Application Demonstration

OpenMaple API for VisualBasic and Java

Maplesoft, a division of Waterloo Maple Inc., 2004

Introduction

Using the enhanced OpenMaple API in Maple 9.5, you can embed real-time calls to Maple routines inside VisualBasic, Java and C programs. OpenMaple is a suite of VB, Java and C routines that gives programs written in those languages access to Maple routines, both built-in and user-defined. OpenMaple is the reverse of ExternalCalling, which gives Maple programs access to external data structures in VB, Java, C, MATLAB and Fortran. This worksheet provides commented source code of VB and Java programs that use the OpenMaple API. They are also available in the **samples/OpenMaple** directory that comes with Maple 9.5.

VisualBasic API

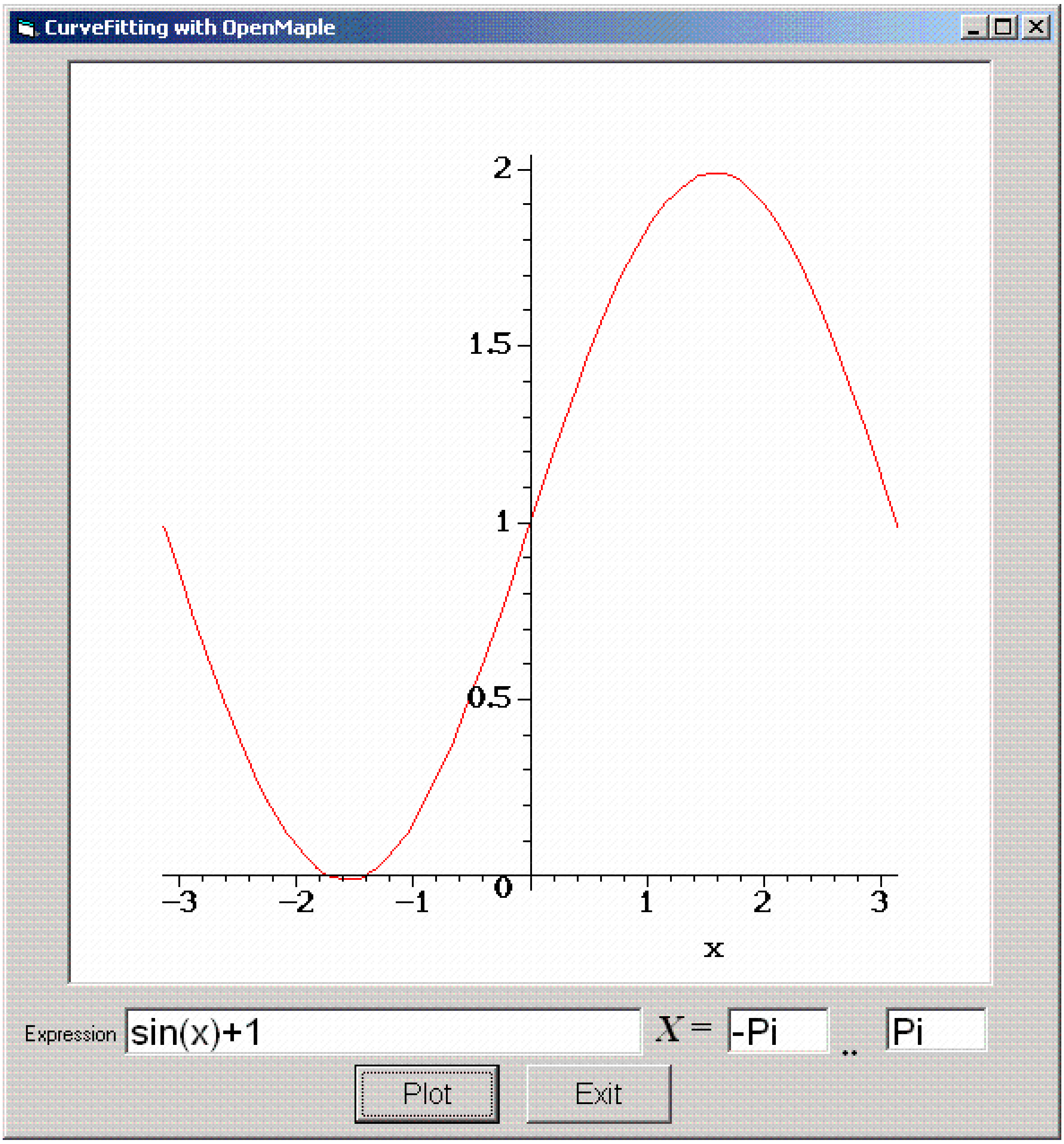
Plotting Example

Here is a simple VB program that uses Maple to plot a function in a window.

It is composed of one form called **MainForm** and two modules called **PlotterModule** and **MAPLE** module. The **MAPLE** module is available in the **extern/include** directory in the Maple 9.5 directory.

MainForm

The **MainForm** has an input box for the expression to plot and the horizontal axis limits. The user types in the expression and clicks on the Plot button to redraw the plot.



sin(x)+1

-Pi

Pi

Source Code

```
Public kv As Long 'kv (kernel vector) is the handle
for the Maple kernel

Public PlotFileName As String

Private Sub Form_Load()
   Dim error As String
   Dim args(0 To 1) As String

   'init callbacks
   cb.lpTextCallBack = GetProc(AddressOf TextCallBack)

   cb.lpErrorCallBack = GetProc(AddressOf
ErrorCallBack)
   cb.lpStatusCallBack = 0
   cb.lpReadLineCallBack = 0
   cb.lpRedirectCallBack = 0
   cb.lpQueryInterrupt = 0
   cb.lpCallBackCallBack = 0

   'start Maple by initiating kv with OpenMaple's
StartMaple command.
   kv = StartMaple(0, args, cb, 0, error)
   If kv = 0 Then
       MsgBox "Error starting Maple: " + StrConv
(error, vbUnicode), vbCritical, ""
       Unload Me
       End
   End If

   PlotFileName = ""

End Sub

Private Sub Form_Unload(Cancel As Integer)
   StopMaple kv
End Sub

Private Sub ExitButton_Click()
   StopMaple kv
   End
End Sub

Private Sub PlotButton_Click()
   Dim TempFile, command As String
   Dim temppath(256) As Byte
   Dim DirLen, r As Long

   'get a temporary file name for the plot
   DirLen = GetTempPath(256, temppath(0))
   If DirLen <> 0 Then
       TempFile = Left(StrConv(temppath(), vbUnicode),
DirLen) & "plot" & Int(Rnd() * 1000) & ".gif"
   End If

   'set up the plot to export as a .gif file named
TempFile
   'Note: kv is always an argument to
EvalMapleStatement
```

```
   Call EvalMapleStatement(kv, "plotsetup
('gif','plotoutput'=`" + TempFile + "`);")

   'generate the plot using the Maple plot command
   r = EvalMapleStatement(kv, "plot(" &
FunctionBox.Text & ", x=(" & LowerRange.Text & ")..("
& UpperRange.Text & "));")
   If r <> 0 Then
       Picture1.Picture = LoadPicture(TempFile)
       If PlotFileName <> "" Then
           Kill PlotFileName
       End If
   End If
   PlotFileName = TempFile
End Sub
```

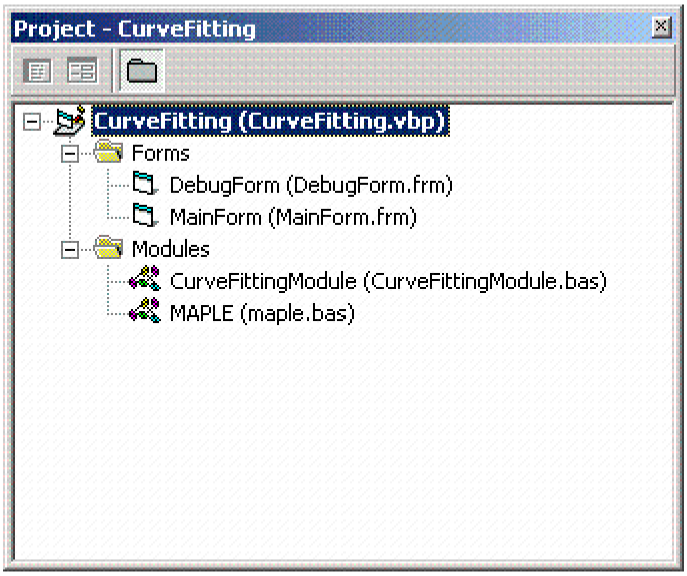
PlotterModule

```
Public cb As MapleCallBack
Public Declare Function GetTempPath Lib "kernel32.dll"
Alias "GetTempPathA" _
 (ByVal nBufferLength As Long, ByRef lpBuffer As Byte)
As Long

'Output Callback
Public Sub TextCallBack(data As Long, ByVal tag As
Integer, ByVal output As Long)
   Dim OutputString As String
   OutputString = MaplePointerToString(output)
End Sub

'Error Callback
Public Sub ErrorCallBack(data As Long, ByVal Offset As
Integer, ByVal output As Long)
   MsgBox " at offset " + str(Offset) + " " +
MaplePointerToString(output), vbInformation, ""
End Sub
```

Curve Fitting Example

This VB program allows the user to create a series of points in the plane using the mouse and then use Maple to fit different curves to them. This example is available in the **samples/OpenMaple** directory.

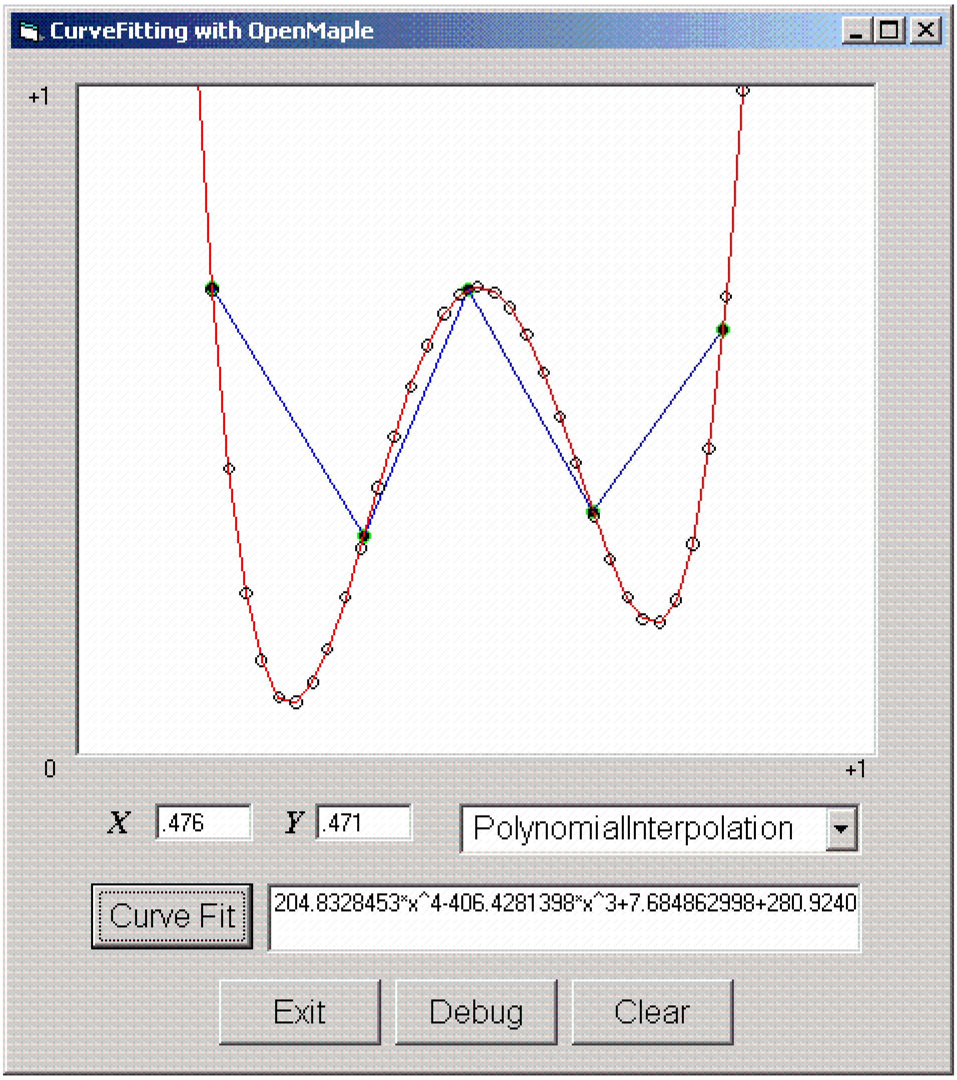
It is composed of two forms, **MainForm** and **DebugForm,** and two modules, **CurveFittingModule** and **MAPLE** module. The **MAPLE** module is available in the **extern/include** directory in the Maple 9.5 directory.



CurveFitting (CurveFitting.vbp)
  Forms
    DebugForm (DebugForm.frm)
    MainForm (MainForm.frm)
  Modules
    CurveFittingModule (CurveFittingModule.bas)
    MAPLE (maple.bas)

MainForm

The **MainForm** is the main window that is launched when the program is started. The user clicks in the window to add points, which appear in blue. The current x and y position of the cursor is also shown as the user moves the cursor around the window. The user selects a type of interpolation and presses the Curve Fit button, which generates the fitted curve (shown in red) and the corresponding equation. The Debug button takes the user to the debug form, which shows the values of the points and the output from the Maple kernel. Exit and clear buttons exit the program and clear the points.



X .476

Y .471

PolynomialInterpolation

204.8328453*x^4-406.4281398*x^3+7.684862998+280.9240

```
'kv is the OpenMaple MKernelVector handle
Public kv As Long

'coordinates to last point drawn
Dim lastX As Integer
Dim lastY As Integer

'Points holds the x,y locations you clicked on in the
picture window.
'Note, we'd like to declare Points as
'Dim Points(1 To 100, 1 To 2) As Double
'and pass to maple Points(1,1).  This will work ok
except that when
'NumPoints = 5, there will be 95 zero entries in the
points array, which we
'don't want used in the curve fit.  We also don't want
to grow the array by
'one point at a time because it's so inefficient.  So,
instead, we'll fake a
'2D array using C order indexing into a 1D array.
```

```
Dim Points(0 To 100) As Double
Dim NumPoints As Long

Private Sub Clear_Click()
   Picture1.Refresh
End Sub

Private Sub ClearButton_Click()
   NumPoints = 0
   lastX = -1
   ReDim CurveFittingModule.fit(0)
   Picture1.Refresh
End Sub

Private Sub FitButton_Click()
   'call Maple to fit the drawn curve using kv
   Screen.MousePointer = vbHourglass
   CurveFit kv, Points, NumPoints
   Screen.MousePointer = vbDefault
End Sub

Private Sub ExitButton_Click()
   'shutdown OpenMaple
   StopMaple kv
   End
End Sub

Private Sub DebugButton_Click()
   'show the debug form
   DebugForm.Visible = True
End Sub

Private Sub Form_Load()
   Dim error As String
   Dim args(0 To 1) As String

   'initiate callbacks
   cb.lpTextCallBack = GetProc(AddressOf TextCallBack)

   cb.lpErrorCallBack = GetProc(AddressOf
ErrorCallBack)
   cb.lpStatusCallBack = 0
   cb.lpReadLineCallBack = 0
   cb.lpRedirectCallBack = 0
   cb.lpQueryInterrupt = 0
   cb.lpCallBackCallBack = 0

   'start Maple. Note that cb is an argument to the
StartMaple command
   kv = StartMaple(0, args, cb, 0, error)
   If kv = 0 Then
       MsgBox "Error starting Maple: " + StrConv
(error, vbUnicode), vbCritical, ""
       Unload Me
       End
   End If

   lastX = -1

End Sub

Private Sub Form_Unload(Cancel As Integer)
   StopMaple kv
End Sub

Private Sub Picture1_MouseMove(Button As Integer,
Shift As Integer, x As Single, y As Single)
   'update the x/y location text boxes
   MainForm.XLocation.Text = format(x /
MainForm.Picture1.ScaleWidth, "#.###")
   MainForm.YLocation.Text = format
((MainForm.Picture1.ScaleHeight - y) /
MainForm.Picture1.ScaleHeight, "#.###")
End Sub

Private Sub Picture1_MouseUp(Button As Integer, Shift
As Integer, x As Single, y As Single)
   'record the new point
   Points(2 * NumPoints) = x /
MainForm.Picture1.ScaleWidth
   Points(2 * NumPoints + 1) =
(MainForm.Picture1.ScaleHeight - y) /
MainForm.Picture1.ScaleHeight
   DebugForm.DrawPointList.AddItem format(Points(2 *
NumPoints), "0.000") + "," + vbTab + format(Points(2 *
NumPoints + 1), "0.000")
   NumPoints = NumPoints + 1

   'draw the new point
   Picture1.FillStyle = vbFSSolid
   Picture1.Circle (x, y), 40, vbGreen
   If lastX >= 0 Then
       Picture1.Line (lastX, lastY)-(x, y), vbBlue
   End If
   lastX = x
   lastY = y
End Sub

Private Sub Picture1_Paint()
   'draw the curves
   DrawPoints Points, NumPoints
   DrawCurve
End Sub
```

CurveFittingModule

This module implements all the functionality to connect to Maple.

Source Code

```
Public cb As MapleCallBack
Public fit() As Double
Public showFunction As Boolean

'Output Callback
Public Sub TextCallBack(data As Long, ByVal tag As
Integer, ByVal output As Long)\

   Dim OutputString As String

   OutputString = MaplePointerToString(output)
   DebugForm.OutputText.Text =
DebugForm.OutputText.Text + vbCrLf + OutputString\

   If showFunction Then
       MainForm.FunctionBox.Text = OutputString
   End If
End Sub

Public Sub ErrorCallBack(data As Long, ByVal Offset As
Integer, ByVal output As Long)\

 MsgBox " at offset " + str(Offset) + " " +
MyPointerToString(output), vbInformation, ""\

End Sub
```

```
Public Function ConvertArrayToMaple(ByVal kv As Long,
ByRef a() As Double, ByVal size As Long)\

   Dim r As Long
   Dim rts As RTableSettings
   Dim bounds(0 To 3) As Long

   ' sizex2 Maple rtable
   bounds(0) = 1
   bounds(1) = size
   bounds(2) = 1
   bounds(3) = 2

   ' setup the array details
   RTableGetDefaults kv, rts

   rts.data_type = RTABLE_FLOAT64
   rts.num_dimensions = 2
   rts.foreign = 1
   rts.order = RTABLE_C

   ' create the vertex Array
   r = RTableCreateFloat64(kv, rts, a(0), bounds(0))
   Call MapleALGEB_Printf1(kv, "rtab = %a", r)

   ConvertArrayToMaple = r

End Function

' OpenMaple: Calls Maple to find a function that
closely
'   approximates the chosen points using the selected
method.
Public Sub CurveFit(ByVal kv As Long, ByRef a() As
Double, ByVal size As Long)
   Dim vertexArray, args, tmp, varX, varEQN, r,
FitFunction As Long

   If size <= 2 Then
       GoTo end_CurveFit
   End If

   ' convert the vertex array to a Maple Array
   vertexArray = ConvertArrayToMaple(kv, a, size)

   ' build up the arguments for calling the
```

```
appropriate CurveFitting function
   args = NewMapleExpressionSequence(kv, 2)


   'put the array into the expression sequence
   MapleExpseqAssign kv, args, 1, vertexArray


   'create a Maple variable named "x"
   varX = ToMapleName(kv, "x", True)
   Call MapleALGEB_Printf1(kv, "x = %a", varX)


   'put "x" in the expression sequence
   MapleExpseqAssign kv, args, 2, varX
   Call MapleALGEB_Printf1(kv, "args = %a", args)

   'get the name of the curve-fitting interpolation
method as a Maple object
   FitFunction = EvalMapleStatement(kv,
("CurveFitting:-" + MainForm.Method.Text + ";"))\

   If FitFunction = 0 Then
       GoTo end_CurveFit
   End If
   Call MapleALGEB_Printf1(kv, "fit = %a",
FitFunction)

   ' call the CurveFitting function
   r = EvalMapleProcedure(kv, FitFunction, args)

   ' r will be 0 if there was an error (eg.
BSplineCurve was
   ' used to fit too few points.  Don't draw anything
in this case.
   If r = 0 Or IsMapleNULL(kv, r) Then
       GoTo end_CurveFit
   End If

   ' assign the result to the Maple variable `eqn`,
and show the equation
   varEQN = ToMapleName(kv, "eqn", True)
   tmp = MapleAssign(kv, varEQN, r)
   showFunction = True
   Call EvalMapleStatement(kv, "eqn;")
   showFunction = False
   ' Use Maple's plot command to generate a list of points to plot.

  ' The guts of the following line is
  '  op([1,1],plot(r,x=0..1));
  'show the value in the debug window
   Call EvalMapleStatement(kv, "op([1,1],plot(eqn,x=0..1,numpoints=10));")
   r = EvalMapleStatement(kv, "Array(op([1,1],plot(eqn,x=0..1)),datatype=float[8],order=Fortran_order);")\


   'Ensures the result of the above command is an rtable (Array)
   If r <> 0 And IsMapleRTable(kv, r) Then
       ' extract the data pointer from the array
       fit = DoubleRTableDataBlock(kv, r)
       MainForm.Picture1.Refresh
       DrawCurve
   End If

end_CurveFit:

End Sub

'draw the fitted curve
Public Sub DrawCurve()
   Dim lx, ly, x, y As Double
   Dim i As Long

   ' do nothing if fit not defined
   On Error GoTo end_drawCurve
   MainForm.Picture1.FillStyle = vbFSTransparent
   lx = fit(LBound(fit), 1) * MainForm.Picture1.ScaleWidth
   ly = MainForm.Picture1.ScaleHeight - fit(LBound(fit), 2) * MainForm.Picture1.ScaleHeight\

   DebugForm.FitList.Clear
   DebugForm.FitList.AddItem format(fit(LBound(fit), 1), "0.000") + "," + vbTab + format(fit(LBound(fit), 2), "0.000")\

   For i = LBound(fit) + 1 To UBound(fit)
```

```
       x = fit(i - 1, 1) *
MainForm.Picture1.ScaleWidth
       y = MainForm.Picture1.ScaleHeight - fit(i - 1,
2) * MainForm.Picture1.ScaleHeight\

       DebugForm.FitList.AddItem format(fit(i, 1),
"0.000") + "," + vbTab + format(fit(i, 2), "0.000")\

       MainForm.Picture1.Line (lx, ly)-(x, y), vbRed
       MainForm.Picture1.Circle (x, y), 40, vbOrange
       lx = x
       ly = y
   Next i
   MainForm.Picture1.Circle (x, y), 40, vbOrange

end_drawCurve:
End Sub

'draw the points as clicked
Public Sub DrawPoints(Points() As Double, size As
Long)
   Dim lx, ly, x, y As Double
   Dim i As Long

   If size > 0 Then
       MainForm.Picture1.FillStyle = vbFSSolid
       lx = Points(0) * MainForm.Picture1.ScaleWidth
       ly = MainForm.Picture1.ScaleHeight - Points(1)
* MainForm.Picture1.ScaleHeight\

       MainForm.Picture1.Circle (lx, ly), 40, vbGreen
       For i = 1 To size - 1
           x = Points(2 * i) *
MainForm.Picture1.ScaleWidth
           y = MainForm.Picture1.ScaleHeight - Points
(2 * i + 1) * MainForm.Picture1.ScaleHeight\

           MainForm.Picture1.Line (lx, ly)-(x, y),
vbBlue
           MainForm.Picture1.Circle (x, y), 40,
vbGreen
           lx = x
           ly = y
       Next i
   End If
End Sub
```

Java API

Simple Example: integrate an expression

This simple example starts a Maple session, which is encapsulated with the **Engine** class. The Java program executes the Maple expression int(x,x) and writes the result as standard output.

```
/* import the Java Open Maple classes */
import com.maplesoft.openmaple.*;


/* import the MapleException class */
import com.maplesoft.externalcall.MapleException;
class test
{
   /* declare the main function */
   public static void main( String args[] )
   {
       String a[];
       Engine t;
       int i;


       /* build the command line arguments for Maple
*/
       a = new String[1];
       a[0] = "java";
       try
       {
           /* start the Maple session.  Use the
Default EngineCallBacks.
            * the text Output will be directed to
System.out */
           t = new Engine( a, new
EngineCallBacksDefault(), null, null );


           /* evaluate a Maple expression, in this
case int(x, x) */
           t.evaluate( "int( x,x );" );
       }
       catch ( MapleException e )
       {
           /* if a Maple error occurs, exit */
           System.out.println( "An exception
occured\n" );
           return;
       }


       /* on success print a message */
       System.out.println( "Done\n" );
   }
}
```

Intermediate Example: Java emulator for command-line Maple

This Java program emulates a command-line version of Maple. It shows how to evaluate Maple expressions, access Maple help, and interrupt Maple computations within a Java program.

Source Code

```
/*
* a Java emulator of command-line Maple.  This demonstrates many of the
* features of Java OpenMaple.
*/

/* import the OpenMaple classes */
import com.maplesoft.openmaple.*;

/* import the MapleException class */
import com.maplesoft.externalcall.MapleException;

import java.io.*;

class jcmaple
{
   /* the Maple session object */
   static private Engine kernel;


 /* the EngineCallBacks object*/
   static private JCEngineCallBacks callbacks;


 /* the MHelpBackVector object*/
   static private JCHelpCallBack helpcallbacks;
```

```
  /* where to read input from */
   static private BufferedReader input;

   /* initialize input, the input source */
   private static void initInput( String file )
   {
        if ( file == null )
        {
            input = new BufferedReader( new
InputStreamReader( System.in ) );
        }
        else
        {
            try
            {
                input = new BufferedReader( new
FileReader( file ) );
            }
            catch ( FileNotFoundException e )
            {
                System.err.println( e.getMessage() );
                System.exit(1);
            }
        }
   }

   /* initialize the Maple kernel and callback objects
*/
   private static void initKernel( )
   {
        String args[];

        args = new String[1];
        args[0] = "java";

        /* create the callback objects */
        callbacks = new JCEngineCallBacks();
        helpcallbacks = new JCHelpCallBack();

        try
        {
            /* start the Maple kernel by creating an
instance of the Engine class */
            kernel = new Engine( args, callbacks,
null, null );
        }
        catch ( MapleException e )
        {
            System.err.println( e.getMessage()+"\n" );

            System.err.println( "Error starting
OpenMaple session\n" );
            System.exit(1);
        }
    }

        /* the main method of the class jcclass */
    public static void main( String args[] )
    {
        Algebraic ret;
        String expr;
        boolean done;

        /* initialize the input method */
        if ( args.length > 0 )
        {
            initInput( args[0] );
        }
        else
        {
            initInput( null );
        }

        /* start the Maple kernel */
        initKernel( );

        expr = null;
        ret = null;

        System.out.println( "\nWelcome to
jcmaple\n" );

        done = false;
        while ( !done )
        {
            /* print a Maple prompt */
            System.out.print( "> " );

            /* get the input */
            try
            {
                expr = input.readLine();
            }
            catch ( IOException ioE )
            {
                System.err.println( ioE.getMessage
() );
                System.exit(1);
            }

            /* check for some special conditions */
            /* end of file */
            if ( expr == null )
            {
                done = true;
                continue;
            }


           /* check for empty expression */
            if ( expr.equals( "" ) )
            {
                continue;
            }

            /* check for help request */
            if ( expr.charAt( 0 ) == '?' )
            {
                /* look up the help page and display
it */
                String topic = expr.substring( 1 );

                helpcallbacks.initHelp();
                try
                {
                    kernel.getHelp( topic,
kernel.MAPLE_HELP_ALL,
                            helpcallbacks, 78, null );

                }
                catch ( MapleException me )
                {
                    System.out.println( me );
                    continue;
                }

                System.out.println( "Help page for
"+topic );
```

```
                helpcallbacks.display();
                System.out.println( "\n" );

                continue;
            }

            /* evaluate the Maple expression the user
enters */
            try
            {
                ret = kernel.evaluate( expr );

                if ( ret != null )
                {
                    /* check if we should terminate */

                    done = ret.isStop();


        /* since we no longer need this object, dispose it */
                    ret.dispose();
                }
            }
            catch ( MapleException e )
            {
                System.err.println( "Error evaluating
Maple expression\n" );
                System.exit(1);
            }

            /* print any generated input */
            while ( callbacks.numOfLines() > 0 )
            {
                System.out.println( callbacks.getLine
() );
            }
        }

        try
        {
            kernel.stop();
        }
        catch ( MapleException me )
        {
        }
    }
```

```
}
```

Sample Output

```
Welcome to jcmaple

> A := int (z*x^2, x);

                              A := 1/3*z*x^3

> eval (A, {z=2, x=1/2});

                                   1/12

> ?eval
Help page for eval
eval - evaluation

Calling Sequence
    eval(e, x = a)

    eval(e, eqns)

    eval(e)

    eval(x, n)


Parameters
    e    - expression

    x    - usually a name but may be a general
expression

    a    - expression

    eqns - list or set of equations

    n    - positive integer


Description
- The most common use of the eval function is to
evaluate an expression e at a
 given point x=a. For example, eval(x^2+3*x+2,x=1);
```

evaluates the polynomial x^2+3*x+2 at the point x=1 obtaining 6.

- In the second calling sequence, where eqns is a set or list of equations, the expression e is evaluated at all the given points (or expressions) simultaneously.

- Usually, the left-hand sides of the equations will be variable names being evaluated. However, if the left-hand side x of an equation is not a simple name, a syntactic substitution is attempted which has the same limitations as with subs.

- Since eval does pointwise evaluation, eval cannot be used to evaluate an expression at a singularity. Use limit instead.

- The eval command knows how to correctly evaluate derivatives, integrals, piecewise function definitions, etc. whereas the subs command in comparison makes substitutions which may not make sense mathematically. If there are symbolic dependencies between the expression and the point at which it should be evaluated that are considered ``unsafe'', eval will return unevaluated. This knowledge is coded in the Maple library in the Maple procedures `eval/diff`, `eval/int`, `eval/piecewise`, etc. The user may teach Maple how to evaluate a function as follows. Given a function foo(y), the call eval(foo(y),x=a) will result in `eval/foo`(foo(y),x=a); being called if the user has defined `eval/foo` to be a Maple procedure.

- The two remaining calling sequences of the eval function are used to perform explicit evaluation of a variable or expression. The meaning of evaluation in these cases is different from the previous calling sequences. The default evaluation rules in Maple are full evaluation for global variables, and one-level evaluation for local variables and parameters. Sometimes the user requires full evaluation or one-level evaluation explicitly. For example, if x := y and y := 1 in a Maple session, what is the value of x; ? In an interactive session where x and y are global variables, x would evaluate to 1 and we would say that x is ``fully evaluated''. For one-level evaluation, we would use the command eval(x, 1) which would in this case yield y. However, inside a Maple procedure, if x is a local variable or a parameter, then x evaluates to y and we would say x evaluated ``one level''. For local variables and parameters, full evaluation is obtained with eval(x), yielding 1 in this example.

- The call eval(e) means the expression e is fully evaluated up to two delayed evaluations (see uneval). This means that each name in e will be evaluated to its value, and the value will be evaluated recursively, and each function call in e will be executed with its parameters evaluated, but not beyond two levels of delayed evaluations.

- The call eval(x, 1) means evaluate the name x one level. That is, the value assigned to the name x is returned without further evaluation. More generally, eval(x, n) yields n-levels of evaluation of names.

- Evaluation to n levels (n>1) is most useful for expressions that are still just names after n-1 levels of evaluation. The eval(x, n) call returns the result of evaluating the name found at level n-1 to one level.

- While evaluation to n levels can be applied directly to expressions other than names, the outcome can vary greatly depending on how the actual structure of the expression interacts with various automatic simplifications.

- Applying eval to mutable container objects like tables, rtables, and modules does not map over the elements. Use map(eval,M) to evaluate elements in an object, M.

Examples

```
# Examples of using eval to evaluate at a 'point':

> poly := x^3+3*x+2;

                                     3
                            poly := x  + 3 x + 2

> eval(poly, x=1);

                                     6

> poly := x*y+3*z+2;

                           poly := x y + 3 z + 2

> eval(poly,[x=2,y=3,z=t]);

                                  8 + 3 t

> expr := sin(x)/cos(x);

                                       sin(x)
                               expr := ------
                                       cos(x)

> subs(x=0,expr);

                                   sin(0)
                                   ------
                                   cos(0)

> eval(expr,x=0);

                                     0

# Example where eval will return a (correct) error
instead of an incorrect
# answer:

> expr := sin(x)/x;

                                       sin(x)
                               expr := ------
                                         x

> eval(expr,x=0);

Error, numeric exception: division by zero

> limit(expr,x=0);

                                     1

> eval(int(sin(x),x),x=1);

                                  -cos(1)

> integral := int(f(t,a),t=a..x);

                                       x
                                      /
                                     |
                        integral :=  |   f(t, a) dt
                                     |
                                    /
                                      a

> eval(integral,{t=0,a=1});

                                 x
                                /
                               |
                               |   f(t, 1) dt
                               |
                              /
                                1

> subs({t=0,a=1},integral);

                            /
                           |
                           |  f(0, 1) d0 = 1 .. x
                           |
                          /

> der := diff(f(x),x) + f(x);

                                 /d       \
                          der := |-- f(x)| + f(x)
                                 \dx     /

> eval(der,x=0);

                           /d      \|
                           |-- f(x)||      + f(0)
                           \dx     /|x = 0

> subs(x=0,der);

                            diff(f(0), 0) + f(0)

# Example of using eval to evaluate fully or evaluates
names a number of
# levels:

> a := b;

                                   a := b

> b := c;

                                   b := c

> c := x+1;

                                 c := x + 1

> a; # default full recursive evaluation

                                   x + 1
```

```
> eval(a); # force full recursive evaluation

                                    x + 1

> eval(a,1);

                                      b

> eval(a,2);

                                      c

> eval(a,3);

                                    x + 1

> eval(a,4);

                                    x + 1


See Also
Eval, subs, limit, evalf, evalm, evaln, evalhf, evalb,
evala, evalc, evalr,
         value, rtable_eval, ExtendingMaple



> quit
bytes used=84992, alloc=80896, time=0.016
```