

Introduction to API

Understanding Femap API terminology - with
a short introduction into the how's and why's

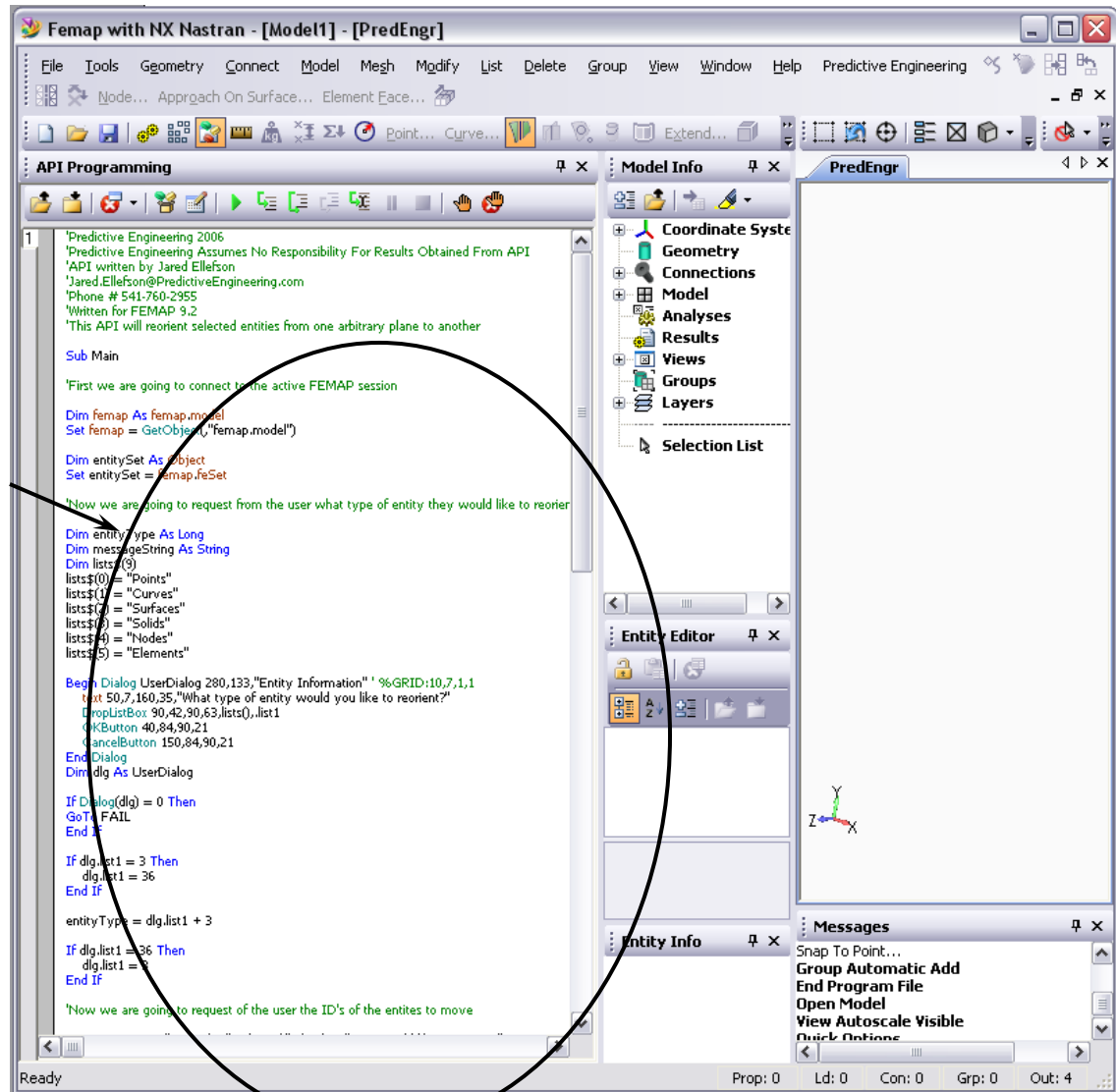
What's in this section:

- What is an API?
- What is object orientated programming?
- API organization
- How to define the FEMAP API objects and use them
- Anatomy of a simple API

What is an API?

- The FEMAP API is an OLE/COM based programming interface and is ***object oriented programming***. If you have never programmed in an object oriented code, it can seem quite different and foreign.
- **API** means “Application Programming Interface”. It is important to understand that the API script you write is ***not part of FEMAP***, but is a stand alone program that is ***interacting*** with FEMAP.
- There are a number of codes that can call FEMAP through the API: Visual Basic, VBA (Excel, Word, Access, ...), C, or C++.
- The most commonly used codes used are Visual Basic, VBA, and WinWrap.
- WinWrap is a flavor of Visual Basic that is included with FEMAP. In the FEMAP interface, WinWrap is ***uncompilable***, for this reason many choose not to use it, but it is a very convenient way to program if your specific application does not need to be compiled.
- This tutorial will focus exclusively on using WinWrap via the FEMAP API window.

- This is the optional FEMAP API editing window.
- Although the window appears to be part of your FEMAP session, it is not. It is merely a code editing tool.



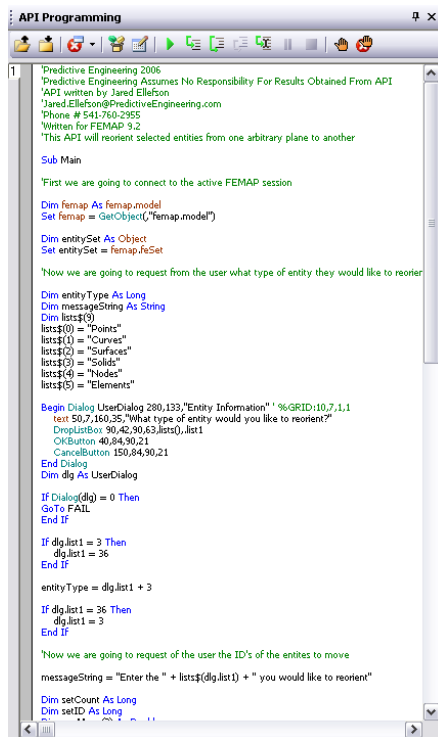
What is object oriented programming?

- Traditional programming is usually seen as being a set of *functions*, or simply as a list of *instructions*.
- Object Oriented Programming (or OOP) can be seen as a group of *Objects* that cooperate with each other. Each of the objects have their own distinct set of capabilities.
- OOP programming is quite complex and includes topics such as *inheritance*, *encapsulation*, among others. These more complex ideas are not immediately necessary, and will not be discussed. In fact, the FEMAP API has made it pretty much unnecessary to ever have to learn these concepts.

Organization

It is helpful to think of each of the entities as being separate.

- Your Visual Basic code acts like a traditional code, i.e. as a set of instructions.
- The VB code makes requests of the API, which then acts upon those requests either by retrieving from and putting things into the FEMAP database.
- FEMAP is a database, which only holds and displays data.



```
'Predictive Engineering 2006
'Predictive Engineering Assumes No Responsibility For Results Obtained From API
'API written by Jared Ellefson
'Jared.Ellefson@PredictiveEngineering.com
'Phone # 541-760-2955
'Written for FEMAP 9.2
'This API will reorient selected entities from one arbitrary plane to another

Sub Main

'First we are going to connect to the active FEMAP session
Dim femap As Femap.model
Set femap = GetObject("femap.model")

Dim entitySet As Object
Set entitySet = femap.feSet

'Now we are going to request from the user what type of entity they would like to reorient

Dim entityType As Long
Dim messageString As String
Dim lists() As String
Dim listsCount As Integer
lists(0) = "Points"
lists(1) = "Curves"
lists(2) = "Surfaces"
lists(3) = "Solids"
lists(4) = "Nodes"
lists(5) = "Elements"

Begin Dialog UserDialog 280,133,"Entity Information" ' %GRID:10,7,1,1
text 50,7,160,35,"What type of entity would you like to reorient?"
DropDownBox 90,42,90,63,lists(),list1
OKButton 40,84,90,21
CancelButton 150,84,90,21
End Dialog
Dim dlg As UserDialog
Dim dlgType As Long

If Dialog(dlg) = 0 Then
GoTo FAIL
End If

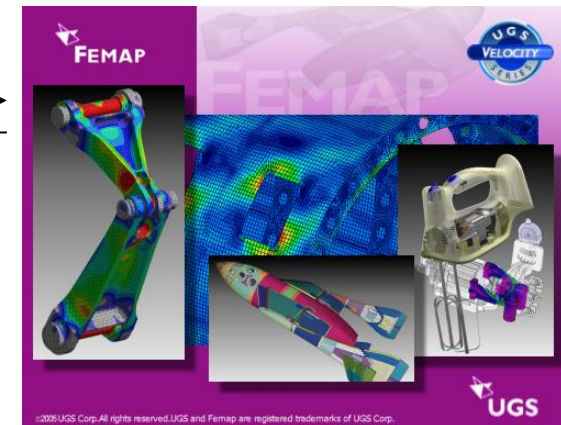
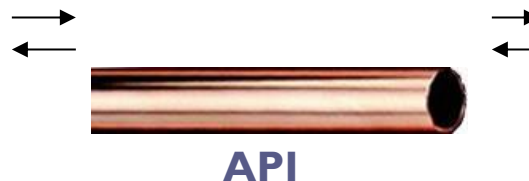
If dlg.list1 = 3 Then
dlg.list1 = 36
End If

entityType = dlg.list1 + 3

If dlg.list1 = 36 Then
dlg.list1 = 3
End If

'Now we are going to request of the user the ID's of the entities to move
messageString = "Enter the " + lists(dlg.list1) + " you would like to reorient"

Dim setCount As Long
Dim setID As Long
```



Visual Basic Code

FEMAP

The FEMAP API Objects

The objects found in the FEMAP API fall into two categories:

- The *FEMAP Application Object*
- *Stand Alone Objects*

Generally speaking, these objects act have the following properties:

- The *FEMAP Application Object* has all the properties needed to ***create*** things. It is the object that will be used to create geometry, measure things, mesh geometry, delete entities, etc.
- The *Stand Alone Objects* are used to ***manipulate*** existing entities.

Object syntax

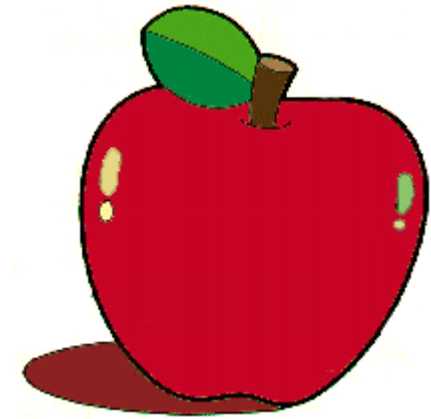
`rc = object.capability(requirements, output)`

The syntax in the above statement is standard. The object is what is being “asked” to act. The capability is what the object is being asked to do. The requirements are what the object needs in order to execute the capability. The output is what the object will “produce,” although often times capabilities will have no output.

The term **rc** is the return code and will generate a specific value depending on a number of object success states. The **rc** status is explained in more detail in a subsequent slide. The reality is that it is an extra - sort of a bonus feature to help in debugging or controlling sequence execution.

Objects that produce an output

How this all works is best explained by a more concrete example. Think of an object as a person, a person who will do things that you ask. Will call this person “Mike”. Say we want “Mike” to go to the store and buy an apple. In order for “Mike” to do this, we need to provide him with a car and money. For this capability, “Mike” will produce an output: an apple. The statement would look like this:



OBJECT	CAPABILITY	REQUIREMENTS	OUTPUT
↘	↘	⌈	↘
rc = Mike.GetApple(Money, Car, Apple)			

Objects that produce NO output

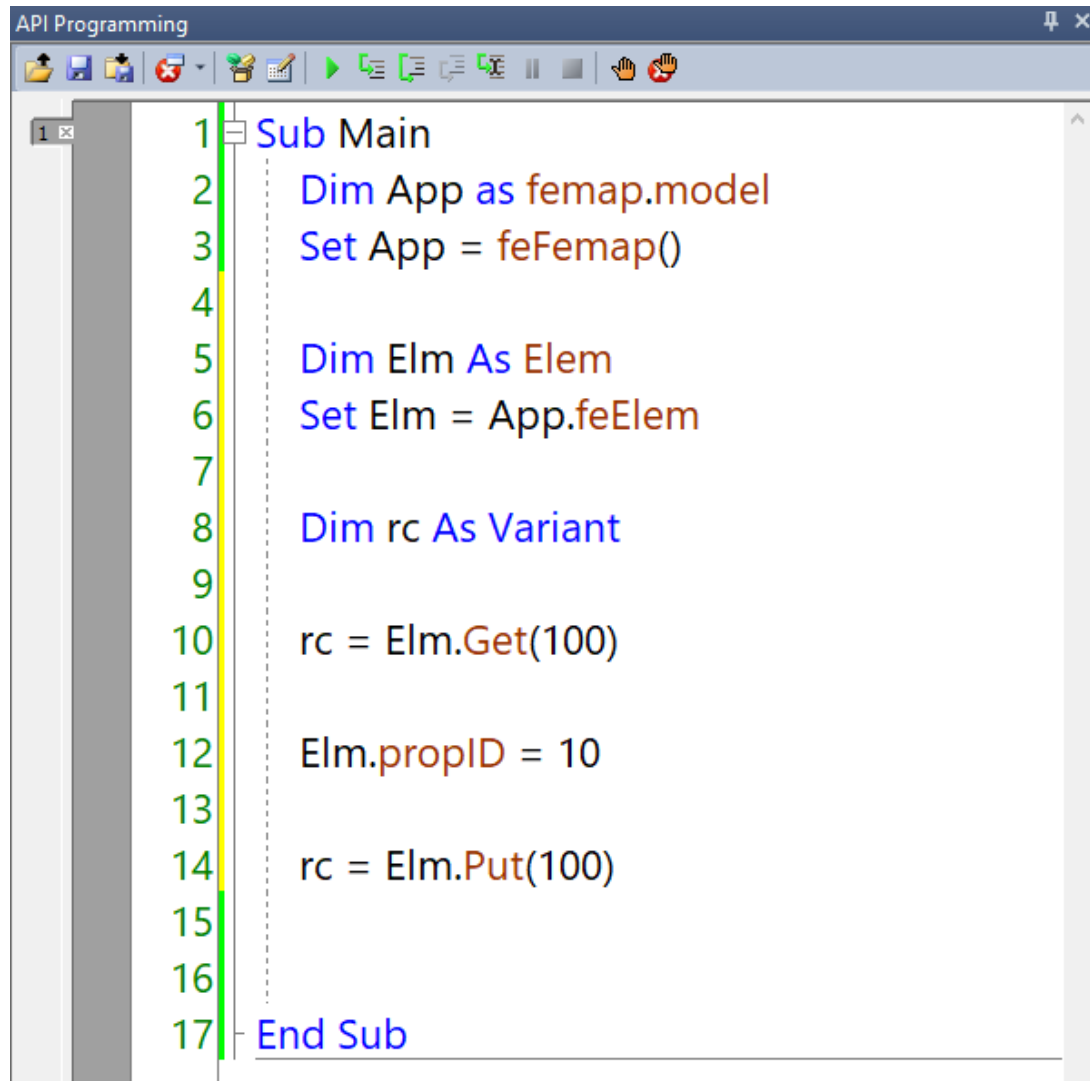
Now suppose we want “Mike” to wash the dishes in the kitchen. We need to provide him with the dishes, soap, a sponge, and a sink. After he is done he will produce NO output for us because we haven’t asked him to bring us anything. All we have done is ask him to go off and do something. The statement looks much like the previous one.



OBJECT	CAPABILITY	REQUIREMENTS	NO OUTPUT
↓	↓	⏟	
rc = Mike.DoDishes(Dishes, Sponge, Soap, Sink)			

Sometimes we ask objects to organize things. Sometimes we will ask them to create or move things. The only time objects will have output is if we ask them to **bring** us something specific. This most likely seems fairly abstract, but once you see how it actually works you will see that it is very intuitive.

Assign propID 10 to Element ID 100



The screenshot shows a window titled "API Programming" with a standard toolbar. The code editor contains a VBA subroutine named "Sub Main". The code is as follows:

```
1 Sub Main
2   Dim App as femap.model
3   Set App = feFemap()
4
5   Dim Elm As Elem
6   Set Elm = App.feElem
7
8   Dim rc As Variant
9
10  rc = Elm.Get(100)
11
12  Elm.propID = 10
13
14  rc = Elm.Put(100)
15
16
17 End Sub
```

Element properties

Property	Description
INT4 color	The element color.
INT4 layer	The layer associated with the element.
INT4 type	Type of element: Rod=1, Bar=2, Tube=3, Link=4, Beam=(Lin=5 Para=37), Spring=6, DOFSpring=7, CurvedBeam=8, Gap=9, PlotOnly=10, ShearPanel (Lin=11 Para=12), Membrane (Lin=13 Para=14), BendingOnly (Lin=15 Para=16), Plate (Lin=17 Para=18), PlaneStrain (Lin=19 Para=20), Laminate (Lin=21 Para=22), Axisymmetric (Lin=23 Para=24), Solid (Lin=25 Para=26), Mass=27, MassMatrix=28, Rigid=29, StiffnessMatrix=30, CurvedTube=31, PlotOnlyPlate=32, SlideLine=33, Contact=34, Axisymmetric Shell (Lin=35 Para=36), Weld=38
INT4 propID	ID of property referenced by the element. This is not required for certain property types, like plot-only and rigid elements. These elements have no properties or materials.
INT4 topology	The shape of the element: 0=Line, 2=Tri3, 3=Tri6, 4=Quad4, 5=Quad8, 6=Tetra4, 7=Wedge6, 8=Brick8, 9=Point, 10=Tetra10, 11=Wedge15, 12=Brick20, 13=Rigid, 15=MultiList, 16=Contact, 17=Weld

Anatomy of a simple API

Sub Main

```
Dim femap As femap.model
Set femap = GetObject(,"femap.model")
```

```
Dim entitySet As Object
Set entitySet = femap.feSet
```

```
Dim vecMove(2) As Double
vecMove(0) = 10
vecMove(1) = 0
vecMove(2) = 0
```

```
Dim entityType as long
entityType = 7
```

```
Dim messageString as String
messageString = "Please Select the Nodes You Would Like To
                Move"
```

```
rc = entitySet.Select(entityType,True,messageString)
```

```
Dim setID As Long
setID = entitySet.ID
```

```
Dim vecLength As Double
```

```
rc = femap.feVectorLength(vecMove,vecLength)
```

```
rc = femap.feMoveBy( entityType, setID, False, vecLength,
                    vecMove)
```

End Sub

Now we'll walk through a simple API. All this API does is move a set of selected nodes 10 units in the x direction. Yes, there is a function that will do this directly without an API, but we are starting simple. The entire script is shown on the left. We will walk through each step in this API.

Defining an object

Sub Main

```
Dim femap As femap.model  
Set femap = GetObject("femap.model")
```

```
Dim entitySet As Object  
Set entitySet = femap.feSet
```

```
Dim vecMove(2) As Double  
vecMove(0) = 10.0  
vecMove(1) = 0  
vecMove(2) = 0
```

```
Dim entityType as long  
entityType = 7
```

```
Dim messageString as String  
messageString = "Please Select the Nodes You Would Like  
To Move"
```

```
rc = entitySet.Select(entityType,True,messageString)
```

```
Dim setID As Long  
setID = entitySet.ID
```

```
Dim vecLength As Double
```

```
rc = femap.feVectorLength(vecMove,vecLength)
```

```
rc = femap.feMoveBy( entityType, setID, False, vecLength,  
vecMove)
```

End Sub

Sub Main at the top of the script signifies that what follows is the main program.

Next we create an object called **femap**. We then set this object equal to the current femap session. Essentially what this does is create the *FEMAP Application Object* and appropriately calls it *femap*. So the object *femap* now has all the properties of the *FEMAP Application Object*.

What we want to do will also require the help of another object, called the **Set** object. **entitySet** now has all the properties inherent in the **Set** object.


Dimensioning variables

```
Sub Main
```

```
Dim femap As femap.model  
Set femap = GetObject("femap.model")
```

```
Dim entitySet As Object  
Set entitySet = femap.feSet
```

```
Dim vecMove(3) As Double  
vecMove(0) = 10.0  
vecMove(1) = 0  
vecMove(2) = 0
```



```
Dim entityType as long  
entityType = 7
```



```
Dim messageString as String  
messageString = "Please Select the Nodes You Would Like To Move"
```

```
rc = entitySet.Select(entityType,True,messageString)
```

```
Dim setID As Long  
setID = entitySet.ID
```

```
Dim vecLength As Double
```

```
rc = femap.feVectorLength(vecMove,vecLength)
```

```
rc = femap.feMoveBy( entityType, setID, False, vecLength, vecMove)
```

```
End Sub
```

Next we declare a 3 dimensional array, composed of 8-byte real numbers called vecMove. This array will represent the vector along which the translation will take place. We then specify each value in the array.

We also declare a variable called entityType as a 4-byte integer and assign it a value.

Using the capabilities of an object

Sub Main

```
Dim femap As femap.model  
Set femap = GetObject("femap.model")
```

```
Dim entitySet As Object  
Set entitySet = femap.feSet
```

```
Dim vecMove(2) As Double  
vecMove(0) = 10.0  
vecMove(1) = 0  
vecMove(2) = 0
```

```
Dim entityType as long  
entityType = 7
```

```
Dim messageString as String  
messageString = "Please Select the Nodes You Would Like To  
Move"
```

```
rc = entitySet.Select(entityType,True,messageString)
```

```
Dim setID As Long  
setID = entitySet.ID
```

```
Dim vecLength As Double
```

```
rc = femap.feVectorLength(vecMove,vecLength)
```

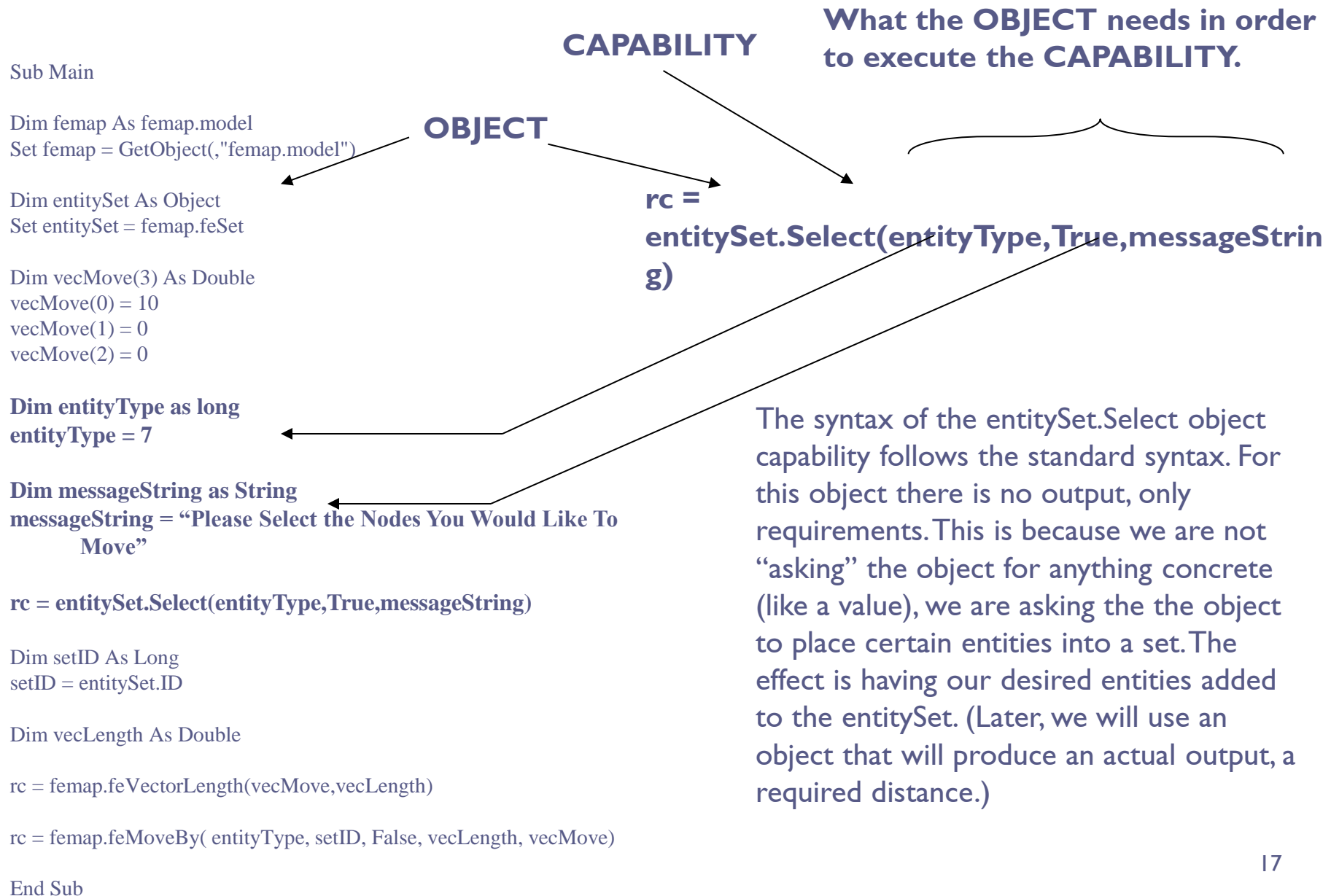
```
rc = femap.feMoveBy( entityType, setID, False, vecLength, vecMove)
```

End Sub

Next, we will declare a string, and give it a value.

Now what we want to do is collect from the user, what nodes they would like moved.

The ***Set*** object has a handy capability that allows us to do this called ***select***.



Entity types

Each entity in the FEMAP API is identified by a name and a number. The entity can be referred to by either. In the preceding piece of code where I refer to the node entity as the number 7, I could also have referred to it as FT_NODE. Either way the API will know to which entity type you are referring.

Entity Type	Numeric Value	Entity Type	Numeric Value
FT_POINT	3	FT_OUT_CASE	28
FT_CURVE	4	FT_OUT_DIR	29
FT_SURFACE	5	FT_OUT_DATA	30
FT_VOLUME	6	FT_REPORT	31
FT_NODE	7	FT_BOUNDARY	32
FT_ELEM	8	FT_LAYER	33
FT_CSYS	9	FT_MATL_TABLE	34
FT_MATL	10	FT_FUNCTION_DIR	35
FT_PROP	11	FT_FUNCTION_TABLE	36
FT_LOAD_DIR	12	FT_SOLID	39
FT_SURF_LOAD	13	FT_COLOR	40
FT_GEOM_LOAD	14	FT_OUT_CSYS	41
FT_NTHERM_LOAD	15	FT_CONTACT	58
FT_ETHERM_LOAD	16	FT_GRTYPE	59
FT_BC_DIR	17	FT_AMGR_DIR	60
FT_BCO	18	FT_TMG_BCO	112
FT_BCO_GEOM	19	FT_TMG_CONTROL	113
FT_BEQ	20	FT_TMG_INTEGER	114
FT_TEXT	21	FT_TMG_REAL	115
FT_VIEW	22	FT_TMG_OPTION	116
FT_GROUP	24		
FT_VAR	27		

Data types

Visual Basic requires the programmer to declare all variables before they are used as well as what type of data they will be. The six data types are shown below. WinWrap corresponds to the Visual Basic 6 data types.

API Definition in the Manual	Description	From Basic		From C++
		Visual Basic 6	Visual Basic .NET	
BOOL	Single byte, True/False value	Boolean	Boolean	Unsigned Char
INT2	2-byte integer	Integer	Integer	short
INT4	4-byte integer	Long	Integer	long, int
REAL4	4-byte real	Single	Single	float
REAL8	8-byte real	Double	Double	double
STRING	character string, null terminated	String	String	char[..]

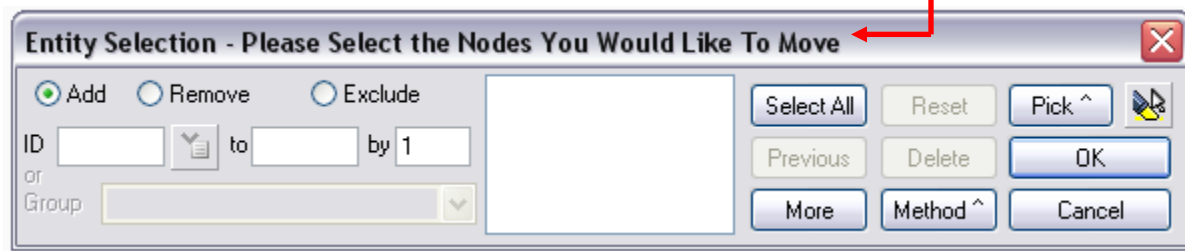
API.pdf

Select (entityTYPE, clear, title)	
Description:	
This function displays a standard selection dialog box to allow a user to choose entities of a specific type and create a selection set.	
Input:	
INT4 entityTYPE	Type of entity to select. For more information, see Section 3.3.5, "Entity Types".
BOOL clear	If True, the set is cleared prior to selection, and only the selected entities will be in the set. If False, previously selected entities will be displayed in the dialog box for editing, or for combining with new selections.
char *title	A text string that will be added to the title bar of the dialog box to give the user more information about what is being selected.
Output:	
None	
Return Code:	
FE_CANCEL	The user cancelled the selection.
FE_NOT_EXIST	No entities of the selected type exist. None were selected and the dialog box was not displayed. If clear=True, the selection set will be empty.
Remarks/Usage:	
After the dialog box is closed, the set contains the list of all IDs that were selected. If clear=False, then some of the entities could have been selected prior to the dialog being displayed. If the user hits "Cancel" to close the dialog, then the set contains whatever it contained before, unless clear=True, in which case it contains nothing.	

rc =
entitySet.Select(entityType,
True,messageString)

OBJECT **CAPABILITY** What the **OBJECT** needs to use the **CAPABILITY**.

rc = entitySet.Select(entityType,True,messageString)



A **Set** object is used to store a set of entities, i.e. a list of nodes. The **select** capability displays the above shown dialogue box so the user can select which nodes they are interested in. After the user selects these nodes, they are added to the set called **entitySet**.

In order to do this, the **Set** object needs:

- it needs to know what type of entity to ask for: **entityType**, which has already been set to 7; this number corresponds to the node entity,
- the **True** statement tells the object to clear the set of any old entities that may be in it,
- and a message to give the user: **messageString**

Return codes

Often statements like the following are found in API's:

```
rc = object.capability(requirements,output)
```

The **rc** stands for *return code*. After the object executes it's capability, it returns a code that corresponds to it's success in executing the capability. If the object is successful, a -1 is returned. If it is not successful, something else will be returned depending upon what went wrong. All the return codes are found in the table on the right.

FEMAP Return Codes

FE_OK	-1	FE_NOT_AVAILABLE	6
FE_FAIL	0	FE_TOO_SMALL	7
FE_CANCEL	2	FE_BAD_TYPE	8
FE_INVALID	3	FE_BAD_DATA	9
FE_NOT_EXIST	4	FE_NO_MEMORY	10
FE_SECURITY	5	FE_NO_FILENAME	16

Return
code

OBJECT

CAPABILITY

REQUIREMENTS

OUTPUT

`rc = app.feMeasureDistance(pt1, p2, dist)`

feMeasureDistance (p1, p2, dist)

Description:

This function measures the distance between two coordinates.

Input:

REAL8 p1[0..2]	The first coordinate location.
REAL8 p2[0..2]	The second coordinate location.

Output:

REAL8 *dist	The distance between the coordinate locations.
-------------	--

One more type of object syntax

```
Sub Main
```

```
Dim femap As femap.model  
Set femap = GetObject("femap.model")
```

```
Dim entitySet As Object  
Set entitySet = femap.feSet
```

```
Dim vecMove(3) As Double  
vecMove(0) = 10  
vecMove(1) = 0  
vecMove(2) = 0
```

```
Dim entityType as long  
entityType = 7
```

```
Dim messageString as String  
messageString = "Please Select the Nodes You Would Like To Move"
```

```
rc = entitySet.Select(entityType,True,messageString)
```

```
Dim setID As Long  
setID = entitySet.ID
```

```
Dim vecLength As Double
```

```
rc = femap.feVectorLength(vecMove,vecLength)
```

```
rc = femap.feMoveBy( entityType, setID, False, vecLength, vecMove)
```

```
End Sub
```

Certain object capabilities require no input and do not provide output in the conventional way.

Such is the case with the **object.ID** statement.

Instead this syntax returns the desired value to the variable on the left hand side of the equal sign. In this case setID will take on the ID number of the object entitySet. A single program can have multiple set objects defined, each containing their own data. Each of these sets would have a specific ID to differentiate them.

Retrieving the length of the move vector

Sub Main

```
Dim femap As femap.model  
Set femap = GetObject("femap.model")
```

```
Dim entitySet As Object  
Set entitySet = femap.feSet
```

```
Dim vecMove(3) As Double  
vecMove(0) = 10  
vecMove(1) = 0  
vecMove(2) = 0
```

```
Dim entityType as long  
entityType = 7
```

```
Dim messageString as String  
messageString = "Please Select the Nodes You Would Like To Move"
```

```
rc = entitySet.Select(entityType,True,messageString)
```

```
Dim setID As Long  
setID = entitySet.ID
```

```
Dim vecLength As Double
```

```
rc = femap.feVectorLength(vecMove,vecLength)
```

```
rc = femap.feMoveBy( entityType, setID, False, vecLength, vecMove)
```

```
End Sub
```

We will now use a capability of the **FEMAP Application Object** to find the magnitude of the nodal move we will be requesting.

```
rc = femap.feVectorLength(vecMove,vecLength)
```

↑
CAPABILITY

↑
INPUT

↑
OUTPUT

What we are asking of the **FEMAP Application Object** is for it to take our vector, called **vecMove**, and tell us how long it is. What the object gives us, is a new value for **vecLength**. If the operation is successful, **rc** will be given the value of **-1**.

Moving the nodes

Sub Main

```
Dim femap As femap.model  
Set femap = GetObject("femap.model")
```

```
Dim entitySet As Object  
Set entitySet = femap.feSet
```

```
Dim vecMove(3) As Double  
vecMove(0) = 10  
vecMove(1) = 0  
vecMove(2) = 0
```

```
Dim entityType as long  
entityType = 7
```

```
Dim messageString as String  
messageString = "Please Select the Nodes You Would Like To Move"
```

```
rc = entitySet.Select(entityType,True,messageString)
```

```
Dim setID As Long  
setID = entitySet.ID
```

```
Dim vecLength As Double
```

```
rc = femap.feVectorLength(vecMove,vecLength)
```

```
rc = femap.feMoveBy( entityType, setID, False, vecLength, vecMove)
```

End Sub

And last, but certainly not least, we will request that the ***FEMAP Application Object*** moves our nodes.

This capability, called ***feMoveBy***, has the following requirements:

- what type of entity it is moving,
- what set contains the ID's of the entities to move,
- whether or not this is a radial translation,
- the length of the translation,
- and a vector specifying the direction of the translation.

In conclusion

What is most interesting about the script we just explored, is that it only does one thing: it moves the nodes. Everything else found in script exists only to provide the last command with the information it needs to make the move. This is fairly common. Often much of the API script is devoted to retrieving things from the database, interpreting them, changing them, and then finally inserting them back in.

In our case, we retrieved the node numbers of that were to be moved, organized them into a set, and then requested that the ***FEMAP Application Object*** move them.

The previous example is a simple one that uses very little logic. There are no **for** or **while** loops and no **if else** statements, but all of the standard logic statements are available and are used all the time. Anyone with basic programming skills should be able to utilize them as they would in any other language.

You should now understand the basics needed to read and understand basic API's. The only way to become a PRO at writing them is to sit down and do it. In no time you will find that the structure and capabilities are extremely powerful. You will also find that you will never again need to scratch your head and say, "I wish the FEMAP programmers would have included this feature."