                  'Close Cell     ]

    Token = ""                                      'Initialise Token store
    GetToken = True                                 'Assume extraction correct
    DP = False                                      'Assume No DP Found so far
    CellMode = False                                'Assume reading a Number/Function
    CellLoc = ""                                    'Not a cell
    SciNum = False                                  'Assume Not Parsing a Scientific Number
    Parsing = False                                 'Assume Not Parsing a Cell or Number

    Do While Len(Str) <> 0
        Char = Mid(Str, 1, 1)                       'Hold   first character
        If Len(Str) > 1 Then                        'Remove it fron String
            Str = Mid$(Str, 2)                      'Remove first character
        Else
            Str = ""                                'String now empty
        End If

        If Char = "~" Then                          'Check for Function Start Marker
            Rem After GetFun has been called  all functions take the form ~nn(...)
            Token = Char + Mid$(Str, 1, 2)          'Extract Function ID
            Str = Mid$(Str, 3)                      'Remove Function ID
            Exit Do                                 'All work Done
        End If

        If InStr(FunCodes, Char) <> 0 Then          'Check for Terminator or Function
            If SciNum = False Then                  'Not a termiator in a Sci number
                If Parsing = True Then              'When a terminator Function
                    Str = Char + Str                'Restore removed Character
                Else
                    Token = Char                    'Hold Function
                End If
                Exit Do                             'Processing Complete
            End If
        End If
        Parsing = True                              'Busy Processing Number or Cell Address

        If CellMode = True Then                     'Check if building a cell address
            If Char = FunOpC Then                   'Check for 2nd opening bracket
                Rem case when a second cell opener indicator used
                Token = "Err[?]"                    'Mark as Invalid Cell address located
                Str = ""                            'Erase characters in bad number
                GetToken = False                    'Indicate Bad number
                Exit Function                       'All work complete
            Else
                If Char = FunClC Then               'Check for Cell Address Terminator
                    Token = CellLoc                 'Hold Cell Name to access
                    GetToken = MSM(MSMSel, Token)   'Select Cell or Hold Error Message
                    If GetToken = True Then         'Read Selected Cell Contents when possible
                        GetToken = MSM(MSMRdS, Token)
                    End If
                    Exit Function                   'Job Done Reply (Cell Value/Error Message)
                End If
                CellLoc = CellLoc + Char            'Build up Cell Name Address string
            End If
        Else
            If Char = FunOpC Then                   'Check for Cell address opening bracket
                CellMode = True                     'Now process as a cell address
                CellLoc = ""                        'Initialise the cell name
            Else

                Rem Processing a Number digits/symbols only
                If SciNum = False Then
                    Pnt = InStr(FunCodes, Char)     'Check for Terminator or Function
                    If Pnt <> 0 Then Exit Do        'When a function code Terminator
                End If

                Pnt = InStr(DigCodes, Char)         'Check for permitted Digit Codes
                If Pnt = 0 Then                     'When not a digit/separator
                    If InStr("E+-", Char) = 0 Then
                        Token = "ErrNum"            'Mark as Invalid Number located
                        Str = ""                    'Erase characters in bad number
                        GetToken = False            'Indicate Bad number
                        Exit Function               'Processing Done
```

```
                    Else
                        If SciNum = False Then
                            SciNum = True           'Assume Scientific format
                        Else
                            If InStr("+-", Char) <> 0 Then
                                SciNum = False      'Back to normal numbers
                            Else
                                Token = "ErrNum"    'Mark as Invalid Number located
                                Str = ""            'Erase characters in bad number
                                GetToken = False    'Indicate Bad number
                                Exit Function       'Processing Done
                            End If
                        End If
                    End If
                End If

                If Char = "." Then                  'A fraction point found
                    If DP = True Then               'When 2nd fraction point found
                        Token = "Err2DP"            'Mark as Invalid Number located
                        Str = ""                    'Erase characters in bad number
                        GetToken = False            'Indicate Bad number
                        Exit Function               'Processing Done
                    Else
                        DP = True                   'Indicate a DP found
                    End If
                End If

                Num = Num + Char                    'Build up Number string
            End If
        End If
    Loop

    If Len(Token) = 0 Then Token = Num              'Hold Number when not a Function
End Function


Private Function GetHier(Cur As String, Prev As String) As Integer
Rem This routine will identify calculation hierarchy
Rem Modification Record
Rem Initial Design                                  05/Jun/2019 RSP
Rem Revised for Infix-2-RPn source                  27/Jly/2019 RSP
Rem Revised to support Functions                    06/Sep/2019 RSP
Rem TBA                                             05/Jun/2019 RSP

Dim Cp As Integer                                   'Current  Pointer
Dim Cc As String                                    'Current  First Character
Dim Pp As Integer                                   'Previous Pointer
Dim Pc As String                                    'Previous First Character
Dim Pnt As Integer                                  'General Purpose pointer

    'Programmer Note    The ~ character indicates that a Function Name letters follow
    Cc = Mid$(Cur, 1, 1)                            'Hold First Character of Current Token
    Pc = Mid$(Prev, 1, 1)                           'Hold First Character of Previous Token
    Cp = Mid$("11223405", InStr("+-*/^~()", Cc), 1) 'Identify Current  Priority
    Pp = Mid$("11223405", InStr("+-*/^~()", Pc), 1) 'Identify Previous Priority
    If Cp = Pp Then GetHier = 0                     'Hiearchy Same
    If Cp < Pp Then GetHier = -1                    'Hiearchy Lower
    If Cp > Pp Then GetHier = 1                     'Hiearchy Greater
End Function

Private Function CalcSum(AcumS As String, NumS As String, Fun As String) As String
Rem This routine will perform all the basic calculations required on the values

Rem Modification Record
Rem Initial Design                                  05/Jun/2019 RSP
Rem Revised for Infix-2-RPn source                  29/Jly/2019 RSP
Rem Source code tidy                                02/Aug/2019 RSP
Rem TBA                                             05/Jun/2019 RSP

Dim Acum As Double                                  'Acumulator Number
Dim Num As Double                                   'Operation Number

    If IsNumeric(AcumS) And IsNumeric(NumS) Then    'Convert String Variables to Double
        Acum = Val(AcumS)
        Num = Val(NumS)
    Else
        CalcSum = "ErrNum"                          'Bad Number
        Exit Function                               'Immediate Exit
    End If

    CalcSum = ""                                    'Assume Good Calculation
```

```
    Select Case Fun

    Case Is = "+"                                   'Operation Addition
        Acum = Acum + Num

    Case Is = "-"                                   'Operation Subtraction
        Acum = Acum - Num

    Case Is = "*"                                   'Operation Multiplication
        Acum = Acum * Num

    Case Is = "/"                                   'Operation Division
        If Num = 0 Then
            CalcSum = "ErrDiv0"                     'Division by zero (Bad)
            Exit Function                           'Immediate Exit
        Else
            Acum = Acum / Num
        End If

    Case Is = "^"                                   'Operation Raise to Power
        Acum = Acum ^ Num

    Case Else
        CalcSum = "ErrFun"                          'Unknown Function
        Exit Function                               'Immediate Exit

     End Select
     AcumS = Str$(Acum)                             'Convert Answer back to a String
End Function


Private Function SignFix(Txt As String) As String
Rem This routine will resolve ++ +- -- -+ pairs
Rem and also include implicit Multiplication via brackets

Rem Modification Record
Rem Revised for Infix-2-RPn source                  29/Jly/2019 RSP
Rem TBA                                             05/Jun/2019 RSP

Dim CPnt As Integer                                 'Character pointer
Dim P$                                              'Work String
Dim W$                                              'Work String

    P$ = ""                                         'Erase Previous Character
    SignFix = ""                                    'Assume No Expression given
    For CPnt = 1 To Len(Txt)
        W$ = Mid$(Txt, CPnt, 1)                     'Extract a Character
        Select Case P$ + W$                         'Resolve any Bad Pairs
            Case Is = "++": P$ = "+": W$ = ""
            Case Is = "+-": P$ = "-": W$ = ""
            Case Is = "-+": P$ = "-": W$ = ""
            Case Is = "--": P$ = "+": W$ = ""
            Case Is = ")(": P$ = ")*(": W$ = ""
        End Select
        If Len(W$) <> 0 Then                        'When not a known Pair
            SignFix = SignFix + P$                  'Hold Previous Character
            P$ = W$: W$ = ""                        'Remember Current Character
        End If
    Next CPnt
    SignFix = SignFix + P$ + W$                     'Append any omitted residue characters
End Function


Private Function BrackTst(Txt As String) As String
Rem This routine will check that all bracket pairs match and
Rem Also remove all space and tabs

Rem Modification Record
Rem Use own internal Function Stack                 29/Jly/2019 RSP
Rem TBA                                             05/Jun/2019 RSP

Dim CPnt As Integer                                 'Character pointer
Dim SPnt As Integer                                 'Stack Pointer
Dim C$                                              'Work String
Dim W$                                              'Work String
Dim StkF(StkMax) As String                          'Create a Calculation Function Stack

    BrackTst = ""                                   'Assume no Errors
    C$ = ""                                         'Clean String Cleared
    SPnt = 0                                        'Initalise stack
    For CPnt = 1 To Len(Txt)
```

```
        W$ = Mid$(Txt, CPnt, 1)                     'Extract a character
        If W$ = Chr$(9) Or W$ = " " Then W$ = ""    'Remove Tabs and spaces
        If W$ = "(" Or W$ = "[" Then                'Opening Bracket Matching
            If SPnt > StkMax Then                   'Too many Opening Brackets
                'BrackTst = "Err-(["                'Show Bracket Excess Opening
                SPnt = 1                            'Set Stack to show Error
                Exit For                            'Tests Complete
            End If
            StkF(SPnt) = InStr("([", W$)            'Remember Opening Bracket
            SPnt = SPnt + 1                         'Move stack pointer
        End If
        If W$ = ")" Or W$ = "]" Then                'Closing Bracket Matching
            SPnt = SPnt - 1                         'Move stack pointer
            If SPnt < 0 Then                        'Check excess end brackets
                BrackTst = "Err-)]"                 'Show Bracket Excess Closing
                SPnt = 0                            'Reset stack
                Exit For                            'Tests Complete
            End If

            If StkF(SPnt) <> InStr(")]", W$) Then   'Check Open matches End Bracket
                BrackTst = "Err-" + W$              'Show Bracket Conflict
                SPnt = 0                            'Reset stack
                Exit For                            'Tests Complete
            End If
        End If
        C$ = C$ + W$                                'Build Up Cleaned String
    Next CPnt

    If SPnt <> 0 Then
        BrackTst = "Err-(["                         'Show Bracket Excess Opening
    End If

    Txt = C$                                        'Update entered String with Clean Version
End Function



Private Function ModFunID(Txt As String) As String
Rem This routine will Modify Function Names so they all start
Rem with a Tilde ~ character and a unique function code.
Rem Where Constant Functions are defined their value is subsituted
Rem This routine works in conjunction with Function FunCalc
Rem that allows function nesting feature.

Rem Modification Record
Rem Total redesign (To Replaces routine Fun2Num)    07/Sep/2019 RSP
Rem Function ID code changed to ~ numeric form      09/Sep/2019 RSP
Rem TBA                                             05/Jun/2019 RSP

Rem
Rem Supported Functions that can be processed are :-
Rem PI()                    'Will be replaced by its value
Rem ACos(CosSine Value)     'Arc Cosine  of Angle
Rem ASin(Sine Value)        'Arc Sine    of Angle
Rem Atan(Tangent Value)     'Arc Tangent of Angle
Rem Cos(Angle in Radians)   'CoSine of Angle
Rem D2R(Angle in Degrees)   'Convert Degrees to Radians
Rem Exp(Num)                'Raise e to power Num (Antilog)
Rem Log(Num)                'Calculate Natural Log of Num
Rem R2D(Angle in Radians)   'Convert Radians to Degrees
Rem Sin(Angle in Radians)   'Sine of Angle
Rem Sqr(Num)                'Calculate square root of Num
Rem Tan(Angle in Radians)   'Tangent of Angle

Dim ExpS As String                                  'Start text of expression
Dim ExpE As String                                  'End   text of expression
Dim FunId As String                                 'Function Identity Name
Dim FunType As Integer                              'Current Function Type number
Dim FunRep As String                                'Function replacement string
Dim PntB As Integer                                 'Start ( Position pointer
Dim PntE As Integer                                 'End   ) Position pointer
Dim PntS As Integer                                 'Start Position pointer
Dim FunSet                                          'An array used to store Function Names

    If InStr(Txt, "~") <> 0 Then                    'Check Expression for a Function flag
        ModFunID = "ErrExp"                         'Error Expression contains Function flag
        Exit Function                               'Immediate Exit
    End If
    ModFunID = ""                                   'Assume Error Free
    Txt = UCase(Txt)                                'Convert expression to Uppercase
```

```
    Rem Initially Scan/Remove any Alpha function names and replace with ~Function Code variants.
    Rem ============================================================================================
    Rem
    Rem Note last array parameter must be "END" or app will loop forever

    Rem  FunType =  0       1       2       3      4      5      6      7      8      9      10
    FunSet = Array("ACOS", "ASIN", "ATAN", "COS", "SIN", "TAN", "END")
    FunType = 0                                    'Start at beginning of Function list ie. ACOS
    Call RepFunId(FunSet, 0, Txt)                  'Update Function IDs ~00 to ~05

    Rem  FunType =  6      7      8      9      10
    FunSet = Array("EXP", "LOG", "SQR", "D2R", "R2D", "END")
    FunType = 6                                    'Start at beginning of Function list ie. EXP
    Call RepFunId(FunSet, 6, Txt)                  'Update Function IDs ~06 to ~10

    Rem Next remove any Zero Parameter Constant Functions.
    Rem ================================================
    Rem
    Rem FunType =   0     1      2      3      4      5      6      7      8      9
    FunSet = Array("PI", "END")
    Do                                             'Scan the Equation
        FunType = 0                                'Start at beginning of Function list ie. Pi
        Do                                         'Scan the functions list
            FunId = FunSet(FunType)                'Extract a Function ID
            If FunId = "END" Then Exit Do          'Check if Scan now complete
            PntS = InStr(Txt, FunId + "(")         'Try to Locate Function code
            If PntS <> 0 Then Exit Do              'When Function Found stop scanning
            FunType = FunType + 1                  'Move on to next Function
        Loop

        If FunId = "END" Or PntS = 0 Then Exit Do  'Scanning complete
        PntB = PntS + Len(FunId)                   'Locate Start Bracket Point
        PntE = InStr(PntS, Txt, ")")               'Locate End   Bracket Point

        If PntE - PntB <> 1 Then                   'Check for empty Brackets
            ModFunID = "ErrFun"                    'Error Invalid Function (Structure ?)
            Exit Function
        End If

        ExpS = "": ExpE = "": PntE = PntE + 1      'Assume expression has no prefix or postfix
        If PntS > 1 Then
            ExpS = Mid$(Txt, 1, PntS - 1)          'Extract Expression Prefix
        End If
        If PntE <= Len(Txt) Then
            ExpE = Mid$(Txt, PntE)                 'Extract Expression Postfix
        End If

        Select Case FunType                        'Locate Insertion/Conversion
        Case Is = 0                                'Insert Pi
            FunRep = V2S$(Pi)                      'Convert defined Pi value to string
        End Select

        Rem Update the Equation expression
        Txt = ExpS + FunRep + ExpE                 'Replace the Function with value
'DoEvents    'Debug Break
    Loop

End Function


Private Sub RepFunId(FunSet, FunBase As Integer, Txt As String)
Rem This routine will Modify Function Names so they all start
Rem with a Tilde ~ character and a unique function code.

Rem Modification Record
Rem Initial Design                              07/Sep/2019 RSP
Rem Function ID code changed to ~ numeric form  09/Sep/2019 RSP
Rem Adapted to support extend set of names      12/Sep/2019 RSP
Rem TBA                                         05/Jun/2019 RSP

Dim ExpS As String                              'Start text of expression
Dim ExpE As String                              'End   text of expression
Dim FunId As String                             'Function Identity Name
Dim FunNewId As String                          'Function New Identity Name
Dim FunType As Integer                          'Current Function Type number
Dim PntE As Integer                             'End of Name Position pointer
Dim PntS As Integer                             'Start Position pointer

    Do                                          'Scan the Expression Line
        FunType = 0                             'Start at beginning of list
        Do                                      'Scan the List of Functions
```

```
            FunId = FunSet(FunType)                 'Extract a Function ID
            If FunId = "END" Then Exit Do           'Check if Scan now complete
            PntS = InStr(Txt, FunId + "(")          'Try to Locate Function code
            If PntS <> 0 Then Exit Do               'When Function Found stop scanning
            FunType = FunType + 1                   'Move on to next Function
        Loop
        If FunId = "END" Or PntS = 0 Then Exit Do   'Scanning complete
        FunNewId = V2S$(FunType + FunBase)          'Calculate New Function Identity Name
        If Len(FunNewId) < 2 Then                   'Make Function Name 3 Characters Long
            FunNewId = "~0" + FunNewId
        Else
            FunNewId = "~" + FunNewId
        End If
        PntE = PntS + Len(FunId)                    'Calculate end point of function name
        ExpS = "": ExpE = ""                        'Assume expression has no prefix or postfix
        If PntS > 1 Then
            ExpS = Mid$(Txt, 1, PntS - 1)           'Extract Expression Prefix
        End If
        If PntE <= Len(Txt) Then
            ExpE = Mid$(Txt, PntE)                  'Extract Expression Postfix
        End If
        Txt = ExpS + FunNewId + ExpE                'Replace the marked function name
'DoEvents    'Debug Break
    Loop
End Sub



Private Function FunCalc(Tkn As String, Reply As String) As Boolean
Rem This routine will subsitute Functions for their numeric value

Rem Modification Record
Rem Initial Design that allows function nesting     06/Sep/2019 RSP
Rem Configured for new Function ID codes            09/Sep/2019 RSP
Rem Example code for 2 parameter functions          12/Sep/2019 RSP
Rem TBA                                             05/Jun/2019 RSP

Rem
Rem Functions that can be processed are :-
Rem ACos(CosSine Value)     'Arc Cosine  of Angle
Rem ASin(Sine Value)        'Arc Sine    of Angle
Rem Atan(Tangent Value)     'Arc Tangent of Angle
Rem Cos(Angle in Radians)   'CoSine of Angle
Rem Sin(Angle in Radians)   'Sine of Angle
Rem Tan(Angle in Radians)   'Tangent of Angle
Rem Exp(Num)                'Raise e to power Num (Antilog)
Rem Log(Num)                'Calculate Natural Log of Num
Rem Sqr(Num)                'Calculate square root of Num
Rem D2R(Angle in Degrees)   'Convert Degrees to Radians
Rem R2D(Angle in Radians)   'Convert Radians to Degrees

Dim FunCde As Integer                           'Function Code
Dim FunPar As String                            'Function Parameter string
Dim FunVal As Double                            'Function Reply Value
Dim FunVal1 As Double                           'Function Value Parameter 1
Rem Just in case you wish to add functions with more than one parameter
'Dim FunVal2 As Double                           'Function Value Parameter 2
'Dim FunVal3 As Double                           'Function Value Parameter 3
'Dim FunVal4 As Double                           'Function Value Parameter 4

    FunCalc = True                              'Assume Function processed OK

    Rem                 Functions with One or More Parameters
    If Stack(StkPop, FunPar) = False Then       'Get Parameters 1 from the Stack
        Reply = "ErrStk"                        'Error message
        FunCalc = False                         'Problem with Conversion
        Exit Function                           'All Work Done
    End If
    FunVal1 = Val(FunPar)                       'Hold Parameter as a number Value

'    Rem Just an example of how you might split up functions with many parameters
'    If FunCde >= 50 Then
'        Rem             Functions with Two or More Parameters
'        If Stack(StkPop, FunPar) = False Then   'Get Parameters 2 from the Stack
'            Reply = "ErrStk"                    'Error message
'            FunCalc = False                     'Problem with Conversion
'            Exit Function                       'All Work Done
'        End If
'        FunVal2 = Val(FunPar)                   'Hold Parameter as a number Value
'    End If
```

```
    FunCde = Val(Mid(Tkn, 2))                   'Extract Function Code
    Select Case FunCde
    Case Is = 0                                 'ACOS   ~00
        FunVal = ACos(FunVal1)                  'Convert Value
    Case Is = 1                                 'ASIN   ~01
        FunVal = ASin(FunVal1)                  'Convert Value
    Case Is = 2                                 'ATAN   ~02
        FunVal = Atn(FunVal1)                   'Convert Value
    Case Is = 3                                 'COS    ~03
        FunVal = Cos(FunVal1)                   'Convert Value
    Case Is = 4                                 'SIN    ~04
        FunVal = Sin(FunVal1)                   'Convert Value
    Case Is = 5                                 'TAN    ~05
        FunVal = Tan(FunVal1)                   'Convert Value
    Case Is = 6                                 'EXP    ~06
        FunVal = Exp(FunVal1)                   'Convert Value
    Case Is = 7                                 'LOG    ~07
        FunVal = Log(FunVal1)                   'Convert Value
    Case Is = 8                                 'SQR    ~08
        FunVal = Sqr(FunVal1)                   'Convert Value
    Case Is = 9                                 'D2R    ~09
        FunVal = FunVal1 * Pi / 180             'Convert Value
    Case Is = 10                                'R2D    ~10
        FunVal = FunVal1 * 180 / Pi             'Convert Value

'    Case Is = 50                                'Function with 2 parameters
'        FunVal = FunVal1 + FunVal2              'An Add Function maybe ?

    Case Else
        Reply = "ErrFun"                        'Error message
        FunCalc = False                         'Unknown Function
        Exit Function                           'All Work Done
    End Select

    Reply = V2S$(FunVal)                        'Return converted value

End Function



Private Function ACos(Ang As Double)
Rem Inverse CoSine Routine
Rem This routine does NOT exist as an Intrinsic VB Function
Rem Ang is the Cosine of the angle you want and must be from -1 to 1.
Rem This routine returns an angle in radians in the range 0 to Pi
Rem Author R. J. Spriggs  Creation Date 04/12/2005  Last update 12/12/05
Rem Mod RJS 10/12/2005 90 degree (x=1) check and correct
Rem Mod RJS 12/04/2006 Algorithm error corrected
Rem Mod RJS 08/08/2012 Stop Removed, Invalid value limited to Max or Min

Dim X As Double

    X = Ang                                             'Hold Entry Value
    'If X > 1 Or X < -1 Then Stop                       'Value outside valid range
    If X > 1 Then X = 1                                 'Limit to Max +0 Degrees
    If X < -1 Then X = -1                               'Limit to Min +180 Degrees
    If X = 1 Or X = -1 Then                             'Check if Range end Degrees
        ACos = (1 - X) * Pi / 2
    Else
        'Ang = x / Sqr(-x * x + 1)
        'ACos = Atn(Ang) + Pi / 2
        'ACos = Atn(x / Sqr(-x * x + 1)) + Pi / 2       'removed 12/04/2006
        ACos = (Pi / 2) - Atn(X / Sqr(-X * X + 1))      'added 12/04/2006
    End If
End Function


Private Function ASin(Ang As Double)
Rem Inverse Sine Routine
Rem This routine does NOT exist as an Intrinsic Function
Rem Ang is the Sine of the angle you want and must be from -1 to 1.
Rem This routine returns an angle in radians in the range -Pi/2 and Pi/2
Rem Author R. J. Spriggs  Creation Date 04/12/2005  Last update 08/08/12
Rem Mod RJS 10/12/2005 90 degree (x=1) check and correct
Rem Mod RJS 30/04/2012 Invalid entry = Responce=0 to allow routine to continue (removed)
Rem Mod RJS 08/08/2012 Stop Removed, Invalid value limited to Max or Min

Dim X As Double

    X = Ang                                             'Hold Entry Value
```

```
    'If X > 1 Or X < -1 Then Stop                       'Value outside valid range
    If X > 1 Then X = 1                                 'Limit to Max +90 Degrees
    If X < -1 Then X = -1                               'Limit to Min -90 Degrees
    'If X > 1 Or X < -1 Then                            'Value outside valid range
    '    ASin = 0: Exit Function                        'A wrong Answer
    'End If
    If X = 1 Or X = -1 Then                             'Check if Range end Degrees
        ASin = (Pi / 2) * X
    Else
        ASin = Atn(X / Sqr(-X * X + 1))
    End If
End Function


Rem
Rem ==============================
Rem Memory Store Manager Routines.
Rem ==============================
Rem

Public Function MSM(Act As Integer, CellInfo As String) As Boolean
Rem
Rem This routine will perform a number of Cell Memory Store Management Functions
Rem     It    can Initialise the Cell Store (Erase All)
Rem     Or it can Create/Load Information in a cell store location
Rem     Or it can Erase                      a cell store location
Rem     Or it can Tidy/Compact all             cell store locations
Rem     Or it can Extract Information from   a cell store location
Rem     Or it can Update  Information in     a cell store location
Rem     Or it can Update the Acumulator        cell store location A0
Rem     Or it can Select                     a cell store location
Rem     Or it can produce a debug print of     cell store locations
Rem Modification Record
Rem Initial Design                                  05/Jun/2019 RSP
Rem Revised to be more similar to Stack and Queue   02/Aug/2019 RSP
Rem and renamed to MSM (Memory Store Manager)       02/Aug/2019 RSP
Rem Routine Now Public                              14/Aug/2019 RSP
Rem TBA                                             05/Jun/2019 RSP

Rem Error Message/Codes when False in parameter CellInfo
Rem Err[?]          Cell Address Invalid
Rem ErrCMax         Storage structure Full
Rem ErrCFun         Unknown Cell Storage structure Function


Static CellV(CellMax) As String                     'Create a Cell Values Store
Static CellN(CellMax) As String                     'Create a Cell Name   Store
Static CellCnt As Integer                           'Current Number of Cells
Static CurCell As Integer                           'Current Selected Cell Number

Dim Cnt As Integer                                  'General Work Counter
Dim Pnt As Integer                                  'General Work Pointer
Dim CPnt As Integer                                 'General Work Pointer
Dim W$                                              'General Work  String
Dim Item As String                                  'General Work  String
Dim ICell As String                                 'General Work  String
Dim NCell As String                                 'General Work  String (Holds a Cell Name)
Dim PCell As String                                 'General Work  String
Dim VCell As String                                 'General Work  String (Holds a Cell Value)
Dim Igs As String                                   'Ignore Characters String

    MSM = True                                      'Assume all operation have worked well
    Igs = " " + Chr$(9)                             'Hold Space and Tab Character Ignore Sequence

    Select Case Act                                 'Check out what to do
    Case Is = MSMLdA                                'Places Data in the Acumulator Cell
        CellV(0) = CellInfo                         'Load Acumulator Cell with Data

    Case Is = MSMRdA                                'Read Data from the Acumulator Cell
        CellInfo = CellV(0)                         'Read contents of the Acumulator Cell

    Case Is = MSMSel, MSMCre                        'Scan for a Selected Cell or Create a Cell
        W$ = SmartUC(CellInfo, Igs)                 'Convert Cell Name characters to Tidy UpperCase
        CurCell = -1                                'Set Current Cell address as invalid
        For Pnt = 0 To CellCnt                      'Check all allocated Cell names
            If CellN(Pnt) = W$ Then
                CurCell = Pnt                       'Hold selected Cell address
                Exit For                            'Job Done
            End If
        Next Pnt
```

```
        If CurCell < 0 Then                         'When Cell Address invalid
            If Act = MSMSel Then
                MSM = False                         'Cell Address unknown
                CellInfo = "Err[?]"                 'Cell Address unknown

            ElseIf CellCnt < CellMax Then
                CellCnt = CellCnt + 1               'Create a new Cell
                CellN(CellCnt) = W$                 'Load Cell with it Cell name
                CellV(CellCnt) = ""                 'Flush any Cell Data
                CurCell = CellCnt                   'Remember Cell Address
            Else
                MSM = False                         'Selection/creation Failed
                CellInfo = "ErrCMax"                'Cell Storage Area full
            End If
            If CurCell < 0 Then CurCell = 0         'Select Acumulator when Error
        End If

    Case Is = MSMRdS, MSMLdS
        If CurCell < 0 Then                         'Check Cell Address Valid
            MSM = False                             'Load Failed
            CellInfo = "Err[?]"                     'Cell Address Invalid
        Else
            Select Case Act
            Case Is = MSMRdS                        'Read Data from the Selected Cell
                CellInfo = CellV(CurCell)           'Extract Cell Data

            Case Is = MSMLdS                        'Places Data in the Selected Cell
                CellV(CurCell) = CellInfo           'Load Cell with Data
            End Select
        End If

    Case Is = MSMDel                                'Erase/Delete the Selected Cell
        If CurCell > 0 Then                         'Check Cell Address Valid
            CellN(CurCell) = ""                     'Erase the Celll
        Else
            MSM = False                             'Load Failed
            CellInfo = "Err[?]"                     'Cell Address Invalid
        End If

    Case Is = QSMInit                               'Select when Initialise requested
        CellCnt = 0                                 'Indicate Number of additional Cells Defined
        CellN(0) = "A0"                             'Acumulator Cell (Invalid Spreadsheet Address)
        CellV(0) = "0"                              'Acumulator Contents
        CurCell = 0                                 'Assume Current Selected Cell is Acumulator
        For Pnt = 1 To CellMax                      'Repeat for whole of structure
            CellN(Pnt) = ""                         'Erase Names List
            CellV(Pnt) = ""                         'Erase Value List
        Next Pnt

    Case Is = MSMTdy                                'Select when Memory Tidy requested
        Cnt = 0                                     'Assume only Acum Cell defined
        For Pnt = 1 To CellCnt                      'Only Scan Previously Defined cell list
            If Len(CellN(Pnt)) <> 0 Then            'Only process when Cell Exists
                Cnt = Cnt + 1                       'Point at next free Cell
                If Cnt <> Pnt Then                  'When Data is to be moved
                    CellN(Cnt) = CellN(Pnt)         'Shuffel Cell Name down Table
                    CellV(Cnt) = CellV(Pnt)         'Shuffel Cell Data down Table
                    CellN(Pnt) = ""                 'Release the Old Cell
                    CellV(Pnt) = ""                 'Release the Old Cell
                End If
            End If
        Next Pnt
        CellCnt = Cnt                               'Indicate number of Cells now Defined
        CurCell = 0                                 'Select Acumulator as default Selected


Rem All this Case code may be removed at run time  if Debug not needed
    Case Is = QSMDebug                              'Produce Debug Print
        Rem Display state of all Cell Entries
        Item = NewLine                              'Start on a Fresh Line
        Item = Item + "Contents of Cell Manager Structure" + NewLine
        Item = Item + "Cell Count =" + Str$(CellCnt) + NewLine

        For Pnt = 0 To CellCnt
            Item = Item + V2S$(Pnt) + Chr$(9) + "CellN() >" + CellN(Pnt) + "<" + Chr$(9)
            Item = Item + "CellV() >" + CellV(Pnt) + "<" + NewLine
        Next Pnt
        CellInfo = Item + NewLine
        Exit Function
Rem End Remove
```

```
    Case Is = MSMCmd                                'Command Checker Feature

        Rem Initially check for an assignment request
        ICell = SmartUC(CellInfo, Igs)              'Tidy the entry Command line
        CPnt = InStr(ICell, ":=")                   'Look for assignment symbol
        If CPnt <> 0 Then                           'Check for an Assignment Command
            If CPnt > 2 Then                        'Check for a Cell assignment request
                NCell = Mid(ICell, 1, CPnt - 1)     'Extract Cell Name
                VCell = Mid$(ICell, CPnt + 2)       'Extract the Value
                If MSM(MSMCre, NCell) = True Then   'Select Cell then Load it
                    If MSM(MSMLdS, VCell) = True Then
                        CellInfo = "OK"             'Indicate Loading OK
                        Exit Function
                    End If
                End If
            End If
            CellInfo = "Err[?]"
            MSM = False                             'Give a Fail reply
            Exit Function
        End If


        CPnt = InStr(ICell, "<-")                   'Look for Move From Cell Request Trigger
        If CPnt = 1 Then                            'Check for the Trigger
            NCell = Mid(ICell, 3)                   'Extract Cell Name
            If MSM(MSMSel, NCell) = True Then       'Select Cell when OK Make Transfer
                CellV(0) = CellV(CurCell)           'Acum = Cell Contents
                CellInfo = "OK"                     'Indicate Loading OK
            Else
                CellInfo = "Err[?]"
                MSM = False                         'Give a Fail reply
                Exit Function
            End If
        End If


        CPnt = InStr(ICell, "->")                   'Look for Move To   Cell Request Trigger
        If CPnt = 1 Then                            'Check for the Trigger
            NCell = Mid(ICell, 3)                   'Extract Cell Name
            If MSM(MSMSel, NCell) = True Then       'Select Cell when OK Make Transfer
                CellV(CurCell) = CellV(0)           'Cell = Acum Contents
                CellInfo = "OK"                     'Indicate Loading OK
            Else
                CellInfo = "Err[?]"
                MSM = False                         'Give a Fail reply
                Exit Function
            End If
        End If


        If InStr(ICell, "ERASE,") = 1 Then          'Check for ERASE Command
            NCell = Mid(ICell, 7)                   'Extract Cell Name
            If MSM(MSMSel, NCell) = True Then       'Select Cell then delete it
                If MSM(MSMDel, NCell) = True Then
                    CellInfo = "OK"                 'Indicate Deletion OK
                    Exit Function
                End If
            End If
            CellInfo = NCell
            MSM = False                             'Give a reply
        End If


        If InStr(ICell, "TIDY") = 1 Then            'Check for Tidy Command
            If MSM(MSMTdy, PCell) = True Then CellInfo = "OK"
        End If


        If InStr(ICell, "INIT") = 1 Then            'Check for Initialise Command
            If MSM(QSMInit, PCell) = True Then CellInfo = "OK"
        End If


    Case Else
        MSM = False                                 'Unknown Cell Manager Function
        CellInfo = "ErrCFun"

    End Select
End Function
```

```
Public Function Queue(Mode As Integer, Item As String) As Boolean
Rem This routine will manage a Data Queue and Flag any errors
Rem Features available
Rem 1.  Initialise the Queue structure
Rem 2.  Put    data in the Queue
Rem 3.  Remove data from the Queue
Rem 4.  Identify how many items in the Queue
Rem 5.  View contents at Head of Queue

Rem Modification Record
Rem Initial Design                                  29/Apr/2019 RSP
Rem Added items 4. & 5.                             27/Jly/2019 RSP
Rem Routine Now Public                              14/Aug/2019 RSP
Rem TBA                                             05/Jly/2019 RSP

Static Que(QueMax) As String                        'Define the Queue Storage area
Static QCnt As Integer                              'Define Items in Queue counter
Static QAddPnt As Integer                           'Define the Queue Add pointer
Static QRemPnt As Integer                           'Define the Queue Rem pointer

    Queue = True                                    'Assume Queue Operation Processed OK
    If QAddPnt > QueMax Then QAddPnt = 0            'Keep Add Pointer in Range
    If QRemPnt > QueMax Then QRemPnt = 0            'Keep Rem Pointer in Range

    Select Case Mode
    Case Is = QueAdd                                'Add a value onto the Queue
        If QCnt <= QueMax Then                      'Check if data can be added
            Que(QAddPnt) = Item                     'Store the Data Value
            QAddPnt = QAddPnt + 1                   'Update Queue Add pointer
            QCnt = QCnt + 1                         'One More item in the Queue
        Else
            Queue = False                           'Queue Operation failed
        End If

    Case Is = QueRem                                'Remove a value from the Queue
        If QCnt > 0 Then                            'Check if data can be Removed
            Item = Que(QRemPnt)                     'Extract the Data Value
            QRemPnt = QRemPnt + 1                   'Update Queue Remove pointer
            QCnt = QCnt - 1                         'One Less item in the Queue
        Else
            Queue = False                           'Queue Operation failed
        End If

    Case Is = QSCnt                                 'Identify number of items in the Queue
        Item = Str$(QCnt)                           'Number of items in Queue

    Case Is = QSPeek                                'Identify items at Head of the Queue
        Item = Que(QRemPnt)                         'Extract the Data Value

    Case Is = QSMInit                                'Initialise the Queue
        For QAddPnt = 0 To QueMax                   'Flush out any prior data in Queue
            Que(QAddPnt) = ""                       'Store a default Empty Data Value
        Next QAddPnt
        QCnt = 0                                    'Indicate no items in Queue
        QAddPnt = 0                                 'Set up Queue Add    pointer
        QRemPnt = 0                                 'Set up Queue Remove pointer

Rem All this Case code can be removed in run time build  if Debug not needed
    Case Is = QSMDebug                              'Produce Debug Printout
    Dim Pnt
        Rem Display state of all Queue Controls and Entries
        Item = Item + NewLine                       'Start on a Fresh Line
        Item = Item + "Contents of Queue Structure" + NewLine
        Item = Item + "QCnt    =" + Str$(QCnt) + NewLine
        Item = Item + "QAddPnt =" + Str$(QAddPnt) + NewLine
        Item = Item + "QRemPnt =" + Str$(QRemPnt) + NewLine + NewLine

        For Pnt = 0 To QueMax
            Item = Item + V2S$(Pnt) + Chr$(9) + " >" + Que(Pnt) + "<" + NewLine
        Next Pnt
Rem End Remove

    Case Else
        Queue = False                               'Unknown Queue Mode
    End Select

End Function
```

```
Public Function Stack(Mode As Integer, Item As String) As Boolean
Rem This routine will manage a Data Stack and Flag any errors
Rem Features available
Rem 1.  Initialise the Stack structure
Rem 2.  Put    data on   the Stack
Rem 3.  Remove data from the Stack
Rem 4.  Identify how many items in the Stack
Rem 5.  View contents at Top of Stack

Rem Initial Design                                  29/Apr/2019 RSP
Rem Added items 4. & 5.                             27/Jly/2019 RSP
Rem Mod to ensure stack can be fully filled         02/Aug/2019 RSP
Rem Routine Now Public                              14/Aug/2019 RSP

Static Stk(StkMax) As String                        'Define the Stack Storage area
Static StkPnt As Integer                            'Define the Stack pointer
Const StkLmt = StkMax + 1                           'Define Max Items that can be held in stack

    Stack = True                                    'Assume Stack Operation Processed OK
    Select Case Mode
    Case Is = StkPush                               'Push a value onto the Stack
        If StkPnt >= 0 And StkPnt <= StkMax Then
            Stk(StkPnt) = Item                      'Store the Data Value
            StkPnt = StkPnt + 1                     'Update Stack/Heap pointer
        Else
            StkPnt = StkLmt                         'Reset up Stack/Heap pointer
            Stack = False                           'Stack Operation failed
        End If

    Case Is = StkPop                                'Pop  a value from the Stack
        If StkPnt > 0 And StkPnt <= StkLmt Then
            StkPnt = StkPnt - 1                     'Update Stack/Heap pointer
            Item = Stk(StkPnt)                      'Extract the Data Value
        Else
            StkPnt = 0                              'Reset up Stack/Heap pointer
            Stack = False                           'Stack Operation failed
        End If

    Case Is = QSCnt                                 'Identify number of items in the Stack
        Item = Str$(StkPnt)                         'Number of items in Stack

    Case Is = QSPeek                                'Identify items at Top of the Stack
        If StkPnt > 0 Then
            Item = Stk(StkPnt - 1)                  'Extract the Data Value
        Else
            Item = ""
        End If

    Case Is = QSMInit                               'Initialise the Stack
        For StkPnt = 0 To StkMax                    'Flush out any prior data in Stack
            Stk(StkPnt) = ""                        'Store a default Empty Data Value
        Next StkPnt
        StkPnt = 0                                  'Set up Stack/Heap pointer

Rem All this Case code can be removed in run time build if Debug not needed
    Case Is = QSMDebug                              'Produce Debug Printout
    Dim Pnt

        Rem Display state of all Stack Controls and Entries
        Item = Item + NewLine                       'Start on a Fresh Line
        Item = Item + "Contents of Stack Structure" + NewLine
        Item = Item + "StkPnt =" + Str$(StkPnt) + NewLine + NewLine

        For Pnt = 0 To StkMax
            Item = Item + V2S$(Pnt) + Chr$(9) + " >" + Stk(Pnt) + "<" + NewLine
        Next Pnt
 Rem End of Remove

    Case Else
        Stack = False                               'Unknown Stack Mode
    End Select

End Function




Rem ====================================================
Rem     For information only
```

```
Rem
Rem     Source code of routines located in other Modules.
Rem
Rem     Note :
Rem     When these routines are located in external modules
Rem     they needs'Private' to be changed to 'Public'
Rem
Rem ================================================


'Private Function NewLine$()
'Rem This routine will reply with a string Containing CR + LF
'     NewLine$ = Chr$(13) + Chr$(10)  'Reply with Line Termination string
'End Function
'

'Private Function V2S$(Value)
'Rem This routine will Convert Value to String (no prefix Space)
'Dim A$
'     If Value < 0 Then
'         A$ = Str$(Value)                    'Hold number converted to string
'     Else
'         A$ = Mid$(Str$(Value), 2)           'Hold number converted to string
'     End If
'     If Mid(A$, 1, 1) = "." Then             'Zero Prefix when Fraction only
'         A$ = "0" + A$
'     End If
'     V2S$ = A$                               'Reply with number converted to string
'End Function
'

'Private Function SmartUC(Mess As String, Igs As String) As String
'Rem This routine will tidy a message by converting all LC characters to UC
'Rem as well as removing any of the Ignore Characters and String Markers.
'Rem Additionaly this routine will skip areas marked as a string when
'Rem using text enclosed in Single or Double quotes enclosed pairs.
'Rem Example of marked Strings   'Hello World' or "Hello World"
'Rem so if
'Rem Igs = Tab character and a Space
'Rem and Mess = > 'String 1 ' and Tab Char " String " <
'Rem This function replys >String 1 AND String 2<
'
'Rem Initial Design                                  16/Mar/2001 RSP
'Rem Modified for use in VB6 applications            05/Aug/2019 RSP
'
'Const LCa = 97                                          'Hold code of LC "a"
'Const LCz = 122                                         'Hold code of LC "z"
'Const LCUC = LCa - 65                                   'Create offset convert constant
'Dim Pnt As Integer                                      'General Purpose Pointer
'Dim Num As Integer                                      'General Purpose Number
'Dim SMark As String                                     'String Start Marker
'Dim Char As String                                      'A Character store
'Dim OStr As String                                      'The output Work character store
'Dim WStr As String                                      'A Work character store
'
'     WStr = Mess                                       'Hold copy of entry string
'     OStr = ""                                         'Initialise Output String
'     SMark = ""                                        'Initialise String Marker
'     For Pnt = 1 To Len(Mess$)                         'Process the whole entry message
'         Char = Mid$(WStr, Pnt, 1)                     'Extract a Character
'         Num = Asc(Char)                               'Hold Ascii value of extracted Character
'
'         If Len(SMark) = 0 Then                        'Whilst not in a string
'             If Char = "'" Or Num = 34 Then            'Check if this is a string starter
'                 SMark = Char                          'Remember Start Code
'                 Char = ""                             'Remove the Trigger
'             End If
'             If Len(Igs) <> 0 Then                     'Check for an Ignore character
'                 If InStr(Igs, Char) <> 0 Then         'When Ignore Character foung
'                     Char = ""                         'Remove Character
'                 End If
'             End If
'             If Len(Char) <> 0 Then
'                 If Num >= LCa And Num <= LCz Then     'Check for LC Character
'                     Num = Num - LCUC                  'Convert Case of Character
'                     Char = Chr$(Num)                  'Hold Converted Character
'                 End If
'             End If
'         Else                                          'Processing a String
'             If Char = SMark Then                      'Check if this is the string Terminator
'                 SMark = ""                            'Indicate out of a string
```

```
'                Char = ""                              'Flush the delimiter
'            End If
'        End If
'        OStr = OStr + Char                             'Build up Output string
'    Next Pnt
'    SmartUC = OStr                                     'Reply with converted string
'End Function
'
```