AC 2010-2338: ON MAPLET DEVELOPMENT AND PROGRAMMING TUTORIAL FOR SCIENCE AND ENGINEERING STUDENTS

Aarti Narayanan, Magnificat High School

Ganapathy Narayanan, The University of Toledo

On Maplet Development and Programming Tutorial for Science and Engineering Students

Abstract

The information contained in this paper is of an introductory nature a tutorial on the 'Maplet Development and Programming' for selected science and engineering students. The knowledge gained by the authors is documented here to help students who are interested in pursuing science and/or engineering for their degree, and who have a keen interest in a hands-on experience for solving selected science/engineering problems using MAPLE. As a part of this tutorial, technical how-to details are given to help the student develop simple maplets with simple programming constructs that are needed to solve these selected problems. The process of developing Maplets demonstrated within this tutorial will provide a hands-on experience using MAPLE. The authors believe that this paper will engage engineering and science students to create or modify existing Maplets for their other college course activities. The tutorial assumes a familiarity with basic programming constructs, and would be appropriate after an introductory programming course, one year of the calculus and one year of any physical science.

Introduction

The content of this paper serves as an introduction to 'Maplet Development and Programming'. The hands-on Maplet examples given in this paper may serve as the first GUI (Graphical User Interface) programming experience for the high school or college student, and are directed towards students pursuing a first science and/or an engineering degree. A minimum of the calculus, science, the use of computers, and the use of MAPLE is expected to appreciate the application details given in this paper to develop simple Maplets with its programming constructs. Any student can easily create these Maplet examples given in this paper with minimum effort.

The body of the paper starts with the motivation discussion section with an explanation for a need to teach this introductory MAPLE programming language and the creation of Maplets for high school/college science and engineering students to help solve science or engineering problems encountered in their coursework. The reminder of the paper deals with the actual demonstration of the Maplet tutorial starting with a discussion on 'What is a Maplet?', then proceeds with the detailed discussion on MAPLE programming language constructs and the two methods to create Maplets in MAPLE. In short, one way to create Maplets is by using MAPLE programming constructs, and the other method is to use 'Maplet Builder' assistant that MAPLE provides to create MAPLE program constructs for building the corresponding Maplets interactively.

Motivation for a need to teach Maplets

The MAPLE software provides an opportunity for students to develop a GUI interface to a powerful math solving program. While many engineering curricula include programming languages, the MAPLE software provides an easier entry into the problem solving possibilities with Maplet programming. Let us see why MAPLE provides an easier entry for students.

It is a well known fact that most high schools and colleges help students to learn the basic symbolic analysis (Algebra, Trigonometry, Geometry and Calculus) and numeric procedures (Plot of functions, Solution of Algebraic Equations, Series, Finding Roots of Equations, and so on). These learning subject materials form the basis to help student to solve science and engineering problems of today. However, just knowing the subject material to solve a particular science or engineering problem is not enough for a modern student of today. The ability to automate the solution process or the student ability to produce the answer 'quickly' is becoming a norm in the scientific community and in the workplace. The invention of computers and its successive developments have helped tremendously in terms of automation of performing numerical, symbolic and other mundane calculations.

In the age of working with and using computers for solving everyday problems, it is imperative that the young 'motivated' students are taught the 'computer languages' to develop the computer programs. There are a number of computer languages that have been developed for use in automating sequence of 'computations'. Some older computer languages such as FORTRAN & BASIC (developed in 1950's) have given understanding to the development of modern object-oriented languages such as Visual Basic, C++ and JAVA. Of course, these are not the only known exhaustive set of computer languages. There are many other computer languages of special purpose significance and are used in select scientific environment.

In particular, the students today are taught some computer language programming constructs and the associated program creation to help solve science and engineering problems, with some built-in graphical user interface convenience. In fact, many schools and college curricula have this mind. Most high schools teach either MS Quick Basic or some advanced scientific and graphic calculator like TI-83 or TI-89. Many colleges teach C or Visual Basic or C++ or JAVA as part of programming constructs learning for automation or graphical user interface creation. In all these languages, the general programming constructs are the same with subtle differences, namely consider programming constructs for decision making (if-then-else), for loops (do-loops or for-next loops), for limits within (while or until), and for break/or pause or stop statements. These Visual Basic or C++ or JAVA based developed programs are just sequence of computational steps for the computer with the 'knowledge base' that must be provided by the

programmer (student) to solve only that particular science or engineering problem. Of course, the program will depend on a specific knowledge base that the program is designed for.

The MAPLE computer program is a powerful symbolic and numerical software developed by professional programmers with 'built-in math knowledge' of Algebra, Trigonometry, Calculus, Functions, Differential Equations, Plot capability, and in addition, the well-known computer programming constructs of other above languages. Thus, a program constructed using MAPLE can use the Math knowledge base in addition to providing the automation steps to solve for a powerful science or engineering problem. In addition, the authors found that MAPLE programming construct is far superior and 'easy to learn' in terms of creating automated programs with graphical user interfaces. Please note that 'easy to learn' term is used when one compares the program constructs 'easiness' of other languages including Visual Basic, C++ or JAVA for developing GUI programs with that of the facility provided by MAPLE.

A second view point is that high school or undergraduate students do spend a semester or more in learning programming constructs of any language, either directly as in an introductory programming course or in conjunction with another introductory course learning to solve problems using programming constructs on computers. Today, these courses are in the curricula of major engineering or science programs in many well known schools, in some format or another. The authors believe that instead of Visual Basic or C++ or JAVA language programming constructs, each school should consider such a programming construct tool that is not only useful to write the regular textual programs on computers, but also the tool chosen should help the students to create sophisticated, powerful, math-oriented, general purpose, web-convertible programs, and be able to create a Graphical User Interface (GUI) for these programs. Many such tools exist today for such teaching purpose, and one such tool is the MAPLE Software. The second author contends that the programming tool must also include more sophisticated and symbolic/numerical oriented mathematical analysis capability that a student of science and engineering would use, and that the MAPLE software tool provides this ability.

In fact, MAPLE is a powerful and sophisticated symbolic tool with advanced numerical facilities that are not available in the Basic or C or C++ or Java languages. MAPLE has all the usual (if-the-else, do-loops, while-loops, break-pause-stop) programming constructs that these other languages provide. But the power to easily create the GUI programs in MAPLE along with the background math (knowledge base) kernel is far superior in comparison to the other programming languages.

One specific example of the easiness of GUI program development in MAPLE is given here. GUI programs are event-based programs that require both the placement of graphical objects that the user sees on screen and the development of the necessary event action routines. With MAPLE software, these GUI programs are only a few lines (long) for simple programs, and

MAPLE takes care of most of the graphical object creations with one line per object. Also, the associated event actions are also very short. Such GUI programs are either impossible (like the case of programs that can perform 100 digit computations in a symbolic manner) or very long (100 lines or more for even simple GUI programs) in Visual Basic or C++ or Java, with specific object type specification being important.

The reasons cited above lead the authors of this paper to provide a hands-on basic tutorial as a way of demonstrating the ease and power of creating GUI programs with MAPLE programming constructs. The paper assumes knowledge of the usual (if-then-else, do-loops, while-loops) program constructs in MAPLE, but the paper focuses its discussion on the powerful GUI program creation within MAPLE as a hands-on tutorial. The usual program constructs are well discussed within the MAPLE software under help or an introductory MAPLE programming book[1] can be read.

A word of caution is in order for the effective learning of the Maplets creations and the associated MAPLE program constructs discussed in this paper. Since the paper is written with examples with minimal information on MAPLE program constructs[2], it is essential for the student, or the reader of this paper, to type the GUI programs in this paper on a computer with MAPLE installed to fully appreciate the easiness of MAPLE GUI programs creation, and to become fully acquainted with the associated 'Maplets program constructs'. The Maplets elements table provided in this paper for such Maplet constructs may be used as a reference guide for creating other Maplet programs. In addition, if more help on performing calculus within MAPLE is needed for using MAPLE statements within Maplets or its modules, Ref 3 may give a helping hand in learning MAPLE for its general use.

What is a Maplet?

A Maplet is a Graphical User Interface (GUI) application for and written in the MAPLE software and a Maplet can be launched from a MAPLE session. In short, a Maplet allows a MAPLE software user to combine MAPLE math libraries and user procedures with 'Interactive Windows and Dialogs'.

To understand Maplets and their creation in the MAPLE software, one needs to understand the creation of a 'Graphical User Interface' (GUI) application in general. A GUI application is a combination of 'Graphical User Interface Elements' and of a set of 'Procedures' needed to carry out 'Actions' associated with the 'User' requests through the 'Clicks' of 'GUI Elements'. In this paper, the discussion will be limited to the specific Maplet 'GUI' elements and to the simple 'Action' procedures needed for the Maplets created for the paper. No attempt has been made in this paper to be exhaustive in order to explain all the programming options to write complex and advanced procedures or advanced Maplets in MAPLE. However, this paper does explain the

major concept details of the 'Maplet Application Development(MAD)' in MAPLE, and simple Maplet examples are given to give hands-on experience in learning and creating Maplets in MAPLE.

There are two approaches to creating Maplets within the MAPLE software: (1) First is the 'Maplet Creation using Command lines'. This approach is a bit more involved than the second interactive Maplets build approach to create Maplets. (2) The second is the 'Maplet Creation using the 'Maplet Builder Assistant'; the MAPLE Command approach (the first approach) is far more powerful and more flexible than using the Maplet Builder Assistant. Hence, the first command lines approach is discussed before the second approach. However, either approach can be used for Maplet creation.

Maplet Creation using Command lines

A Maplet command construct is easy enough to 'understand' by looking at a very simple 'Hello World!' Maplet (this is the usual first program one created when learning C or C++ or Java language, hence this paper follws the tradition. Figure 1 shows the one line (second line) MAPLE statement to construct this Maplet. The first line in Figure 1 is always needed in any Maplet to load the 'Maplet Elements' before using the Maplets elements on the following command lines of the Maplet statements. There are several things to take notice of in this Maplet construct. All other Maplets have a very similar construct except for the list element '["Hello World!"]', and each Maplet has a different particular Maple List instead of the list element '["Hello World!"]'.


```
With(Maplets[Elements]);
Maplets[Display] (Maplet( ["Hello World!"] ) );
```

Figure 1: Hello World Maplet Example 1

Parts of a Maplet Application Program Construct

There are two parts to a Maplet Application Program statement construct, as seen in the above 'Hello World' example:

(1) First is the Maplet creation in the MAPLE software by using the 'Maplet' command with a list of Maplet elements within it as an input argument (see second line in Figure 1). In the 'Hello World' Maplet of Figure 1, we used only one argument (Maplet 'LABEL' Element) within the 'Maplet(…)' argument of the 'Maplet Command', and this 'LABEL' element needs some text as the list of 'ONE' Maplet element. Every (and all) list of Maplet elements is represented within

square brackets ("[ ]") as shown in the 'Hello World' example. One could use as many elements within the Maplet input list as is needed by the application program construct. The other application examples given below in this paper show the Maplet input list containing many other Maplets elements.

(2) Second item to notice of a Maplet construct is the 'Maplets[Display]' action verb needed to request that the MAPLE kernel to display this particular Maplet. This action verb 'Maplets[Display]' is ALWAYS needed in the MAPLE software environment to display any of Maplets created using the MAPLE software. The input argument to this Maplet action verb '[Display]' is the particular Maplet that needs to be displayed on the screen for the user. Please note the letter 's' on 'Maplets' at the end of 'Maplet' word while using the action verb 'Display' procedure of 'Maplets' library. Of course, one could define the Maplet with a variable name and use this name as input to the action verb 'Maplets[Display]'. The result of such a Maplet screen display of the GUI window created after execution of the Maplet command within MAPLE is shown on the left side of Figure 1. That is it in terms of the concept of the Maplets creation, and one needs to understand the various Maplet elements (arguments) for other uses.

Maplets Elements

The first line of the 'Hello World' Maplet example (see Figure 1) is mandatory to load the Maplets library of 'Elements' before any Maplet can be constructed by the MAPLE kernel with the use of Maplet statement. The description and the use of other Maplet elements of this library are discussed through the hands-on development of several Maplets Examples. A list of select and important Maplet elements is defined under Table 1.

Before the other Maplet elements are discussed in the Maplets application examples, it is important to understand how a 'LIST' can be constructed in MAPLE. The MAPLE software uses one 'List' or 'Nested List' as the input argument to any 'Maplet(…)' command for its creation. A LIST in MAPLE is an ordered sequence of comma-delimited 'MAPLE expressions or Maplet Elements' that is enclosed in square brackets '[ ]'. For example, '[expr1, expr2, expr3]' is a list of three expressions 'expr1, expr2 & expr3'. Of course, one could have any numbers or text or any appropriate Maplet elements in any of these expressions, and the list may consist of any number of expressions within the '[ ]' square brackets. A 'Nested List' is also an ordered sequence of expressions enclosed in '[ ]' square brackets in which any one or more of these expressions is itself a list. For example, '[expr1, [expr2, expr3], expr4]' is a nested list with the list '[expr2,expr3]' inside the outer list.

Within any Maplet application programs, the text string used as a Maplet 'Label' element, 'TextBox' element used for 'User Prompts', 'TextField' element for user input of fields and other Maplet elements are defined as the expressions in the 'Maplet List' constructed as an input

argument to the Maplet command. The main Maplet Window GUI definition includes the main list (the outermost list) of the nested list used as input argument to the Maplet Command. The other lists within the main list are an ordered sequence of Maplet elements needed as part the Maplet Application Design. Many of these selected Maplet elements are defined in Table 1.

Table 1: Twenty important sample Maplets Elements

Serial Number	Maplet Element Name	myMapletList Expression
#1	"OK" Button	Button("OK", 'action verb')
#2	"Clear" Button	Button("Clear", 'action verb')
#3	"Cancel" Button	Button("Cancel", 'action verb')
#4	"Text" Label	"Enter some Text:"
#5	"Text Field" Input	TextField['TextFieldName'] ( )
#6	"Shutdown" Action Verb	Shutdown( )
#7	Set Option	SetOption('TextField' = " ")
#8	Window	Window( )
#9	"Text Input" Box	TextBox['TextFieldName'] ( )
#10	"Plotter"	Plotter['PlotHandleName'] ( )
#11	"Evaluate" myProc	Evaluate('TextOutFieldName' = "myProc")
#12	"Evaluate" int(f(x),x)	Evaluate('TextOutFieldName' = 'int(TFN1,TFN2)')
#13	"Evaluate" plot( )	Evaluate('PlotHandleName' ='plot([TF1,TB1], TF2=0..SL1)
#14	"CheckBox"	CheckBox['CBname'] ('value' = 'true')
#15	"ComboBox"	ComboBox['CoBname']('value'="red", ["blue", "green", "yellow"])
#16	"Label"	Label("Text")
#17	"ListBox"	ListBox['LBname']('value'="Lvalue1", ["Lvalue2","Lname3"])
#18	"MathMLViewer"	MathMLViewer['MMLVname']('value'=x^2 – 4*x +5)
#19	"RadioButton" (set of two radio buttons)	[RadioButton['RBname1']("first", true, 'group'="'BG1), RadioButton['RBname2']("second",'group'='BG1')]
#20	"Slider"	Slider['SLname'](10, 0..20, 'majorticks'=10, 'minorticks'=2, 'showticks')

Putting together one or more of these Maplet elements in Table 1 as an appropriate list within the 'Maplet command' can produce many of the simpler Maplet Applications, as shown below. Several of these simpler Maplet applications are developed below using the Maplets command lines approach before showing how to use the Maplets Builder Assistant.

Maplets Application Examples

Of course, only select application examples using powerful MAPLE Calculus commands and using powerful MAPLE Student Tutors are given in this paper. Similar Maplets with other MAPLE commands and other MAPLE tutors can be easily constructed by the student following the given hands-on Maplet Application Examples.

Maplet Example 2-A: **'Hello World' Example**

"Hello World" with one Button Element is developed using Maplet command line approach consisting of 'Button' element along with the 'Label' or 'Text' Element, as shown in Figure 2-A


```
With(Maplets[Elements]);
Maplets[Display] (Maplet( [
["Hello World", Button("OK", Shutdown( ) ) ]
        ] ) );
```

Figure 2-A: Hello World Maplet with one Button

Maplet Example 2-B: 'myMapletList' Maplet

The above Maplet Example 2-A can be rewritten in a more generic form of 'myMapletList' variable as input argument to the Maplet command (as shown below in Figure 2-B), where 'myMapletList' expression for this example is the same as the Maplet List consisting of two Maplet elements, the "Text" element with the "Button" element: '["Hello World", Button("OK", Shutdown( ))]'.


```
With(Maplets[Elements]);
myMapletList:= ["Hello World", Button("OK", Shutdown( ) ) ]:
Maplets[Display] (Maplet( [ myMapletList] ) );
```

Figure 2-B: 'myMapleList' Maplet

The Maplet construct format in this example 2-B is a showcase of the 'concept' in the development of any other Maplet application, The other maplets can be created by developing 'myMapletList' appropriate for the Maplet application that the user/student is interested in. However, it is not necessary to create the variable 'myMapletList', and the application developer (student) can type in the Maplet Elements list explicitly, as in Example 2-A. Many of the Maplets applications shown below use the Maplet Elements list directly as the Maplet command input argument without the use of explicit variable 'myMapletList' definition.

Maplet Example 2-C: 'myMaplet' Maplet with Button displayed below text

One variation of the maplet window is to make the 'OK' Button to be shown below the "Hello World" text. It is to be noted that the Maplet command parses for the elements in the nested list, and each list of elements in the outer (Main) list uses the display ROW position in that order. In Figure 2-B, we see the text element "Hello World" and the Button element are in the main list as two elements of the single (one) list, and hence, "Hello World" text and Button 'OK' occupy the same display row of the Maplet Window.

If we wished for the "Hello World" text in display row 1 and the Button 'OK' below in display row 2, then we should have a nested list with two lists inside the main (outer) list, with the first inside list containing the "Hello World" text and the second inside list containing the button, as shown in Figure 2-C. Thus, each inside list of the main list forms a different display row inside the Maplet window.

Also, the Maplet construct in Figure 2-C is rearranged to reflect the generality of expressing the Maplet window with explicit 'Window' element that can assign the application user defined Maplet 'title', and the 'myMaplet' defined containing the 'MyMapletList' elements. In such a case, the Maplets 'Display' procedure displays 'myMaplet' program application supplied as an input argument to it.


```
restart;
with(Maplets[Elements]) :
myMapletList := ["Hello World"], [Button("OK",
    Shutdown( ) ) ] :
myMaplet := Maplet(Window('title'= "myMaplet",
    [eval(myMapletList)])) :
Maplets[Display](myMaplet);
```

Figure 2-C: 'myMaple' Maplet with Button 'OK' below text

How to construct more general and useful Maplet?

Now, with the above Maplet command construct and twenty important Maplet Elements given under Table 1, we can construct various 'powerful' Maplets that are very useful in mathematics, engineering and sciences. Of course, the power of the Maplets in mathematics, science and engineering is due the power of Maple kernel library and associated commands in mathematics and its use in science or engineering problems. So, the application (student) developer can combine one or many Maplet elements of Table 1 with one or many of its occurrences in the Maplets list, arranged sequentially in a nested list as 'myMapleList', and use Figure 2-B or 2-C like Maplet Command lines construct to create any sort of useful and general purpose mathematical or or science or engineering Maplet.

It is worth noting that any Maple procedure or function can be used within the 'Evaluate' Maplet element to make the user constructed Maplet as powerful as the MAPLE software permits the user, with very minimal limitations. All MAPLE built-in and user defined MAPLE procedures and functions, using the usual, if-then-else or do-loops or while-loops, program constructs, are allowed to be used within Maplet command, as needed user triggered action control that perform certain work within the Maplet GUI program. Due to this reason, a MAPLE user can construct very useful and powerful Maplet programs, in mathematics, engineering or sciences, even with few powerful MAPLE commands like 'plot', 'dsolve', 'int', 'diff' and so on. It is shown below how some of these useful and powerful Maplets are constructed with a combination of Maplet elements from Table 1, as part of 'myMapletList' in the Maplet Command construct. For these examples, 'myMapletList' variable is not defined at all, but the Maplet list is defined explicitly as input argument to the Maplet command. The Maplet construct of Figure2-C for defining a 'Maplet first with a window title' is used in these examples, and the Maplets Display verb uses the Maplet variable name to display it.

Maplet Example 3: 'Integrate' Maplet

There are many powerful MAPLE commands for use in Calculus, and here in this example, the integrate 'int' command is used to develop an 'Integrate' Maplet. Other similar Maplets for other powerful Calculus commands including 'diff', 'AproximateInt', 'dsolve' can be developed following this Maplet example. For any such Maplet (application) development, it is best to design the placements of all needed graphical Maplet elements first, and then construct the Maplet elements List using Table 1. For the 'Integrate' Maplet, the Maplet elements to be used are 'Labels', two 'TextField'elements, three 'Button' elements, and one 'TextBox' element needed for resulting output of the integrated expression. Figure 3-A shows the 'Integrate' Maplet (Application) command lines. Figure 3-B displays the on-screen window of the 'Integrate Maplet' with a sample User function entered for integration and its result as seen in the output TextBox.

```
restart;
with(Maplets[Elements]):
myIntMaplet :=  Maplet(Window('title'="MyIntMaplet", [
["Integrand: ", TextField['TF1']() ],
            ["Variable of integration: ", TextField['TF2'](3)],
            TextBox['TB1']('editable' = 'false', 3..40),
            [Button("Integrate", Evaluate('TB1' = 'int(TF1,TF2)')),
             Button("OK", Shutdown(['TF1', 'TF2', 'TB1'])),
             Button("Clear", SetOption('TF1' = "")) ] ] ) ):
Maplets[Display](myIntMaplet);
```

Figure 3-A: 'Integrate' Maplet Application

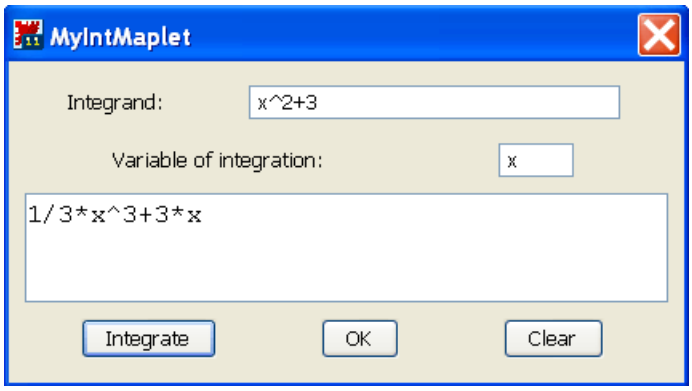

Figure 3-B: 'Integrate' Maplet (on-screen display)

Maplet Example 4: 'Plot' Maplet

Next hands-on example is a Maplet 'plot' application that will allow the user to graphically visualize the 2-D plot of ANY mathematically consistent continuous (one-to-one transformation) function y=f(x). The user has the desire to have a slider input for the x-variable values say between -20 to 20 (this can be changed for other 'Plot' Maplets). Thus, this Maplet (GUI) Application requires many Maplet elements, including (1) one or many user "TextField" Input Box for input of function, and/or variable limits, (2) one plot output box using the 'Plotter' element, (3) one or more Buttons like 'PLOT', 'Cancel', 'Close' etc. The Maplet Application design consists of arranging these elements on various rows (inner Lists) and columns (elements of inner list) of the window, and thus, the 'myMapletList' construct, to be used in the Maplet Command construct, is as shown in Figure 4-A. Figure 4-B displays an on-screen window with the plot output result of 'Plot Maplet' constructed for the 'Plot' application. With a bit of work, 'Plot-3D' Maplet can be developed by the student.

```
restart;
with(Maplets[Elements]):
myPlotMaplet := Maplet( Window('title'="myPlotMaplet",[
            ["Enter Function to Plot", TextField['TF1']() ],
            ["Function of Variable, x or t  ", TextField['TF2'](3)],
            Plotter['PL1'](),
            Slider['SL1']( 0..20, 5, 'showticks', 'majorticks'=5, 'minorticks' = 1, 'visible' = 'true'),
            [Button['B1']("Plot", Evaluate('PL1' = 'plot([TF1], TF2=-SL1..SL1)') ),
             Button("OK", Shutdown(['TF1', 'TF2'])),
             Button("Clear", SetOption('TF1'= "")) ] ] ) ):
Maplets[Display]( myPlotMaplet );
```

Figure 4-A: 'Plot' Maplet Application

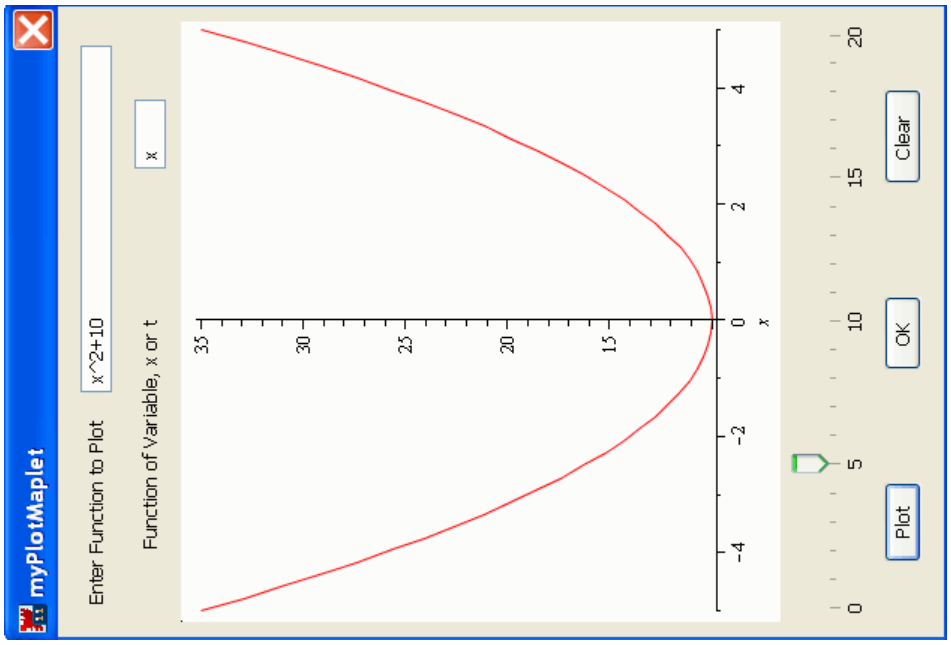

Figure 4-B: 'Plot' Maplet (On-Screen Display)

Pre-calculus Geometry Example 5: 'Triangle' Third Side Length Compute Maplet

A pre-calculus geometry example 5 is the 'Triangle' Maplet construct that can compute the third side length of a triangle using the cosine law. Figure 5-A shows the 'Triangle Maplet' application command lines, and the on-screen display is on Figure 5-B. There are many such pre-calculus geometry problems that a high school or college student encounters in his/her mathematics course. Most such pre-calculus geometry or trigonometric problems can follow this hands-on example to yield a variety of pre-calculus geometry or trigonometric Maplets. Of

course, with a bit of work, this Maplet can be more generalized for solving any number of geometry problems in pre-calculus.

```
restart;
with(Maplets[Elements]):
myTriangleMaplet := Maplet(Window('title' = "myTriangleMaplet", [
        ["Enter One Side Length of Triangle", TextField['TS1']()],
        ["Enter Second Side Length of Triangle", TextField['TS2']()],
        ["Enter Included Angle (in Degrees)", TextField['TIA']()],
        ["Computed Third Side Length of Triangle", TextField['TS3']()],
        [Button['B1']("Compute Third Side Length",
        Evaluate('TS3' = 'sqrt('TS1'^2+'TS2'^2-2*'TS1'*'TS2'*cos((1/180)*'TIA'*Pi))')),
        Button("OK", Shutdown('TS1', 'TS2', 'TIA')),
        Button("Clear", SetOption('TS1' = ""))]])):
Maplets[Display](myTriangleMaplet);
```

Figure 5-A: 'Triangle' Geometry Maplet Application

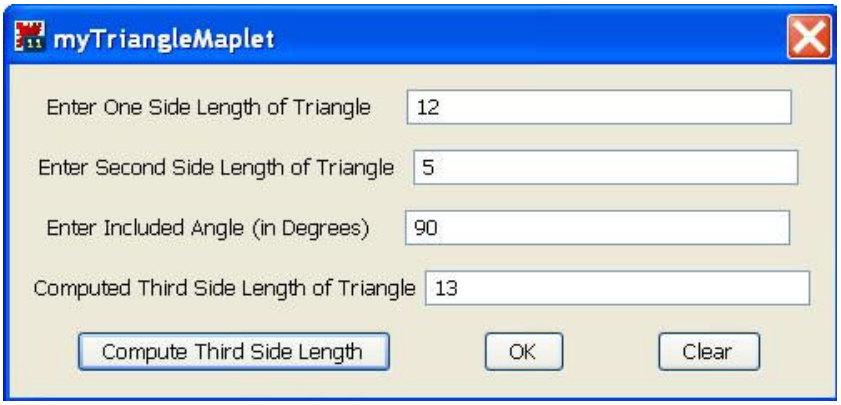

Figure 5-B: 'Triangle' Geometry Maplet Application (on Screen display)

First Order Differential equation Solver Maplet Example 6: 'dsolve' Maplet

One of the important areas of mathematical study is the understanding of solving the first and second order Differential Equations (DE) that occurs in sciences and in engineering. Once having understood the how-to of determining the DE solution, one could use the MAPLE 'dsolve' command to solve most DE's that occur in sciences and engineering. So, instead of the long hand approach of typing in the 'dsolve' command, it would help to develop a 'dsolve' Maplet for solving the physical problem. Figure 6-A shows the powerful and useful 'dsolve' Maplet application commands lines for this First Order DE Maplet development. A similar second order DE 'dsolve' Maplet can be easily developed by the student following Figure 6-A. The on-screen display of the 'dsolve' Maplet is shown as Figure 6-B. The 'dsolve' Maplet

application can also be modified for other similar symbolic solution commands available in MAPLE.

```
restart;
with(Maplets[Elements]):
mydSolveMaplet := Maplet(Window('title' = "mydSolveMaplet", [
        ["Enter First Order Differential Equation", TextField['TS1']()],
        ["Enter Independant Variable, x or t", TextField['TS2']()],
        ["Enter Initial Condition y(0) = ", TextField['TIC']()],
        ["Solution y(x) = ", TextField['TS3']()],
        [Button['B1']("dSolve", Evaluate('TS3' = 'dsolve({'TS1', 'TIC'})')),
        Button("OK", Shutdown('TS1', 'TS2', 'TIA')),
        Button("Clear", SetOption('TS1' = ""))]])):
Maplets[Display](mydSolveMaplet);
```

Figure 6-A: 'dsolve' First Order Differential Equation

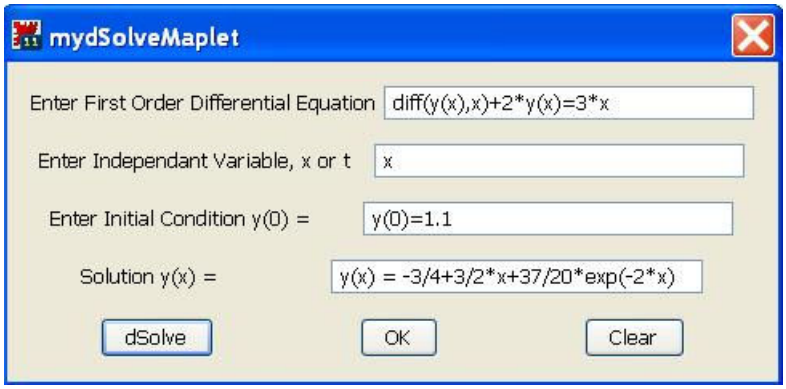

Figure 6-B: 'dsolve' First Order Differential Equation Solver Maplet

Maplet Example 7: Call MAPLE Tutors under Student Package

The **Student** package wthin the MAPLE software is a collection of sub packages designed to assist with the teaching and learning of standard undergraduate mathematics. There are many routines for displaying functions, computations, and theorems in various ways. There is also support for stepping through important computations. The **Student** package is also designed to provide an introduction to the power of the full MAPLE system. The **Student** package contains the following sub packages: precalculus(Precalculus sub-package), linear algebra (Linear Algebra sub-package), single-variable calculus (Calculus1 sub-package), multiple-variable calculus (Multivariate Calculus), and vector calculus(VectorCalculus sub-package).

A ‘call MAPLE Tutor’ Maplet to access or use any of these sub-packages under the Student package is created using a Combo-Box Maplet element. Figure 7-A shows the ‘Call MAPLE Tutor’ Maplet application command lines. The combo box element helps the Maplet user in either choosing the two sub-packages already in the drop-down list or in editing the MAPLE Tutor names in the sub-packages. The names of the sub-packages and the names of all the available MAPLE Tutors under the Student package can be listed by choosing either the ‘with(Student)’ or by editing ‘with(Student[sub-package-name])’ as the value of the ComboBox before clicking the Button ‘Call MAPLE Tutor’ inside the GUI screen. The onscreen window, as shown in Figure7-B, is displayed when the Maplet command is executed.

```
myTutorCallMaplet := Maplet(Window('title' = "myTutorCallMaplet", [
        ["Choose or Modify Tutor Name in BoxValue:",
        ComboBox['CoB1']('value' = "Student[Calculus1][DiffTutor]()",
        ["Student[Calculus1][DiffTutor]()",
        "Student[Precalculus][ConicsTutor]()",
        "with(Student)",  "with(Student[Calculus1]),          "with(Student[Precalculus])"]) ],
        ["End Result of Last Tutor Call", TextBox['Tresult'](3..40)],
        [Button['B1']("Call Maple Tutor", Evaluate('Tresult' = 'CoB1')),
        Button("OK", Shutdown('Tresult', ['CoB1'])),
        Button("Clear", SetOption('Tresult' = ""))]])):
Maplets[Display](myTutorCallMaplet);
```

Figure 7-A: ‘myTutorCall’ under Student Package Maplet Application

Maplet Creation using the ‘Maplet Builder Assistant’

There is an alternate way to design and create Maplets in the MAPLE software. This is an interactive and visual form of creating each of the Maplets elements in the associated Maplet designed for a particular application. This ‘Maplet Builder’ is a Maplet itself provided by the MAPLE software under the ‘Tools’ menu as one of the many ‘Assistant’ maplets. Once all the elements of a Maplet is created interactively and placed visually in its place within a Maplet Window, the element properties are inserted for the Maplets elements used for the particular user defined Maplet applicaton. There are one or many properties for each of these Maplet Elements. Here, the paper discusses only the main properties that are essential for the Maplet working.

The two most important properties of Maplets elements are the ‘Caption’ property for the Button, and the ‘OnClick’ property for the specification of the action expression to be executed on a change or on a click of the Maplets element during the ‘Run’ of the Maplet by the user.

So, these two steps of the interactive approach of creating the Maplet application using the ‘Maplet Builder’ is discussed below:

Step 1: Place each of the Maplet Elements on the 'Layout' pane, as is needed for design of the Maplet application, within its window;

Step 2: Modify the properties of each of the Maplet Elements, as placed, for its proper working during the 'Run' of the Maplet by the 'User'.

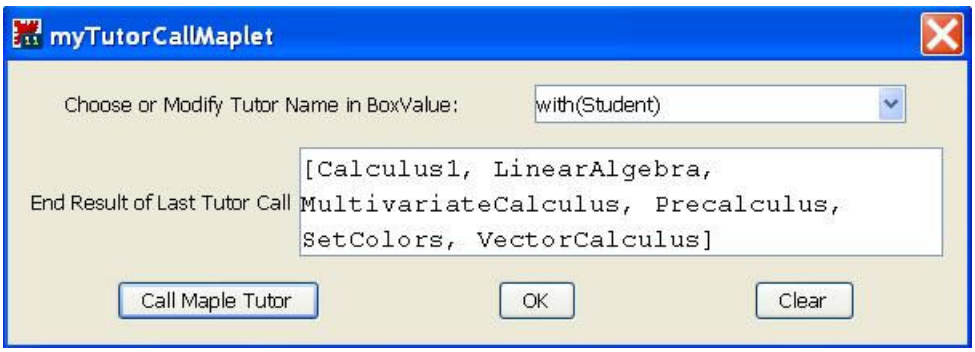

Figure 7-B: 'myTutorCall' Maplet for Student Package Maple tutor Call

Before discussing the two hands-on examples, a short discussion about the 'Maplet Builder' access and its desktop window GUI is given here. A detailed and exhaustive discussion on the 'Maplet Builder' can be obtained in MAPLE using 'Help' facility. It is to be noted that the 'Maplet Builder' is itself an advanced Maplet created by the MAPLE software vendor to help the MAPLE application (program) user-developer to create the various other Maplets. The 'Maplet Builder' Maplet can be run on the MAPLE desktop within the MAPLE environment by choosing the 'Maplet Builder' under the 'Assistants' sub-menu of the 'Tools' menu on the menu bar of the MAPLE desktop.

Maplet Builder Interface

The Maplet Builder contains four panes.

- The 'Palette' pane displays several palettes, which contain Maplet elements, organized by category. The Maplet Builder contains **Body**, **Dialog**, **Menu**, **ToolBar**, **Other**, **Layout**, and **Command** element palettes. For a detailed description of the elements, please read the 'Overview of the Maplet Builder Palette Pane' help page by clicking the 'help' menu inside the 'Maplet Builder'.
- The 'Layout' pane displays the visual elements that are added to the Maplet. The **Layout** pane is the focal point of the Maplet Builder because it shows the visual layout of the Maplet. The Maplet Builder indicates **BoxColumn** and **BoxRow** elements in the **Layout** pane using borders.
- The 'Command' pane displays actions, commands and menu items that you have created for the Maplet.

- The 'Properties' pane displays the properties of an instance of an element in the Maplet. The **Properties** pane allows you to set properties for the elements in the Maplet. For example, the properties of a Button include: the '**foreground'** property to set the color of the button caption (double-click the black foreground color region, which launches a color dialog); the '**onclick'** to set the action that the Maplet performs when the user clicks the button (from the '**onclick'** drop-down menu, and select one of the available '**Action'** elements); the '**caption' t**o set the label on the button (in the '**caption'** text field, enter a string);

Saving, Opening, and Running a Maplet

The three most important operations on any Maplet are the 'open', the 'save' and the 'Run' operations within the 'Maplet Builder'. A short description of how-to of these operations is given below.

- To open a .maplet file within the Maplet Builder, select 'Open' from the 'File' menu. Please note that opening a .maplet file edited outside of the Maplet Builder may not work properly.

- To save a Maplet designed with the 'Maplet Builder', select 'Save' or 'Save As' from the 'File' menu. This will save the Maplet as a .maplet file.

- To run (execute) a Maplet designed with the Maplet Builder, select 'Run**'** from the 'File' menu.

Maplet Example 8: 'Plot' Maplet using the Maplet Builder

All of the previous Maplets can be recreated using the 'Maplet Builder'. However, in this paper, Figure 4-B is recreated interactively using the Maplet Builder. The other Maplets can be recreated in a similar manner. The following is the descriptive steps to create the 'Plot' Maplet interactively using the Maplet Builder.

1) Open a MAPLE session on the computer (assuming the MAPLE software can be accessed locally on the computer or by using the MapleNet);
2) Click the 'Maplet Builder' under the 'Assistants' sub-menu located under the 'Tools' menu. The Maplet Buider interface window opens up under the MAPLE Desktop;
3) To start a new Maplet definition, select 'New' from the 'File' menu;
4) Divide the Maplet window in the 'Layout' pane into several boxes for holding and displaying the arrangement of each of the Maplets Element to be added; (a) To add multiple rows to the Maplet Window in the layout pane, click the layout pane Maplet window, and enter '4' in the **numrows** property; Four boxes appear inside the outer box on the 'Layout' Pane; (b) Click row 1 'Box' in the layout pane, and enter '2' in the **numcolumns** property; (c) Next, click row 2 'Box' in the layout pane, and enter '2' in

the **numcolumns** property; (d) Next, click row 4 'Box' in the layout pane, and enter '3' in the **numcolumns** property;

5) Drag the appropriate Maplet elements in each of these boxes, one or more per box seen in the Maplet layout window; (a) Drag the 'Label' element to row 1, column 1 box; (b) Drag another 'Label' element to row 1 column 2 box; (c) Drag the 'TextField' element to row 2, column 1 box; (d) Drag another 'TextField' element to row 2, column 2 box; (e) Drag the 'Plotter' element to row 3 box; (f) Drag the 'Slider' element under the 'Plotter' element in row 3 box; (g) Drag one 'Button' each to row 4, columns 1, 2 & 3 boxes;
6) Change the properties of these Maplets elements as needed and designed for the Maplet; (a) Click 'Label1' element, change the **caption** property to the string, 'Enter a function of x or t to Plot:'; (b) Click 'Label2' element, change the **caption** property to the string, 'Is it a function of x or t? Enter x or t symbol:'; (c) Click 'Plotter' element, double-click the **background** property, and in the **Color** dialog that appears, click the white font in the top-left corner of the color palette, and then click **OK**; (d) Click 'Button1' element, change the **caption** property to the string 'Plot', and change the **onclick** property dropdown value to select **<Evaluate>**. An **Evaluate Expression** window appears. The **Target** menu lists the available target elements. The **Option** menu lists the available elements for the target selected. The List group box lists the available elements to retrieve the information from. In the **Evaluate Expression** dialog window, ensure the **Target** is set to **Plotter1**, and enter the expression to evaluate as '**plot(TextField1, x = -slider1..slider1)'**. Ensure that a semi-colon (**;**) is not included at the end of the **plot** command. Lastly, click **Ok** in the **Evaluate Expression** window to return to the Maplet Builder. (e) Next, click 'Button2' element, change the **caption** property to the string 'OK', and change the **onclick** property dropdown values to select **<Shutdown>**. In the **Shutdown Event dialog**, click **OK** to return to the Maplet Builder. (f) Next, click 'Button3' element, change the **caption** property to the string 'Clear Plot', and change the **onclick** property dropdown value to select <Set Option>. In the <set Option> dialog window, ensure that the Target is set to 'Plotter1', and choose the value to blank. (g) Lastly, click on the 'Slider1' element and change 'filled' to 'true' value; change 'upper' to value '20'; change the 'majorticks' to value '5'; change 'Onchange' to 'Button1' referencing 'Plot' button.
7) Save the Maplet as 'myPlotMaplet2' by choosing 'Save As' from the 'File' menu;
8) Run 'myPlotMaplet2' Maplet by choosing 'Run' from the 'File' menu.

Maplet Example 9: 'UsingmyIntProcMaplet' Maplet using the Maplet Builder

Instead of sidetracking the student with a very advanced MAPLE procedure development and its use in a Maplet, the 'Integrate' Maplet example 3-A is modified to show how a simple 'myIntProc' MAPLE procedure is used in the 'UsingmyIntProcMaplet' Maplet shown in Figure 8. This simple 'myIntProc' MAPLE procedure checks the user expression string for correct

syntax in input for use in the 'int' MAPLE command. The 'myIntProc' MAPLE procedure is used in the expression for the Button 'Integrate' used in the Maplet.

An interesting combination Maplet can be easily created using the 'myIntProc' MAPLE procedure, and the 'myPlotMaplet2' Maplet example 8 done previously using the Maplet Builder. The combination Maplet can result by modifying 'myPlotMaplet2' Maplet example 8 to create an new application showing two plots, one of a given function and second of its integrated function. Such activity is a minimal work by changing the 'plot' expression in the <Evaluate> action of the 'Plot' Button 'OnClick' property. Such an exercise is left as an exercise for the student.

```
restart;
with(Maplets[Elements]);
myIntProc := proc()
        local integrand, var;
        use Maplets[Tools] in
        integrand := Get('TF1'::algebraic);
        var := Get( 'TF2'::name);
        end use;
        int( integrand, var);
end proc:
myIntProcMaplet :=  Maplet(Window('title'="MyIntProcMaplet", [
["Integrand: ", TextField['TF1']() ],
        ["Variable of integration: ", TextField['TF2'](3)],
        TextBox['TB1']('editable' = 'false', 3..40),
        [Button("Integrate", Evaluate('TB1' = "myIntProc")),
         Button("OK", Shutdown(['TF1', 'TF2', 'TB1'])),
         Button("Clear", SetOption('TF1' = "")) ] ] ) ):
Maplets[Display](myIntProcMaplet);
```

Figure 8: 'UsingMyProc' Maplet Application

An advanced Maplet Example 10: 'BeamDeflection' Maplet

Instead of giving descriptive steps to create the 'BeamDeflection' Maplet, the on-screen display of this advanced Maplet is shown in Figure 9. This example is an advanced Maplet that is useful either in Civil or Mechanical Engineering courses. The Maplet Application Developer student can choose either the commands lines approach or the Maplet Builder to create this advanced 'BeamDeflection' Maplet. The Maplet elements used in this Maplet are the following: (1) Several Labels; (2) Several Text Fields; (3) Four Buttons; (4) One Plotter; (5) One output TextBox.

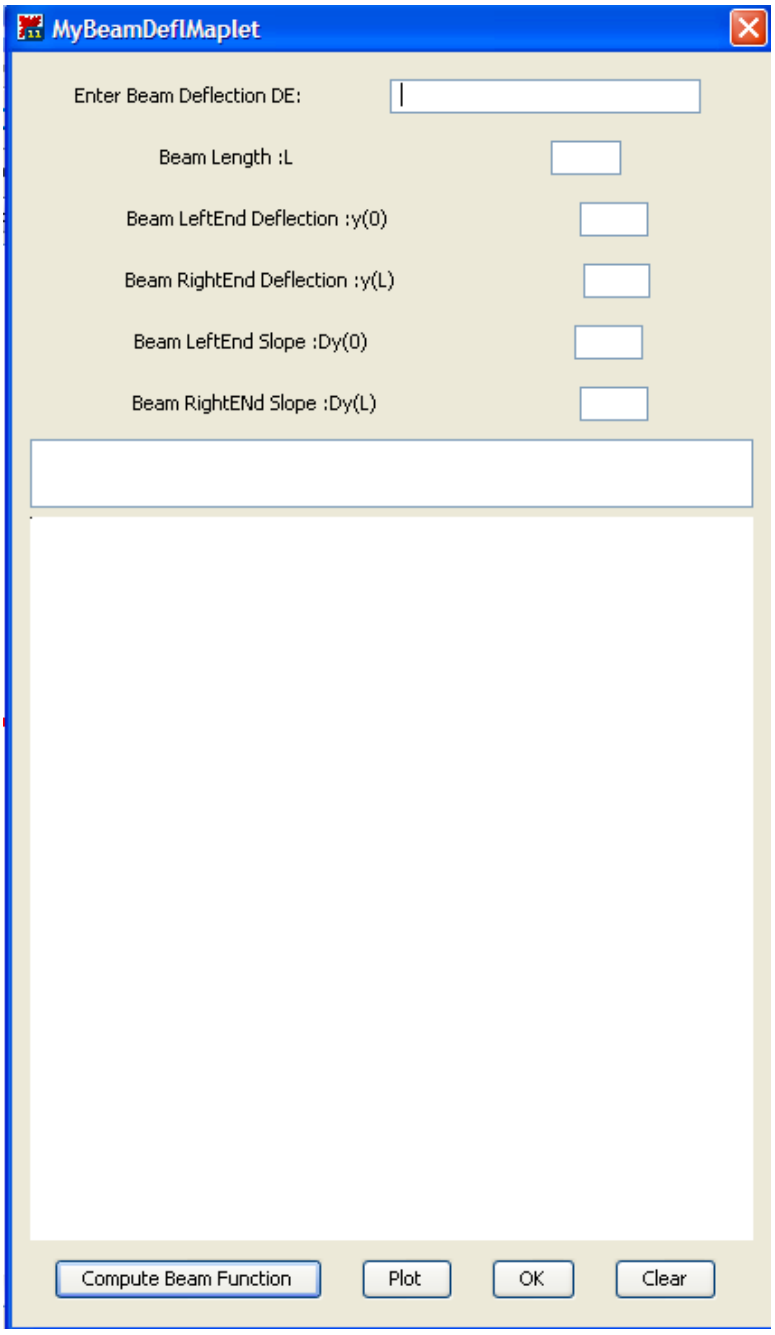

Figure 9: 'BeamDeflectionProc' Maplet Application

Conclusions

A motivational discussion is given to state the importance of teaching the MAPLE software tool for Maplet programming and construction within MAPLE. A simplified view of how to create

Maplets within the MAPLE destop work environment through hands-on Maplets examples is discussed in this paper. The essential concepts of such Maplets creation is outlined and emphasized. No attempt in this paper is made to be exhaustive on the subject. The 'Introduction to Programming[1]' MAPLE guide has one chapter devoted to the Maplets building though the 'Commands lines' approach to help the application user in terms of writing out the 'Body' of the Maplets elements. However, no attempt is made in the 'Introduction to Programming' guide to give realistic hands-on Maplet examples creation pertinent to engineering or sciences. This paper is an attempt to fill this gap along with the teaching of Maplet Creation through hands-on examples to the high school or college student of engineering, mathematics or sciences.

Also, though it is very important to understand the usual 'Procedural' programming constructs in advanced Maplets creation, no attempt has been made in this paper to teach the 'Procedural' programming to create the MAPLE procedures or functions that can be used within the Maplet to create more advanced and sophisticated Maplets. The one very simple example number 10 shows how an outside user defined procedure can be used with the Maplet. The 'Advanced Programming[2]' MAPLE guide has an exhaustive and detailed discussion on the creation of MAPLE procedures and functions. However, the advanced programming guide also details many of the internal workings of MAPLE that may prove to be difficult for the MAPLE student, especially one who wishes to learn limited procedures or functions of MAPLE quickly.

Many advanced procedures or functions or Maplets are available, either free or at a low cost, on the MAPLE website. If the student has an interest in learning such advanced procedures, Maplets or functions, he or she can open and examine any of these advanced Maplets or procedures within MAPLE or in any text editor, as per the saved file extensions dictate.
Please note that MAPLE is very sensitive to text characters, and traps any wrongly typed MAPLE statement, either mistyped syntax or an incorrect use of MAPLE command. This is both good and bad for the Maplet designer, good in the sense of catching mistyped wrong syntax, and bad in the sense of expecting the MAPLE user to be completely familiar with any MAPLE or Maplet statement that are used in the Maplet. This causes some frustration, and consumes additional time to develop medium to advanced Maplets. One way to speed up learning the application of any MAPLE command or the correct use of Maplet elements is to read on how these MAPLE commands or Maplet elements are used in the available working medium-to-advanced Maplets. Of course, a certain minimum detailed understanding of the use of MAPLE commands is necessary for any Maplet creation, and MAPLE use knowledge can be taught easily by reading the textbooks like learning 'MAPLE by Examples[3]'.

Only one student (the first author) has used the approach outlined in this 'hands-on examples' paper and this has been the first attempt to teach the Maplets creation using this approach. The teacher (second) author felt it more appropriate to disseminate the 'Maplet creation' Learning study so he can get input suggestions or comments from the academic community for further

improvements. More refinements and more simple-to-medium hands-on examples are possible based on the provided examples in this paper. A future look into teaching more advanced hands-on Maplet examples will occur based on the interest of the student community. For this and other reasons, all these simple hands-on Maplet examples using the command lines, will be soon available from the second author, or from the MAPLE website after being accepted and uploaded on the MAPLE site.

Bibliography

1. 'Introduction to Programming Guide', MAPLE 12, MapleSoft, 2008
2. 'Advanced Programming Guide', MAPLE 12, MapleSoft, 2008
3. Martha L. Abell and James P. Braselton, 'MAPLE By Example, Third Edition', Elsevier, 2005