Advanced FEMAP Programming with COM/OLE

Patrick Kriengsiri
Agenda

- Programming Outside the FEMAP Sandbox
- Creating FEMAP Programs with Visual Studio
  - VB.Net and VC# Programs
- Registering Applications in FEMAP
- FEMAP and Microsoft Excel
- Efficient FEMAP Programming Techniques
Programming Outside the FEMAP Sandbox

Programming for FEMAP is not limited to just the API Programming Window.

- By using the Windows COM/OLE interface to create a reference to the FEMAP type library, external development environments can be used to develop for FEMAP.
- Conversely, other type libraries can be referenced from FEMAP to use FEMAP to control those programs.
Why Develop Using COM/OLE

There are numerous advantages from using the COM/OLE interface in your FEMAP programs

- When using Visual Studio and VB.Net, VC#, VC++.Net, etc, other languages are available and any control in the Windows Forms framework is available
- A language other than Visual Basic can be used
- Much more rich and complex user interfaces can be created
- Programs can be compiled that may improve performance if written correctly
- The majority of the FEMAP-related code is the same
- FEMAP can also be used to drive other programs
  - The FEMAP portion of the code in the API Programming Window is identical
  - More objects are available
  - FEMAP can be used to populate data in a spreadsheet or document
- When creating FEMAP programs using other programs with a VBA development environment, the code is nearly identical
- When using the API Programming Window within FEMAP, a connection to the FEMAP application is made with the `feFEMAP()` function
- This is not available when developing programs external to FEMAP
- Each language has its own way of connecting to a FEMAP instance
- FEMAP can be connected to either by creating a new application or attaching to an instance that is already running
- Externally working with FEMAP objects is very similar using the API programming window – often the main differences are syntactical
External FEMAP Programs – Basic Process

- Due to COM limitations, when using the FEMAP API via an external program, only the first instance will be controlled (this limitation is no longer present when using the API Programming Window).
- Regardless of which type library version (i.e., 10.1, 10.2, 10.3) is referenced, when creating a new FEMAP instance programmatically, the most recent version of FEMAP launched through the GUI will be used.
- When the application object loses scope it must be dealt with according to how its created.
Creating FEMAP Programs in Visual Studio

Visual Studio can be used to write FEMAP programs.

- Multiple language choices
- Enhanced forms and controls are available
- Programs are compiled, rather than interpreted so performance may increase (COM overhead is still present)

Tip

Development is not limited to a paid version of VS. The free “Express” versions are perfectly capable and, from a FEMAP standpoint, offer the same capabilities.
Creating FEMAP Programs in Visual Studio

Porting your FEMAP program written using the API Programming Window begins with adding a reference to the FEMAP Type Library.

If the FEMAP Type Library is not available in the Reference Manager, look for it in the FEMAP install directory. The filename is “femap.tlb”.

Tip: If the FEMAP Type Library is not available in the Reference Manager, look for it in the FEMAP install directory. The filename is “femap.tlb”.
Creating / Porting VB.Net Programs

Most API programs written in the FEMAP API Programming window can easily be ported to FEMAP with a few simple changes

- The connection to FEMAP needs to be changed
- Any variables that were created dynamically need to be declared prior to use
- The Set instruction is no longer used – simply remove them
- Variants are no longer available in .Net
  - They have been replaced with the Object data type
  - Most uses of Variants can simply be converted to Objects
- Returned arrays may have to be re-written
- The Debug object is no longer available
- Dialogs must be re-written
VB.Net – Creating the Application Object

Creating the FEMAP Object

- The feFemap() method is not available outside of FEMAP
- Creating the FEMAP Object
  - Connect to an existing FEMAP instance
    ```vba
    Dim app As femap.model
    app = GetObject(,, "femap.model")
    ```
  - Create a new FEMAP instance
    ```vba
    Dim app As femap.model
    app = CreateObject("femap.model")
    app.feAppVisible(True)
    ```
  - If creating a new FEMAP object, the application is not visible by default and must be shown manually

*Info*

External connections to FEMAP can only be made to the FIRST open FEMAP instance
VB.Net – Data Types

Most data types will port directly from VBA to VB.Net, however Variants are not available

- Generally Variants can just be converted to Objects
- When used with arrays, it’s possible that some code may need to be rewritten
- Make use of IntelliSense in Visual Studio for clues as to what data types are required as it may not entirely match with what is specified in the API Programming Manual
VB.Net – Working with Arrays

When working with arrays,

- Object is NOT the same as Object()
- Arrays passed in as Variants must be passed in as Objects
  - It can be defined as an Object from the start
  - Inline casts using CObj( myArray ) are often faster
- Methods that return arrays as Variants must be returned to Objects, not arrays of the actual type
Example:

```vbnet
Dim app As fmap.model
Dim fSurf As fmap.Surface
Dim rc As zReturnCode

Dim ddInit(3) As Double
Dim ddProj() As Double
Dim odProj As Object

app = GetObject(, "femap.model")
fSurf = app.feSurface

ddInit(0) = 0.0
ddInit(1) = 0.0
ddInit(2) = 0.0

fSurf.Get(1)
fSurf.ProjectOnto(CObj(ddInit), 0.0, odProj)
ddProj = odProj
```
Example:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>odProj</td>
<td>{Double()}</td>
<td>Object</td>
</tr>
<tr>
<td>Double()</td>
<td>{Length=3}</td>
<td>Double()</td>
</tr>
<tr>
<td>(0)</td>
<td>-0.70710678118655268</td>
<td>Double</td>
</tr>
<tr>
<td>(1)</td>
<td>0.0</td>
<td>Double</td>
</tr>
<tr>
<td>(2)</td>
<td>-0.70710678118654335</td>
<td>Double</td>
</tr>
<tr>
<td>ddProj</td>
<td>{Length=3}</td>
<td>Double()</td>
</tr>
<tr>
<td>(0)</td>
<td>-0.70710678118655268</td>
<td>Double</td>
</tr>
<tr>
<td>(1)</td>
<td>0.0</td>
<td>Double</td>
</tr>
<tr>
<td>(2)</td>
<td>-0.70710678118654335</td>
<td>Double</td>
</tr>
</tbody>
</table>
VB.Net – Example: FEMAP Stress Linearization Tool
Creating / Porting Programs to VC#.Net

Converting FEMAP API Programs to C# is similar to converting them to VB.Net, however is a little more involved due to the differences in syntax

- Like VB.Net, VC# is still a managed language, so pointers are still not of concern
- C# does not need line continuation characters and each “command” is ended with a semicolon
- All variables must be declared prior to use and must have a type
  - Declare variables as
    - `<type> <variable name>` rather than
    - `Dim <variable name> as <type>`
- Array indices are handled with brackets as opposed to parenthesis
- All arrays are zero-based
- When declaring an array of a pre-defined size, the number provided is the total count of the array, not the upper bound
Creating / Porting Programs to VC#.Net

- In addition to the syntax changes, some of the native language calls may no longer be the same as in VBA
  - Ex: Square root
    - VBA: `sqr(<value>)`
    - C#: `Math.sqrt(<value>)`;
- Simple conversions such as between doubles and singles or integers and shorts may be performed automatically by the compiler, but generally, type conversions will be required
- Like VB.Net
  - The connection to FEMAP needs to be changed
  - The Set instruction is no longer used – simply remove them
  - Variants have been replaced by Objects
  - Returned arrays may have to be re-written
  - The Debug object is no longer available
  - Dialogs must be re-written
Creating the FEMAP Application Object

- Creating a new FEMAP Session
  ```csharp
  femap.model App;
  App = new femap.model();
  ```

- Connecting to an existing FEMAP session
  ```csharp
  femap.model App;
  ```

- Rather than using the fully qualified name for GetActiveObject, this can be shortened to
  ```csharp
  femap.model App;
  App = (femap.model) Marshal.GetActiveObject("femap.model");
  ```

if a reference is made to the namespace:

```
class
using System.Runtime.InteropServices;
```
VC#.Net – Working with Arrays

Like in VB.Net, the Variant data type does not exist – the generic “Object” data type is used instead

- When sending data in to a function, use of an explicit cast is okay
  - `fs.AddArray( 10, (object) myIntArray );`
- Because of the more type-strict nature of VC#, generally two variables need to be created; one “object” variable will receive the output. This object will then be explicitly cast to the appropriate data type
- While VB.Net, it is usually okay to work with objects as if they were Arrays, however this will cause issues in VC#
- In addition, VC# requires that objects that will be returned to be prefaced with the out keyword
Example:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using femap;
using System.Runtime.InteropServices;

namespace FEMAP_Sandboxes_VS2012
{
    class Program
    {
        static void Main(string[] args)
        {
            femap.model App;
            App = (femap.model)Marshal.GetActiveObject("femap.model");

            femap.Set fs;
            femap.Node fn;

            fs = App.feSet;
            fn = App.feNode;
        }
    }
}
```
Example (continued):

```csharp
object oNodeID;
object oXyz;
int[] nnNodeId;
double[] ddXyz;
int numNode;

fs.AddAll(zDataType.FT_NODE);
fn.GetCoordArray(fs.ID, out numNode, out oNodeID, out oXyz);
nnNodeId = (int[]) oNodeID;
oXyz = (double[]) oXyz;

App.feAppMessage(zMessageColor.FCM_COMMAND,
    "First Node: " + nnNodeID[0].ToString());
```
Registering an Application in FEMAP

External applications with a user interface can be registered as an Add-in pane using an Application method

<table>
<thead>
<tr>
<th>feAppRegisterAddInPane() – Registers an add-in application with FEMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BOOL</strong> bRegister</td>
</tr>
<tr>
<td><strong>INT4</strong> windowID</td>
</tr>
<tr>
<td><strong>INT4</strong> messageID</td>
</tr>
<tr>
<td><strong>BOOL</strong> bPreventClose</td>
</tr>
<tr>
<td><strong>BOOL</strong> bPreventHide</td>
</tr>
<tr>
<td><strong>INT4</strong> location</td>
</tr>
<tr>
<td><strong>INT4</strong> neighborID</td>
</tr>
</tbody>
</table>

* There is an additional method, feAppRegisterAddInPane2(), that will register the dialog without removing the menu or containing window
Registering an Application in FEMAP

Example: Registering an Add-In Pane

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim app As femap.model
        app = GetObject(, "femap.model")

        app.feAppRegisterAddInPane(True, Me.Handle, Me.Handle, False, False, 2, 0)
    End Sub

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        MsgBox("clicked")
    End Sub

End Class
```
Registering an Application in FEMAP

Example: Registering an Add-In Pane
Developing for FEMAP using Microsoft Excel

Like Visual Studio, FEMAP can also be controlled from Microsoft Excel

- Because Excel uses VBA, the porting is nearly seamless
- The creation of the FEMAP object will be identical to VB.Net
  - All restrictions remain the same, such as only working with the first open FEMAP session
- Dialogs will have to be recreated
- One change has to be made to the code – any method that returns a code will have to have that value captured
  - Invalid: `app.feAppMessage(FCM_COMMAND, Test")`
  - Valid: `rc = app.feAppMessage(FCM_COMMAND, Test")`

Tip

Access the Excel VBA environment with the keyboard shortcut ALT-F11
Developing for FEMAP using Microsoft Excel

- In Excel VBA, register the FEMAP type library using Tools -> References

![References - VBAProject](image)

- Create a subroutine or a function

```vba
Sub MyFemapApp()
    Dim app As femap.model
    Set app = GetObject(, "femap.model")
    Dim rc As zReturnCode
    rc = app.feAppMessage(FCM_COMMAND, Test"
End Sub
```
Developing for FEMAP using Microsoft Excel

Example

Shear load for fastener #44, located in the figure below, is calculated from the membrane forces in the x (Nxx) and y (Nyy) directions. Positive shear for the upper skin element is shown.

Fastener #44

Nxx = 213.3 Nm, from FEMAP
Nyy = 12,759 Nm, from FEMAP

Pitch

Nxx = -266.6 lbs
Nyy = 17,359 lbs

Fastener load = 267.18 lbs
Developing for FEMAP using Microsoft Excel

Example

Code:

Function GetFemapLoad( ElementID as Integer, _
    OutputSetID as Integer, _
    OutputVectorID as Integer )

    Dim app as femap.model
    Dim fo as femap.Output

    Set app = GetObject( , “femap.model”)
    Set fo = app.feOutput

    fo.SetID = OutputSetID
    fo.Get( OutputVectorID )

    GetFemapLoad = fo.Value(ElementID)

End Function
Creating / Modifying Excel Workbooks from FEMAP

The FEMAP API Programming Window can also be used to control other programs, such as Excel

- Add a reference to the type library (Right Click -> References)
Creating / Modifying Excel Workbooks from FEMAP

By adding the reference, the Excel class becomes available for use in the API Programming Window

- The main application is Excel.Application objects
  - Like the FEMAP object, when creating a new Excel object, it must be made visible
- Workbooks (files) are Excel.WorkBook objects
- Worksheets are Excel.WorkSheet objects
Creating / Modifying Excel Workbooks from FEMAP

Example:

```vba
Sub Main
    Dim App As femap.model
    Set App = feFemap()

    Dim xapp As Excel.Application
    Dim xWB As Excel.Workbook
    Dim xWS As Excel.Worksheet

    Set xapp = CreateObject("excel.application")
    Set xWB = xapp.Workbooks.Add()
    Set xWS = xWB.Worksheets(1)

    xWS.Cells(1,1) = "Hello World!"

    xapp.Visible = True

End Sub
```
Efficient FEMAP Programming Techniques

Whether programming for FEMAP using the API Programming Window or externally with another IDE, many of the same rules still apply.

Often times the number one issue related to performance is COM overhead:

- This is the transaction overhead required with making a call to the FEMAP API (or any other COM call).
- Traditional VB calls are not affected, nor are native calls in other languages.
- There is nothing that can be done to reduce the overhead on individual calls.
- The COM overhead varies in the API programming window vs. external programs (such as via Visual Studio).
- This depends on the types of calls made as well as the data types being exchanged.
- Database transaction calls with entities tend to perform better inside FEMAP.
- String-related calls tend to perform better in Visual Studio programs.
Many times, the greatest increase in program performance revolves around reducing the overall number of COM calls

- Avoid using individual Get/Put calls for entities such as Node or Element that have GetAll/PutAll methods
- Avoid repeatedly using the .Value method of the Output object – use the helper methods instead
- Avoid retrieving single items from the Results Browsing Object
- When possible, write native methods that can perform some of the same functionality as FEMAP methods, such as the linear algebra methods

Info
The difference between 1,000 and 10,000 calls may not amount to much, but the difference between 10,000 calls and 1,000,000 calls may be minutes or hours
Efficient FEMAP Programming Techniques - Miscellaneous Tips

- Avoid writing large amounts of data to the Message Window
- Lock the UI by calling `feAppLock()` at the beginning of your API program
  - Don’t forget to call `feAppUnlock()`
- Updates to the status bar should only be made judiciously
  - The status bar is not infinitely variable
  - There is no usability gain by updating the status bar 10000 times vs 20 times
- Unless necessary, avoid graphics redraws until the end of your program
- Hide the FEMAP UI
  - This should be used as a last result to extract every ounce of performance
  - Gains may be marginal
  - Makes for a degraded user experience
- If building your model with specific entity ID’s try to keep the ID’s under 5,000,000-10,000,000; as entity ID’s increase, they start to populate different sections of the database and could increase run time
Advanced FEMAP API with COM/OLE

Patrick Kriengsiri
FEMAP Development

411 Eagleview Blvd
Exton, PA, 19341

Phone: 404-353-6596

E-mail: patrick.kriengsiri@siemens.com