SolidWorks 2006 API 1

Human Centered CAD Laboratory

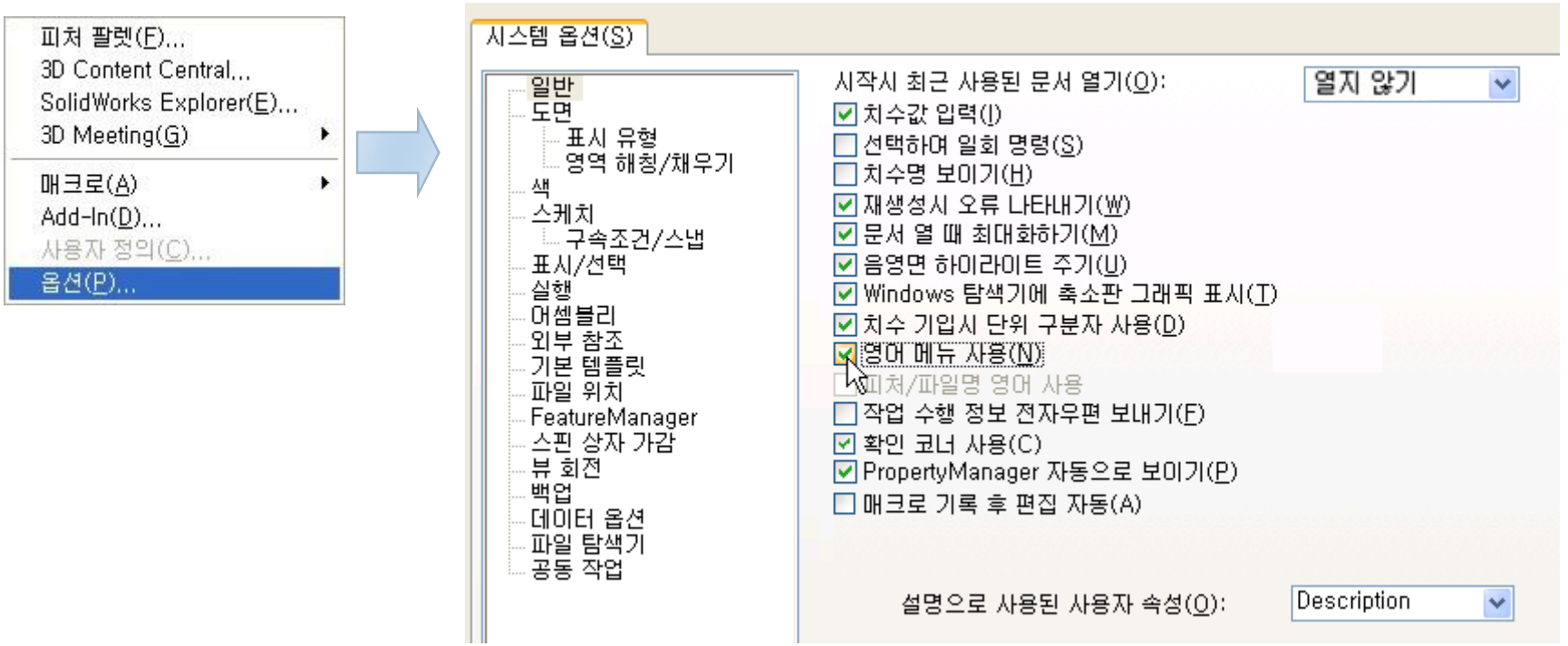
About

- The Goal
  - The goal of this course is to introduce you the SolidWorks 2006 Application Programming Interface(API). Using the SolidWorks API is a great way to automate routines. Just about everything a user leverages through the user interface is available in the API.

- Course Design Philosophy
  - This course is designed around a process or task based approach to training. Rather than focus on individual features and functions, a process-based training course emphasizes the process and procedures you follow to complete a particular task.

Getting Started

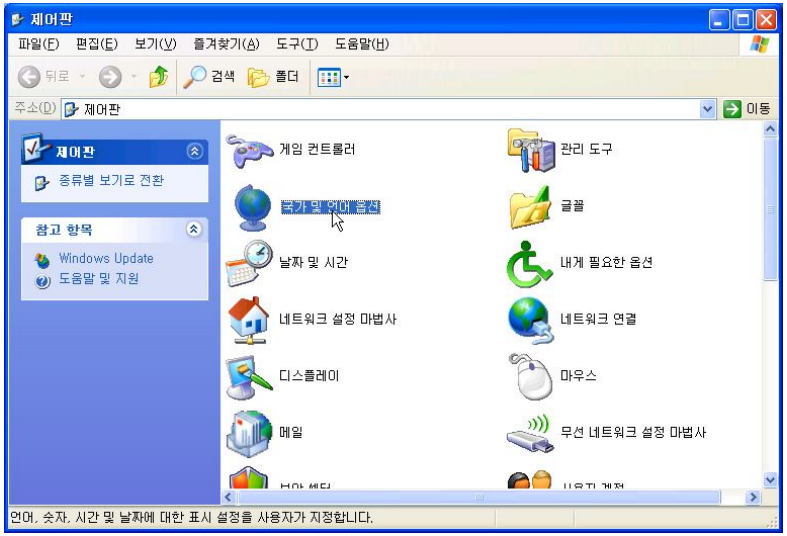
- For foreign students
  - SolidWorks can be run in English mode.
  - How to change the SolidWorks language Option
    - '도구'->'옵션'->'시스템옵션'->'일반'->'영어사용'

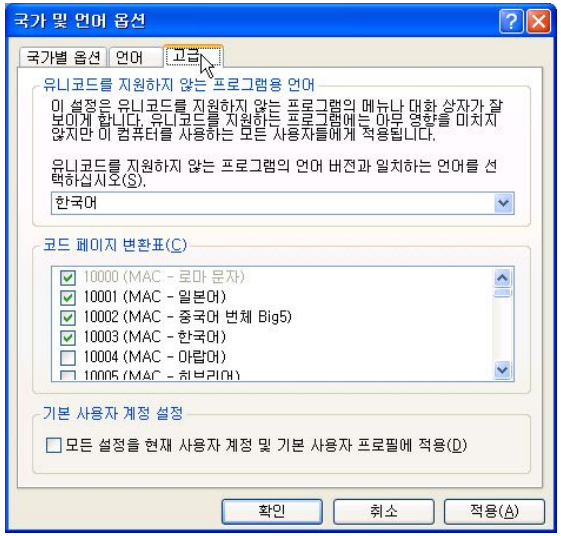


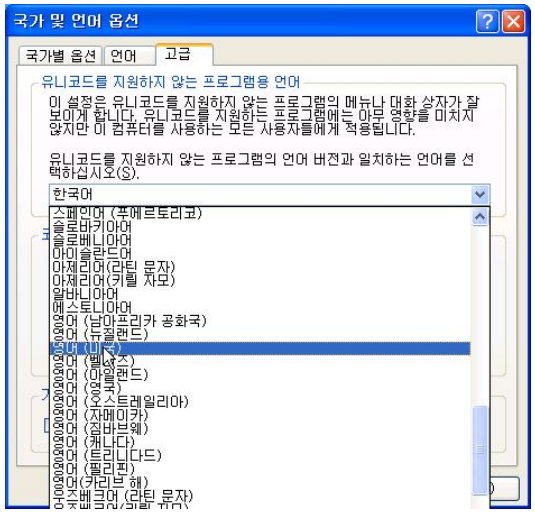
Getting Started – cont'

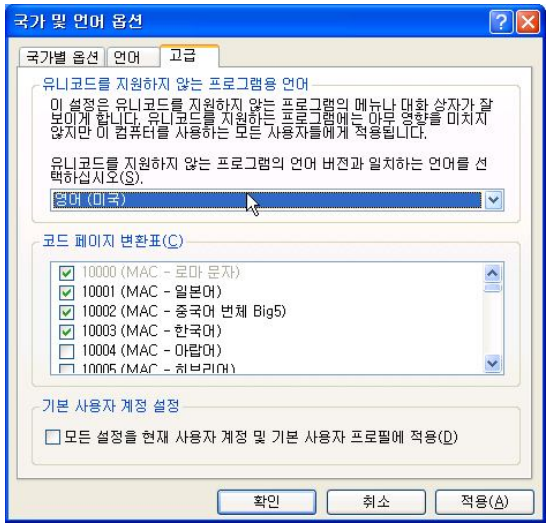
- For foreign students
  - The VISUAL BASIC .NET Editor will be seen in Korean mode on Microsoft Windows XP Korean edition.
  - You can solve this problem by changing some system language options of Windows XP.
  - First, click '시작' button in the lower-left corner of Windows XP, then select '제어판'.
  - Click '국가 및 언어 옵션' , then select '고급' tap.
  - In '유니코드를 지원하지 않는 프로그램용 언어' , change the language from '한국어' to '영어(미국)'.
  - In next page, the entire process will be shown.

Getting Started – cont’



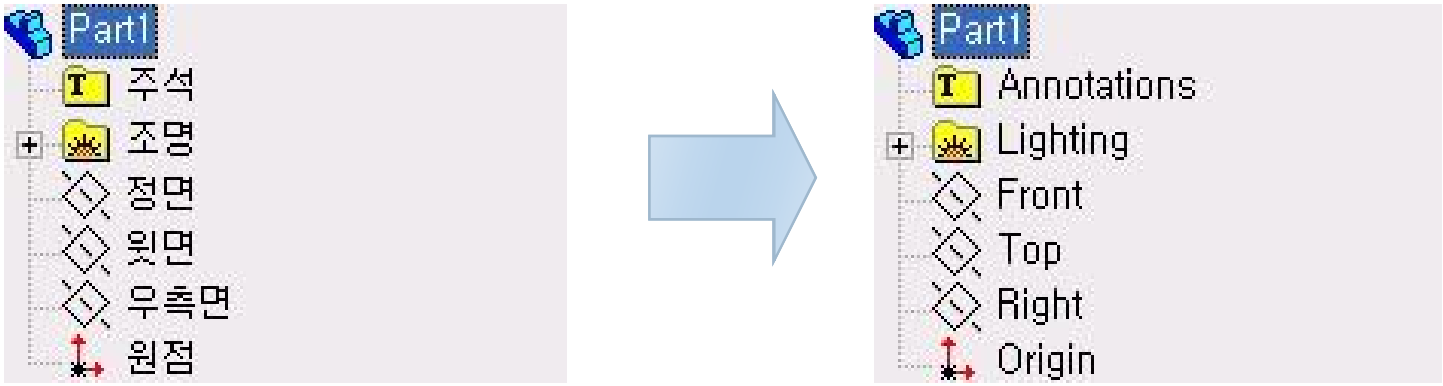






Getting Started – cont'

- Cautions
  - If you changed the Windows non-unicode program language option, you should pay attention to the names of objects in SolidWorks.
  - If you use SolidWorks in English Mode, you had better to modify objects' names in English.
  - You must change followings in English if you changed the non-unicode program language option.

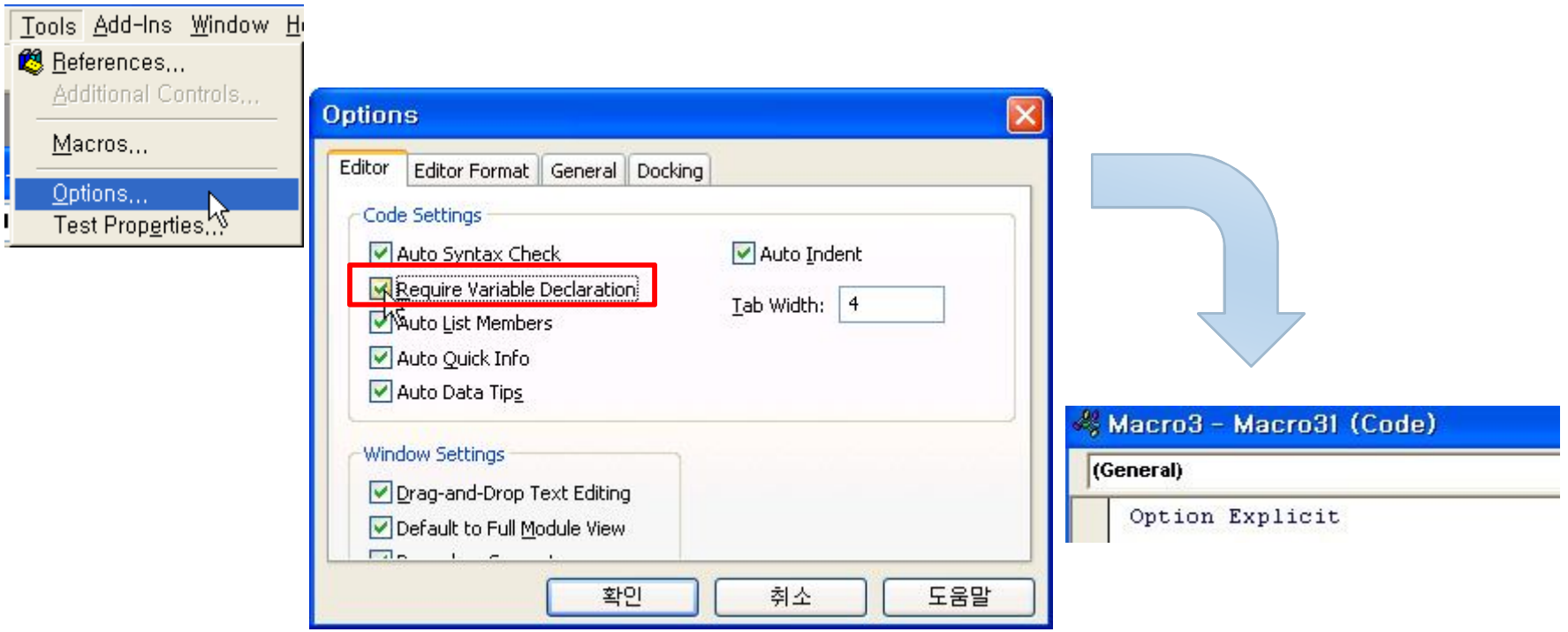


Getting Started – cont'

- A few things to note
  - File types
    - In Solidworks , the macro files we will create are of type:
      SW VBA Macros (*.swp)
    - In Visual Basic .NET, these files may be necessary:
      Project Files (*. Vbp), Class Files (*. cls),
      Resource Files(*.res), Dynamic-Link Library(*.dll)
  - Option Explicit
    - It is strongly recommended that you use the Option Explicit statement in all Visual Basic development. This forces declaration of all variables before use. By doing this, Visual Basic allows you to use error and type checking when debugging applications.

Getting Started – cont'

- A few things to note
  - How to set the Option Explicit function in VB Editor
    - 'Tools' -> 'Options' -> 'Editor' -> 'Code Settings' -> 'Require variable Declaration'



Getting Started – cont'

- Variables
  - Variables are used to store temporary values during the execution of an application. Variables are made up of two parts : *Name* and *Data Type*. You declare a variable with the *Dim* statement, supplying a name for the variable:

    Dim variablename [As data type]

    The optional, *As data type*, clause in the Dim statement allows you to define the data type or object type of the variable. Examples are:

    - Dim swApp AS object ‘Generic object
    - Dim swApp AS sldworks.sldworks ‘Specific object
    - Dim dvarArray(2) AS Double ‘Set of 3 doubles

Getting Started – cont'

- Solidworks Constants – swconst.bas
  - For Solidworks API development, you should always add the Solidworks constant file to each project or macro. In visual basic, select **Project**, **Add Module**, and under the **Existing** tab, browse to the location for this file(it is installed Solidworks):

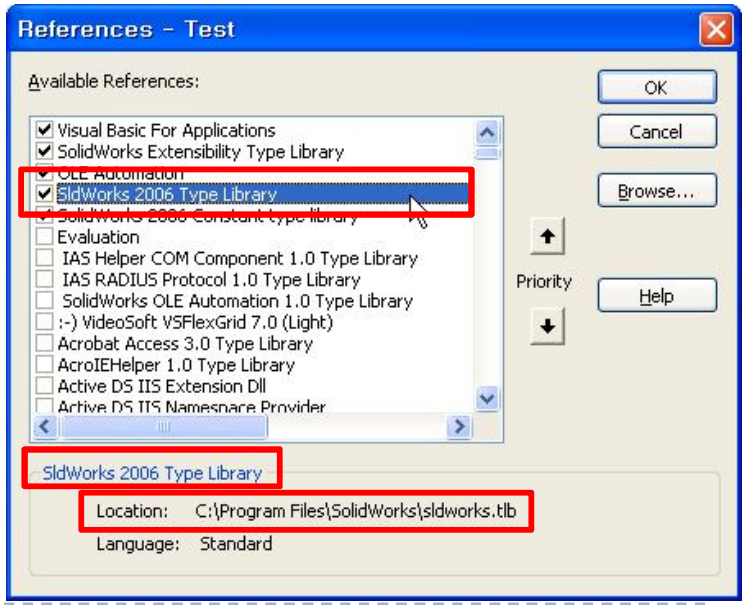
    <install directory>\Solidworks\samples\appcomm\swconst.bas

    This file provides definitions to use with Solidworks API functions.

    When you make your macro file by macro-recording method which is used in this course, you don't need to mind this work.

Getting Started – cont'

- SolidWorks 2006 Type Library
  - Visual Basic programs are enhanced by creating objects with specific Type Library interface types, rather than the generic object type. In order to do leverage this, we add a reference to the SolidWorks 2006 Type Library. As a convenience, the macro recorder will do this for you by default.
  - To make sure the reference is made, in VBA, click **Tools, References** and the option for **SolidWorks 2006 Type Library** should be checked.



Getting Started – cont'

- SolidWorks 2006 Type Library(cont')
  - If it is not checked, browse to its location (installed with SolidWorks):

    <install directory>\SolidWorks\sldworks.tlb
  - This file contains the exposed objects, properties, and methods that are available for Solid Works automation. When the type library is referenced, a drop down list after the dot (.) separator (behavior known as IntelliSense) will display SolidWorks objects, properties and methods that the programmer can use .

Getting Started – cont'

- Early vs. Late Binding
  - Binding is simply a verification process that Visual Basic uses to search through an object-type library (in our case, the SolidWorks 2006 Type Library) make sure an object exists, and that any properties or methods used with it are correctly specified. There are two types of binding:
  - Late Binding

```
Dim swApp As Object
Dim swModel As Object

Sub main()
   Set swApp = Application.SldWorks
   Set swModel = swApp.ActiveDoc
End Sub
```

  - Early Binding

```
Dim swApp As SldWorks.SldWorks
Dim swModel As SldWorks.ModelDoc2

Sub main()
   Set swApp = Application.SldWorks
   Set swModel = swApp.ActiveDoc
End Sub
```

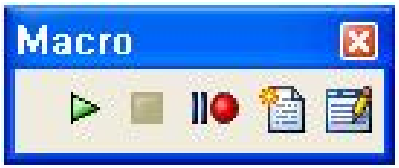
  - It is best to use early binding because it makes your code more efficient and your application faster!

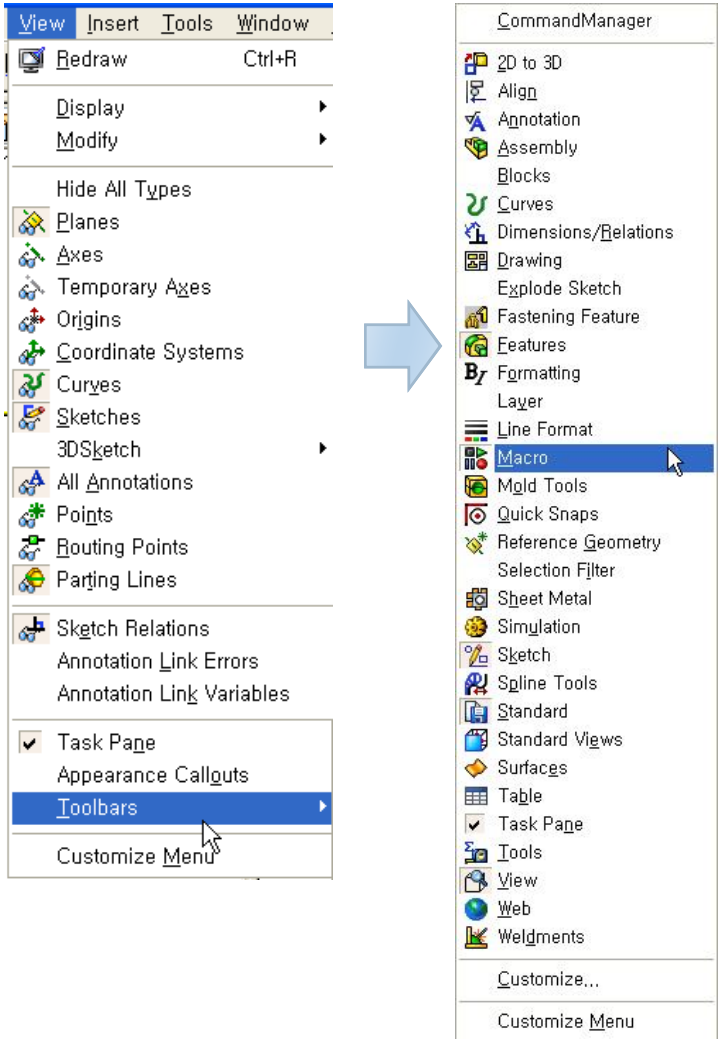
Using the Macro Recorder

- Macro Recording
  - You can record operations performed with the SolidWorks user interface and replay them using SolidWorks macros. A macro contains calls to the Application Programming Interface(API) that are equivalent to operations performed in the user interface. A macro can record mouse clicks, menu choices and keystrokes.
- Macro Toolbar
  - The macro toolbar contains shortcuts to the macro recording commands. You can also access these commands from the Tools, Macro menu.

Using the Macro Recorder – cont'

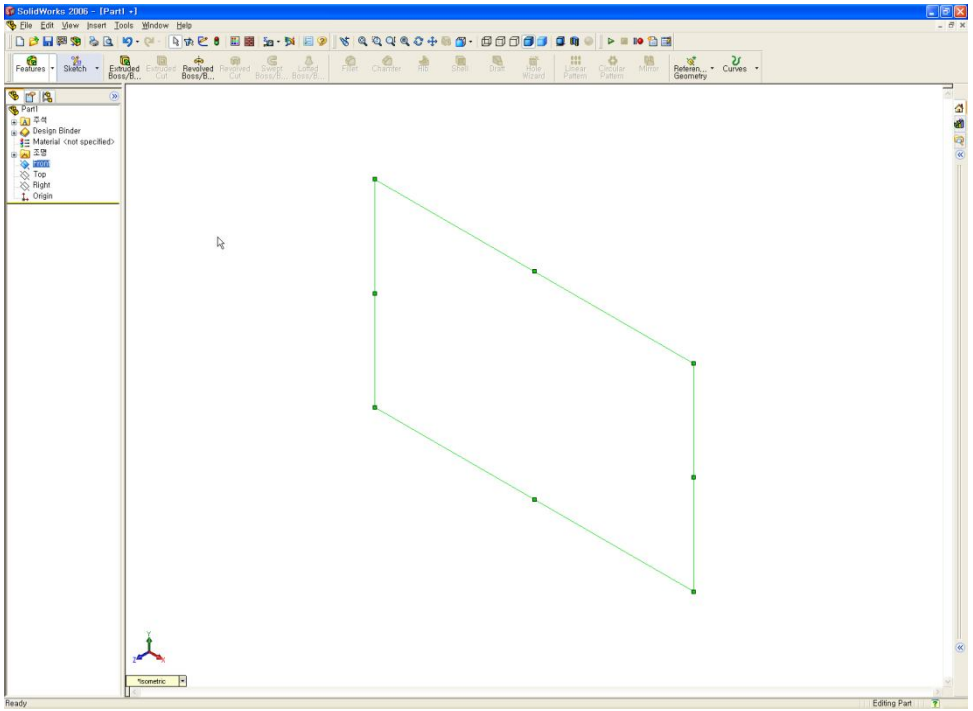
- Macro Toolbar(cont')
  - By default, the Macro toolbar is turned off. To create and use your macros, it is best to view and dock the macro toolbar at the top of the SolidWorks Window. From the **View** menu, select **Toolbars**, **Macro**.





Using the Macro Recorder – cont'

- First try to make an own macro
  1. Start Solidworks and create new part.
  2. View Macro toolbar and click 'Record' button.
  3. Select the Front plane.



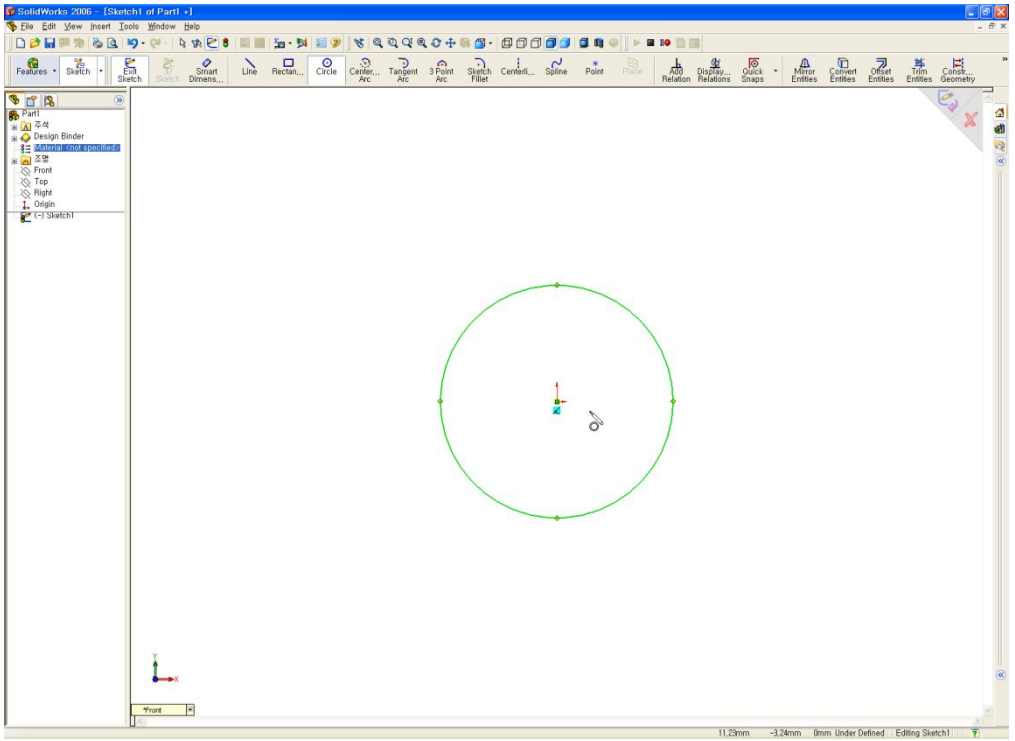
Using the Macro Recorder – cont’

- First try to make an own macro(cont')
  4. Click Sketch 

and click Circle 

.

     - Use an approximate radius of 40 millimeters, then enter the exact value of 40mm in the PropertyManager.



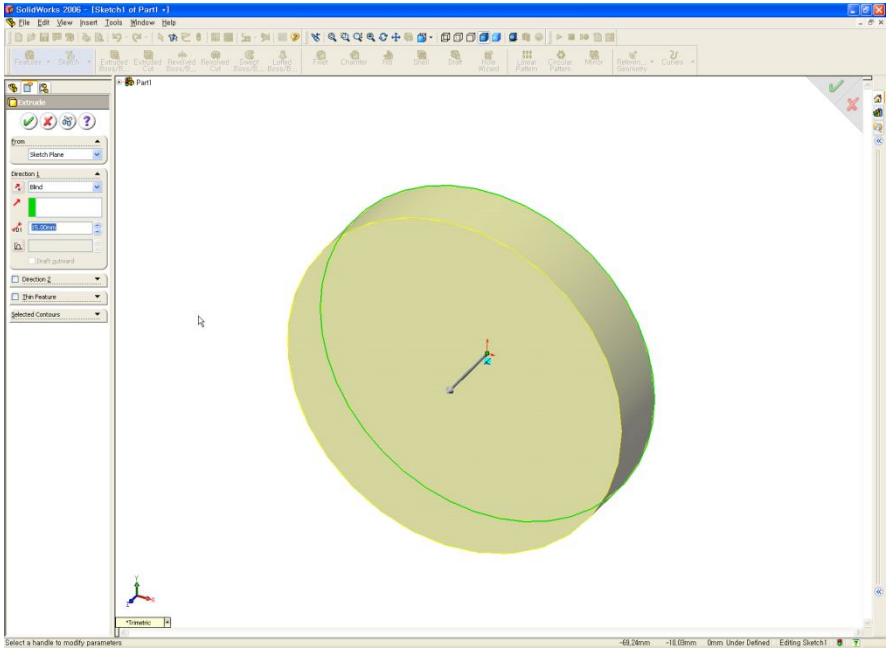
Using the Macro Recorder – cont'

- First try to make an own macro(cont')

5. Click Extruded Boss/Base 

.

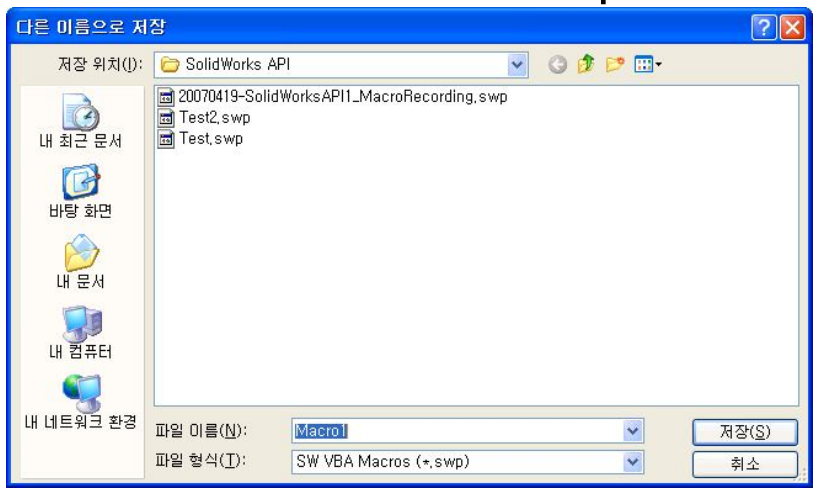
- Drag the extrude path approximately 15 mm, then enter the exact value of 15mm in the PropertyManager. Click OK.

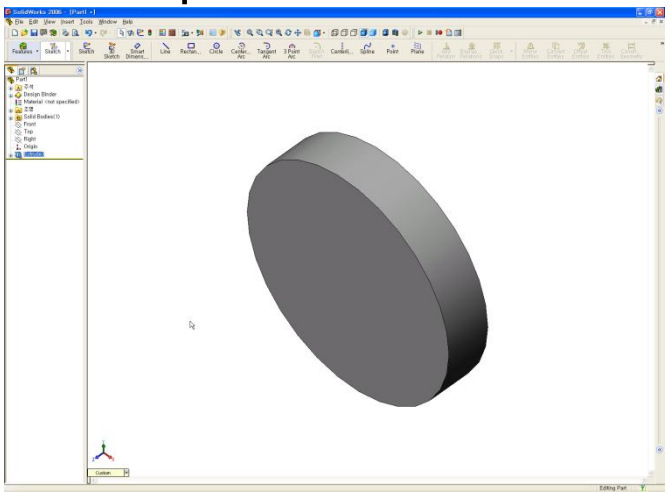


6. Click 'Stop' 

Using the Macro Recorder – cont’

- First try to make an own macro(cont')
  7. ‘Save’ macro.
     - In the Save As dialog, save the macro as Macro1.swp.
  8. Delete all features.
     - Remove the extruded base and sketch created previously.
  9. Click ‘Play’  .
     - Select Macro1.swp from the previous step.



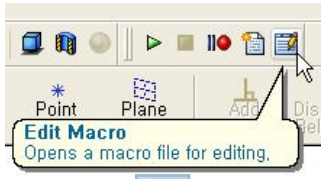


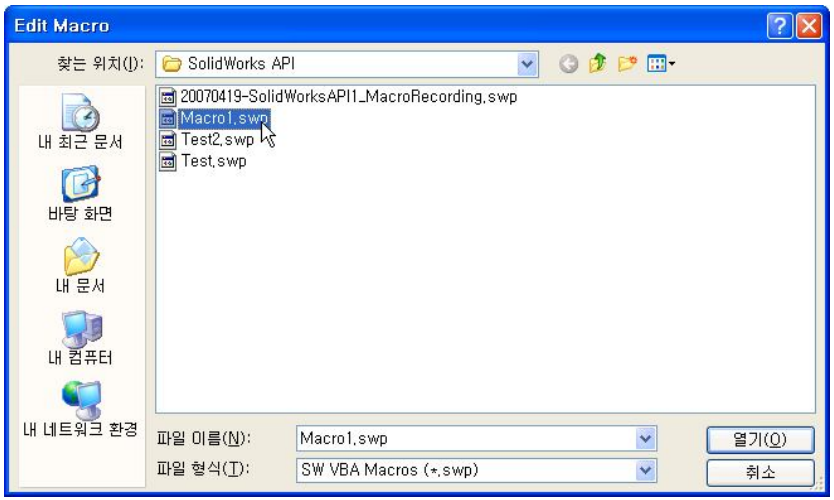
Using the Macro Recorder – cont'

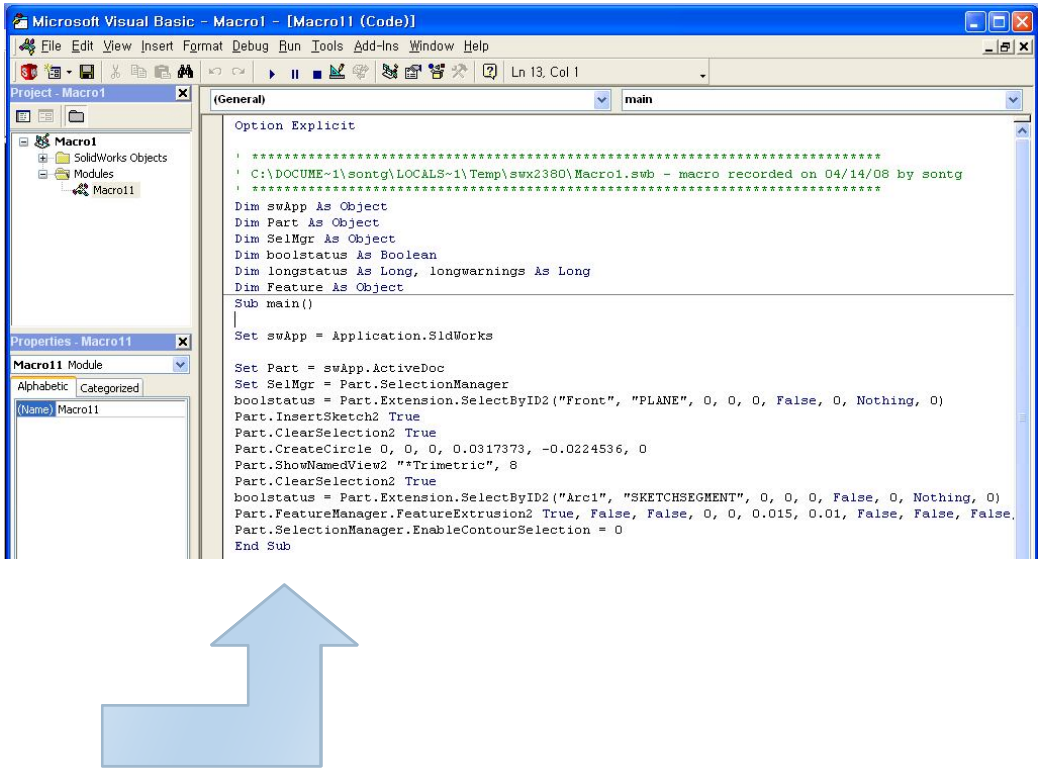
- Automation Review
  - Click1 : Select a plane.
  - Click2 : Insert sketch command.
  - Click3 : Create circle command.
  - Click4 : Center of circle
  - Click5 : Approximate 40mm radius of circle.
  - Keyboard Entry1 – Exact radius : 40 mm.
  - Click6 : OK button
  - Click7 : Extruded boss/base function
  - Click8 : Approximate 15mm depth of extrusion
  - Keyboard Entry2 – Exact depth : 15mm.
  - Click9 : OK button.

Using the Macro Recorder – cont'

- Visual Basic for Applications editor
  - Click the 'Edit' button from the Macro toolbar.
  - Select Macro1.swp.







Using the Macro Recorder – cont'

- Understanding How Macro Code Works
  - Variable Declaration
    - The macro recorder declares(or dimensions) a number of variables by default. You can comment out or delete variables not utilized in the entry point procedure.

```
Option Explicit
'**********************************
'* Macro
'**********************************
Dim swApp As Object
Dim Part As Object
Dim SelMgr As Object
Dim boolstatus As Boolean
Dim longstatus As Long, longwarnings As Long
Dim Feature As Object
```

Using the Macro Recorder – cont'

- Understanding How Macro Code Works
  - Entry Point Procedure
    - This is the beginning of our functionality. Every macro must establish an entry point procedure.

```
Sub main()
```

  - SolidWorks Application Object
    - This line of code will start an instance of SolidWorks or connect to a running instance of SolidWorks. Without it, your program will not run.

```
Set swApp = Application.SldWorks
```

  - SolidWorks Documentation Object
    - In order for us to work within the application, top-level document objects are accessed and made active. This allows the ability to program document specific functionality.

```
Set Part = swApp.ActiveDoc
```

Using the Macro Recorder – cont'

- Understanding How Macro Code Works
  - SolidWorks API Calls
    - An API call allows the macro to perform certain tasks. This is where we see our recorded steps taking shape:
      - Selecting an plane / Inserting a sketch
      - Creating a circle / Extruding a feature

```
Set SelMgr = Part.SelectionManager
boolstatus = Part.Extension.SelectByID2("Front", "PLANE", 0, 0, 0, False, 0, Nothing, 0)
Part.InsertSketch2 True
Part.ClearSelection2 True
Part.CreateCircle 0, 0, 0, 0.0317373, -0.0224536, 0
Part.ShowNamedView2 "*Trimetric", 8
Part.ClearSelection2 True
boolstatus = Part.Extension.SelectByID2("Arc1", "SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
Part.FeatureManager.FeatureExtrusion2 True, False, False, 0, 0, 0.015, 0.01, False, False, False,
False, 0.01745329251994, 0.01745329251994, False, False, False, False, 1, 1, 1, 0, 0, False
Part.SelectionManager.EnableContourSelection = 0
```

  - Procedure End

```
End Sub
```

Using the Macro Recorder – cont'

- Understanding How to Call Methods

1. The SldWorks object(declared *swApp* by default), the highest level object in the SolidWorks API, is accessed by implementing the following lines :

```
Dim swApp As Object
Set swApp = Application.SldWorks
```

2. The document object, PartDoc(Part) is accessed by implementing the following lines which calls a property on the application object:

```
Dim Part As Object
Set Part = swApp.ActiveDoc
```

Using the Macro Recorder – cont'

- Understanding How to Call Methods(cont')

3. Once the application and document objects are set, events, properties and methods on those objects are called. To access these functions the object name is written first, separated by a period, and followed by the full name of the API call:

```
Part.InsertSketch2 True
```

Some APIs require additional parameters :

```
Part.CreateCircle 0, 0, 0, 0.0317373,
-0.0224536, 0
```

Using the Macro Recorder – cont'

- Understanding How to Call Methods(cont')
  - Some APIs require additional objects :

```
Part.FeatureManager.FeatureExtrusion2 True, False,
False, 0, 0, 0.015, 0.01, False, False, False, False, 0.01
745329251994, 0.01745329251994, False, False, False,
False, 1, 1, 1, 0, 0, False
```

  - Some APIs utilize return values (called *retval* in the API help file) :

```
Dim boolstatus As Boolean
boolstatus = Part.Extension.SelectByID2("Arc1",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
```

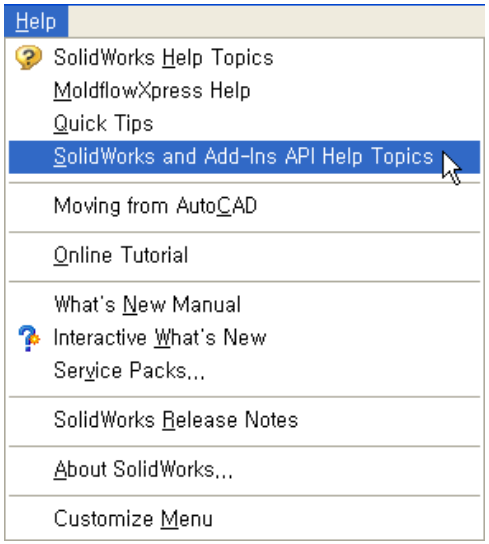
Using the Macro Recorder – cont'

- Passing Parameters
  - FeatureManager::FeatureExtrusion

```
pFeat = FeatureManager.FeatureExtrusion2 ( sd, flip, dir, t1, t2,
d1, d2, dchk1, dchk2, ddir1, ddir2, dang1, dang2, offsetReverse1,
offsetReverse2, translateSurface1, translateSurface2, merge,
useFeatScope, useAutoSelect, t0, startOffset, flipStartOffset )
```

  - Parameter details:
    - Let's see SolidWorks 2006 API Help
    - Click **Help -> SolidWorks API Help Topics**
    - The file is also located in:

      <install directory>\SolidWorks\lang\apihelp.chm



Using the Macro Recorder – cont'

- Cleaning Up native macro code
  - Lines in red can be deleted(unnecessary).
  - Lines in blue can be modified(better API calls exist).

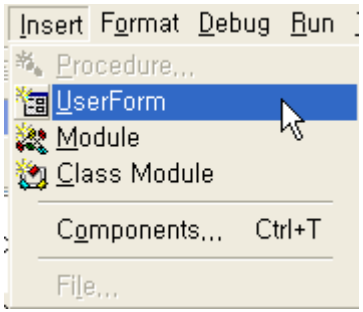
```
Option Explicit
' ************************************************************************
' C:\DOCUME~1\sontg\LOCALS~1\Temp\swx2380\Macro1.swb - macro recorded on 04/14/08 by HCCL
' ************************************************************************
Dim swApp As Object
Dim Part As Object
Dim SelMgr As Object
Dim boolstatus As Boolean
Dim longstatus As Long, longwarnings As Long
Dim Feature As Object
Sub main()
        Set swApp = Application.SldWorks
        Set Part = swApp.ActiveDoc
        Set SelMgr = Part.SelectionManager
        boolstatus = Part.Extension.SelectByID2("Front", "PLANE", 0, 0, 0, False, 0, Nothing, 0)
        Part.InsertSketch2 True
        Part.ClearSelection2 True
        Part.CreateCircle 0, 0, 0, 0.0317373, -0.0224536, 0
        Part.ShowNamedView2 "*Trimetric", 8
        Part.ClearSelection2 True
        boolstatus = Part.Extension.SelectByID2("Arc1", "SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
        Part.FeatureManager.FeatureExtrusion2 True, False, False, 0, 0, 0.015, 0.01, False, False, False, False, 0.01745329251994,
        0.01745329251994, False, False, False, False, 1, 1, 1, 0, 0, False
        Part.SelectionManager.EnableContourSelection = 0
End Sub
```

Using the Macro Recorder – cont'

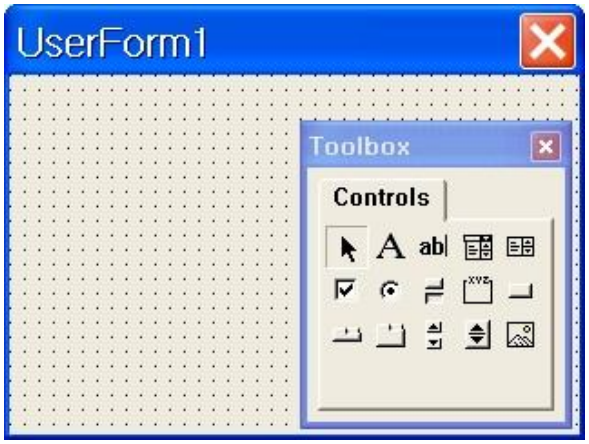
- Cleaning Up native macro code(cont')
  - Never be satisfied with results the SolidWorks macro recorder returns. Always look to improve and clean up your code. We can use the online API help file in SolidWorks to search out new, improved or just alternative API calls for our needs.
  - There is another way to create a circle. *CreateCircle* requires six parameters: *xc, yc, zc, xp, yp, zp*. This method creates a circle based on a center point and a point on the circle. That is not exactly what we performed with the user interface.
  - We can replace this function by *CreateCircleByRadius2*, and only requires these parameters: *xc, yc, zc, radius*.

Using the Macro Recorder – cont'

- Adding Forms to a Macro
  - One of the easiest ways to get someone launch your code is to add a form (called userforms in VBA) to the macro.



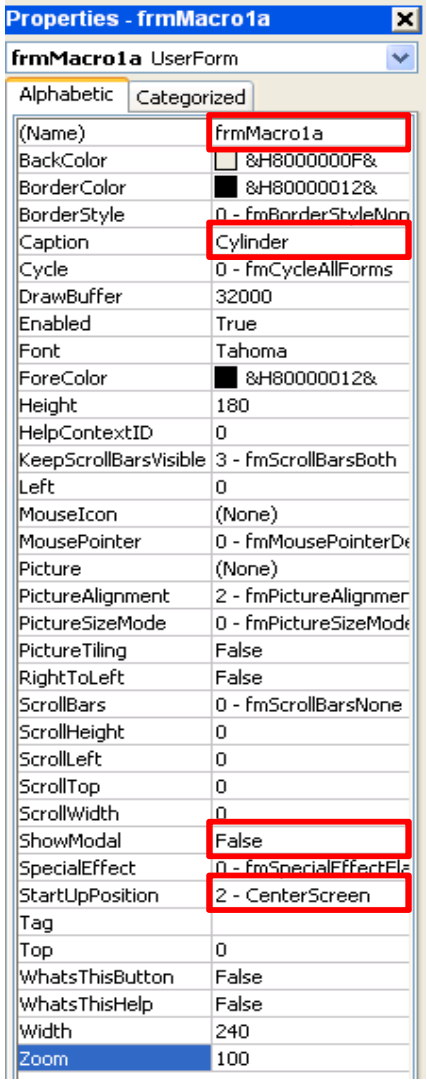
- Where to Find it
  - In VBA, click **Insert, UserForm**.
  - In VBA, within the Project Explorer window, right-click the macro and select **Insert, UserForm**.
  - The VBA Toolbox is displayed along with the form by default. If it does not appear click **View, Toolbox**.



Using the Macro Recorder – cont'

- Add form to macro.
  - In VBA, click Insert, UserForm.
- Edit form properties.
  - With the form highlighted, enter the following property values:

UserForm1:
(Name): frmMacro1a
Caption: Cylinders
ShowModal: False
Startup Position: 2 - centerScreen



Using the Macro Recorder – cont'

- Add controls to form.
  - From the toolbox, drag and drop one label and five command buttons onto the form. Use the following as a guide for each control:

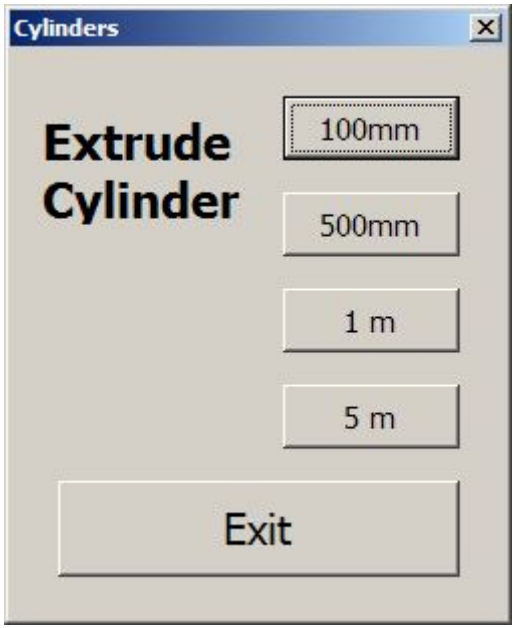
CommandButton1:
(Name): cmd100mm
Caption: 100 mm

CommandButton2:
(Name): cmd500mm
Caption: 500 mm

CommandButton3:
(Name): cmd1m
Caption: 1 m

CommandButton4:
(Name): cmd5m
Caption: 5 m

CommandButton5:
(Name): cmdExit
Caption: Exit



Using the Macro Recorder – cont'

- Add click event to each button.
  - Double-click each button control to set up an empty click event procedure. The VBA editor automatically adds the necessary entry point and end to each event.

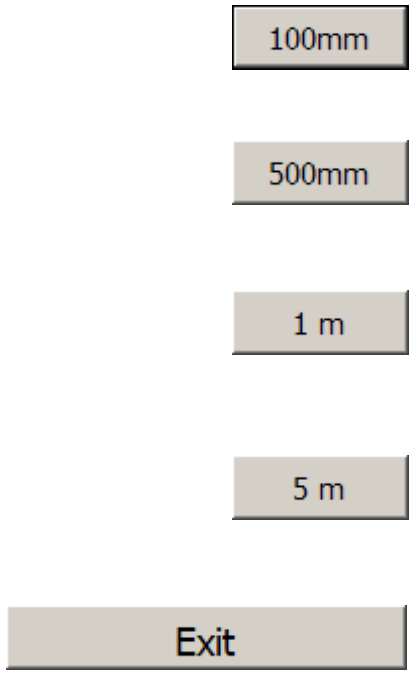
```
Private sub cmd100mm_Click()
End sub

Private sub cmd500mm_Click()
End sub

Private sub cmd1m_Click()
End sub

Private sub cmd5m_click()
End sub

Private sub cmdExit Click()
End sub
```

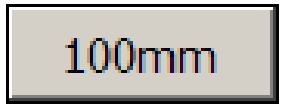


Using the Macro Recorder – cont'

- Move code from module to buttons
  - At this point the entire macro file should contain one module and one form. We want to keep both, but move code to different locations.
  - Cut everything within the module leaving only an empty entry point procedure (*Sub main .... End Sub*). Paste the code in the click event for each command button (except the Exit button).
  - Look closely at the code below. Only one parameter (shown in **bold**) is changed to account for the different extrusion depths of each button.

Using the Macro Recorder – cont'

- Move code from module to buttons(cont')




```
Option Explicit

Private Sub cmd100mm_Click()
    Dim swApp As Object
    Dim Part As Object
    Dim boolstatus As Boolean

    'connect to solidworks
    Set swApp = Application.SldWorks
    Set Part = swApp.ActiveDoc

    'Create a cylinder on the front plane
    boolstatus = Part.Extension.SelectByID("Front", "PLANE", 0, _
    0, 0, False, 0, Nothing)
    Part.InsertSketch2 True
    Part.CreateCircleByRadius2 0, 0, 0, 0.04
    Part.FeatureManager.FeatureExtrusion2 True, False, False, 0, 0, _
    0.1, 0.01, False, False, False, False, 0.01745329251994, 0.01745329251994, _
    False, False, False, False, 1, 1, 1, 0, 0, False
End Sub
```

Using the Macro Recorder – cont'

- Move code from module to buttons(cont')

500mm

```
Part.FeatureManager.FeatureExtrusion2 True, False, False, 0, 0, _
 0.5, 0.01, False, False, False, False, 0.01745329251994, 0.01745329251994, _
 False, False, False, False, 1, 1, 1, 0, 0, False
```

1 m

```
Part.FeatureManager.FeatureExtrusion2 True, False, False, 0, 0, _
 1.0, 0.01, False, False, False, False, 0.01745329251994, 0.01745329251994, _
 False, False, False, False, 1, 1, 1, 0, 0, False
```

5 m

```
Part.FeatureManager.FeatureExtrusion2 True, False, False, 0, 0, _
 5.0, 0.01, False, False, False, False, 0.01745329251994, 0.01745329251994, _
 False, False, False, False, 1, 1, 1, 0, 0, False
```

Program the Exit button to allow the user to terminate the macro.

Exit

```
Private Sub cmdExit_Click()
    End
End Sub
```

Using the Macro Recorder – cont'

- Add code to module.
  - Switch back to the module. In order for the macro to run in SolidWorks, the entry point procedure on the module needs to show the userform. Enter the following line of code:

```
Sub main()
   frmMacro1a.Show
End Sub
```

- Save and Run macro.
  - Save the macro. With SolidWorks open and a new part file created, run the macro either from the Macro toolbar or from the VBA editor.
- Exit macro.
  - Click the **Exit** button to end the macro and return to VBA..

Using the Macro Recorder – cont'

- Add second form.
  - Click **Insert, UserForm.** Enter the following property values (next page).
- Add user interaction controls to second form.
  - To capture input from a user, we add textbox controls that ask the user to specify the depth and diameter rather than hard-coding the values.
  - Drag and drop the necessary labels, text boxes and com mand buttons onto frmMacro1b using the followings as a guide for each control :

Using the Macro Recorder – cont'

- Continue…

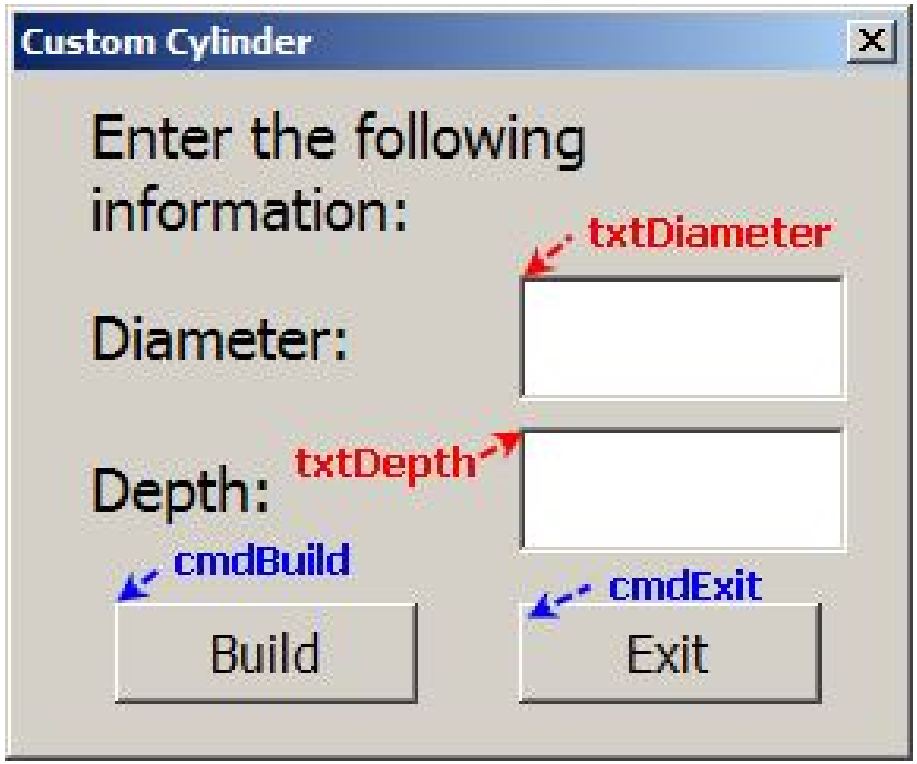
UserForm2:
(Name): frmMacro1b
Caption: Custom cylinder
Startup Postion: 2 - Centerscreen
ShowModal: False

TextBox1:
(Name): txtDiameter
Text: <leave blank>

TextBox2:
(Name): txtDepth
Text: <leave blank>

CommandButton1:
(Name): cmdBuild
Caption: Build

CommandButton2:
(Name): cmdExit
Caption: Exit



Using the Macro Recorder – cont'

- Add code to each button.
  - We want each string value in the text boxes to be converted to a double value for diameter and depth. Make a copy of the working code you already have from one of the buttons on frmMacro1a and paste into a click event for cmdBuild on frmMacro1b. Then make the adjustments to the code:


```
Option Explicit

Private Sub cmdBuild_Click()
    Dim swApp As Object
    Dim Part As Object
    Dim boolstatuS As Boolean
    Dim diameter As Double
    Dim depth As Double

    diameter = CDbl(txtDiameter.Text) / 1000
    depth = CDbl(txtDepth.Text) / 1000
    'Continue on next page …
```

Using the Macro Recorder – cont'

- Add code to each button.(cont')

Build

```
    'Connect to Solidworks
    Set swApp = Application.SldWorks
    Set Part = swApp.ActiveDoc

    'Create a cylinder on the front plane
    boolstatuS = Part.Extension.SelectByID("Front", "PLANE", 0, _
    0, 0, False, 0, Nothing)
    Part.InsertSketch2 True
    Part.CreateCircleByRadius2 0, 0, 0, diameter / 2
    Part.FeatureManager.FeatureExtrusion2 True, False, False, 0, 0, _
    depth, 0.01, False, False, False, False, 0.01745329251994, 0.01745329251994, _
    False, False, False, False, 1, 1, 1, 0, 0, False
End Sub
```

- Also remember to program cmdExit to terminate the macro

Exit

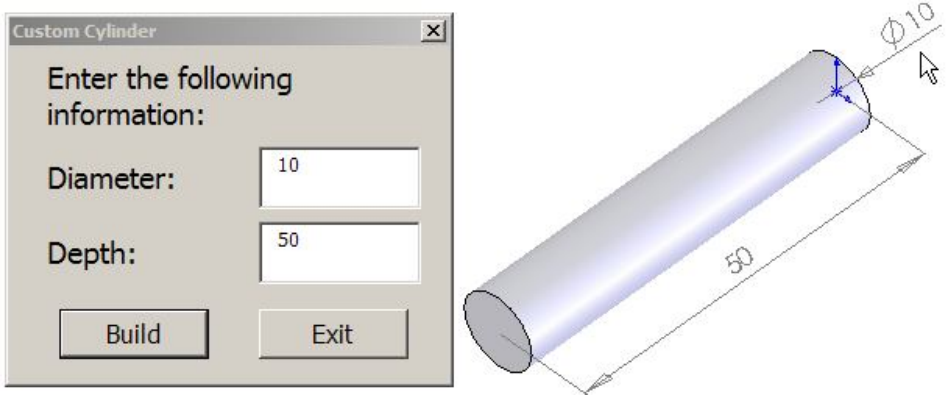
```
Private Sub cmdExit_Click()
    End
End Sub
```

Using the Macro Recorder – cont'

- Modify code of module
  - In order for macro to show userform frmMacro1b instead of frmMacro1a, the entry point procedure on the module needs to show the userform frmMacro1b.

```
Sub main()
    frmMacro1b.Show
End Sub
```

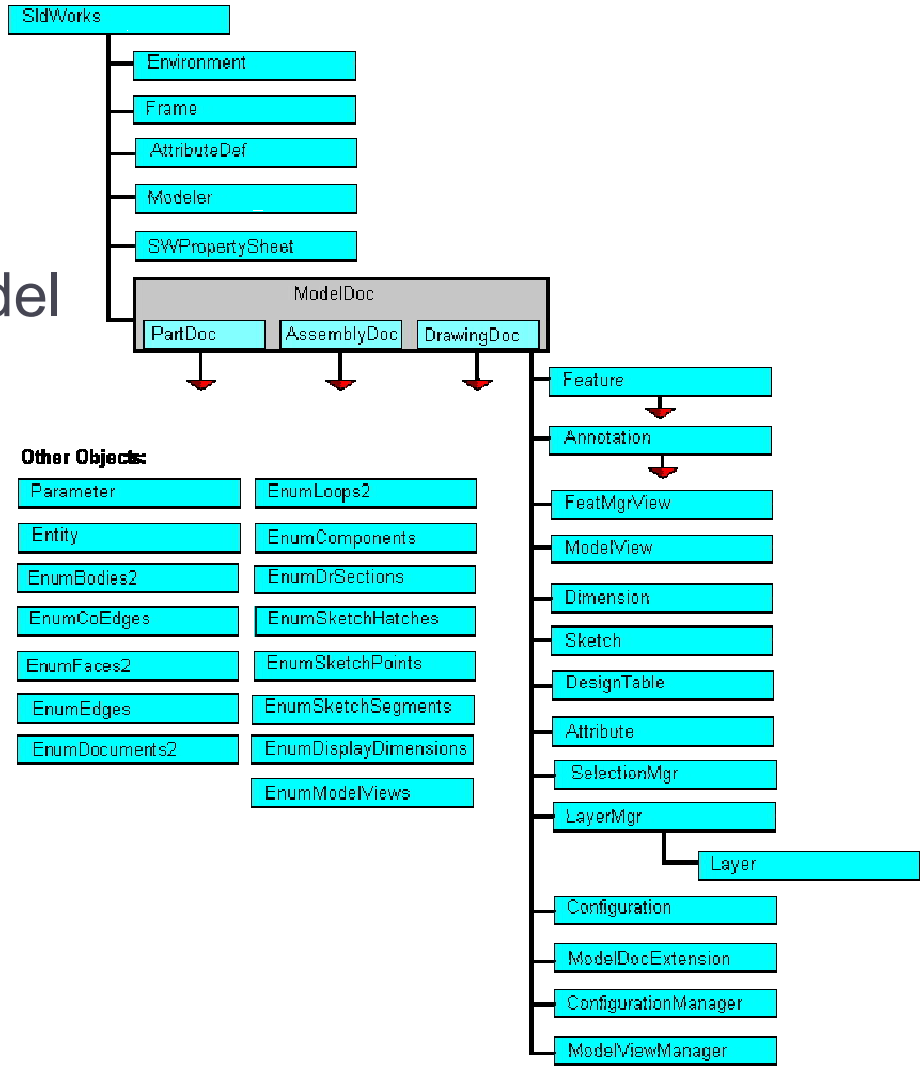
- Save and run macro



- Exit macro

SolidWorks API Object Model

- SolidWorks API Object
  - The following diagram is a general depiction of the SolidWorks API Object Model



SolidWorks API Object Model – cont'

- SldWorks Object(swApp)
  - The SldWorks object (declared swApp by default), the highest level object in SolidWorks, provides access to all other objects exposed in the API. It is also referred to as an interface that provides a general set of functions that allow application level operations. Use the following two lines of code to connect to the SldWorks object:

```
Dim swApp As SldWorks.SldWorks
Set swApp = Application.SldWorks
```

  - The variable, swApp, is arbitrary and is declared using a generic type, Object, rather than a specific type, SldWorks. SldWorks. This is discussed in Early vs. Late Binding.
  - As we progress, it is better practice to use early binding tech niques when programming with the SolidWorks API.

SolidWorks API Object Model – cont'

- SldWorks Methods and Properties

.NewDocument (TemplateName, Papersize, width, Height)
.RevisionNumber ( )
.DisplayStatusBar (Onoff)
.SolidworksExplorer ( )
.OpenDoc6 (FileName, Type, Options, Config, Errors, Warnings)
.LoadFile3 (FileName, ArgumentString, ImportData)
.CreateNewWindow ( )
.Arrangewindows (Style)
.ActiveDoc ( )
.ActivateDoc2 (Name, silent, Errors)
.CloseDoc (Name)
.QuitDoc (Name)
.ExitApp ( )
.DocumentVisible (visible, Type)
.SendMsgToUser (Message, Icon, Buttons)

SolidWorks API Object Model – cont'

- ModelDoc2 Object
  - The ModelDoc2 object *(swModel)* contains functions that are common to all three document types: parts, assemblies and drawings. Use one of the following methods below on the SldWorks object (also known *as accessors)* to connect to the ModelDoc2 object:

```
'option 1
Dim swModel As sldworks.ModelDoc2
set swModel = swApp.ActiveDoc
'option 2
Dim swModel As Sldworks.ModelDoc2
set swModel = swApp.NewDocument (TemplateName, Papersize, _ width, Height)
'option 3
Dim swModel As sldworks.ModelDoc2
Set swModel = swApp.OpenDoc6 (FileName, Type, Options, _
Config, Errors, Warnings)
```

SolidWorks API Object Model – cont'

- ModelDoc2 Methods and Properties

.InsertSketch2 (updateEditRebuild)
.InsertFamilyTableNew ( )
.InsertNote (Text)
.SetToolbarvisibility (Toolbar, Visible)
.AddCustomInfo3 (Config, FieldName, FieldType, Fieldvalue)
.CreateCircle2 (xc, yc, zc, xp, xp, xp)
.EditRebuild3 ( )
.FeatureManager ( )
.InsertFeatureShell (Thickness, Outward)
.SaveAs4 (Name, Version, Options, Errors, Warnings)
.ViewZoomtofit2 ( )

SolidWorks API Object Model – cont'

- Documents Objects(Part, Assembly, Drawing)
  - Document objects are derived from ModelDoc2, therefore, they have access to all of the functions residing on the ModelDoc2 object. To connect to a Document object use one of the ModelDoc2 accessors and perform an a simple error check to validate the file type.

```
Dim swModel As Sldworks.ModelDoc2
Dim swPart AS SldWorks.PartDoc
Dim swAssy As SldWorks.AssemblyDoc
Dim swDraw As SldWorks.DrawingDoc

set swModel = swApp.ActiveDoc
If (swModel.GetType = swDocPART) Then
   Set swPart = swModel
End if
If (swModel.GetType = swDocASSEMBLY) Then
   Set swASSy = swModel
End if
If (swModel.GetType = swDocDRAWING) Then
   Set swDraw = swModel
End if
If swModel Is Nothing Then
   swApp.SendMsgToUser "Open a part, assembly or drawing!"
End if
```

SolidWorks API Object Model – cont'

- PartDoc Methods and Properties

```
.EditRollback ( )
.MaterialPropertyValues ( )
.CreateNewBody ( )
.MirrorPart ( )
```

- AssemblyDoc Methods and Properties

```
.AddComponent4 (CompName, ConfigName, X, Y, Z)
.AddMate3 (MateType, Align, Flip, Distance, dUpperLimit, dLowerLimit,
        RatioNumerator, RatioDenominator, Angle, aUpperLimit, aLowerLimit,
        PositioningOnly, ErrorStatus)
.InsertNewPart2 (FilePathIn, face_or_Plane_to_select)
.ToolsCheckInterference2 (NoComp, LStComp, Coincident, Comp, Face)
```

- DrawingDoc Methods and Properties

```
.GetFirstView
.InsertModelAnnotations (option, AllTypes, Types, AllViews)
.NewSheet3 (Name, Size, In, S1, S2, FA, TplName, W, H, PropV)
```

The End