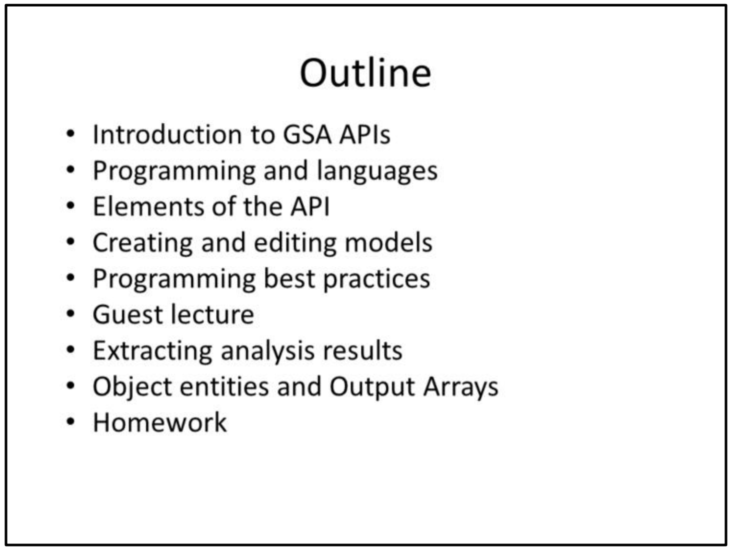
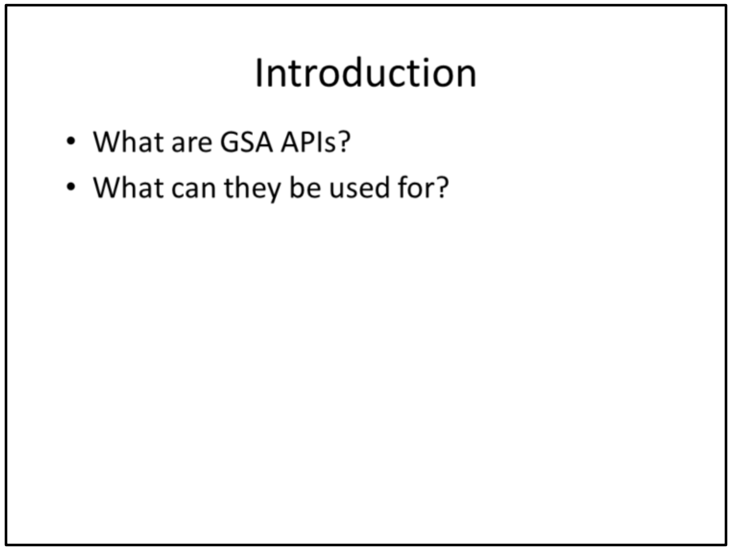
Programming with GSA APIs

An introduction to the COM interface



Outline

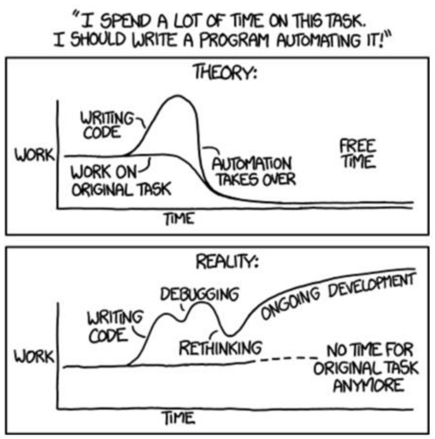
- Introduction to GSA APIs
- Programming and languages
- Elements of the API
- Creating and editing models
- Programming best practices
- Guest lecture
- Extracting analysis results
- Object entities and Output Arrays
- Homework

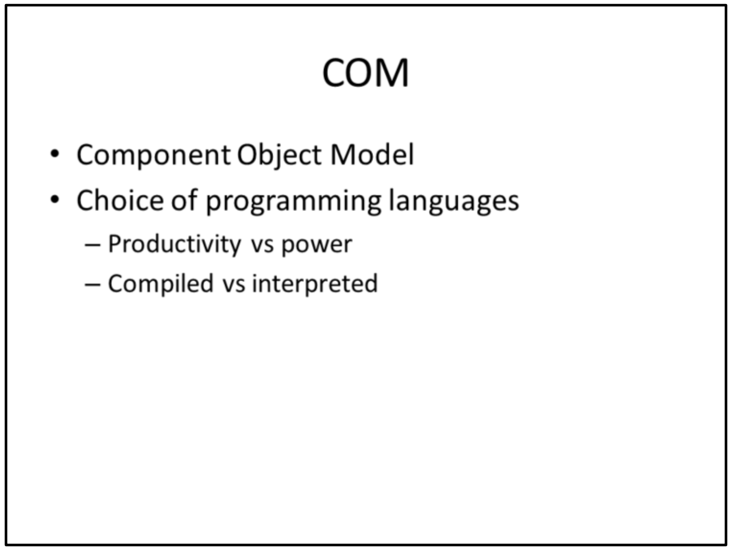


Introduction

- What are GSA APIs?
- What can they be used for?

- GSA APIs are a set of programming functions which users can use to control GSA without opening an interactive session, ie programmatically or ‘remotely’. Many actions that can be performed using the UI can be executed directly or indirectly, by writing a set of instructions - a program or a script – to drive GSA.
- APIs do not add functionality, they are an additional interface to existing functionality
- Uses -They can be used for automating anything within GSA.
    - Scripts can be written for any GSA related task ranging from generating models, loading, automatically running batched analysis to extracting results, post-processing, printing views.
    - Can also be a basis for writing programs that use GSA as a solver engine to generate analysis results for, say, sizing algorithms.
    - For writing plugins that work within 3D modelling software like Rhino and generate a GSA model based on the 3D model.


"I SPEND A LOT OF TIME ON THIS TASK. I SHOULD WRITE A PROGRAM AUTOMATING IT!"
THEORY:
WRITING CODE
WORK
WORK ON ORIGINAL TASK
AUTOMATION TAKES OVER
FREE TIME
TIME
REALITY:
DEBUGGING
WRITING CODE
ONGOING DEVELOPMENT
RETHINKING
WORK
NO TIME FOR ORIGINAL TASK ANYMORE
TIME


COM

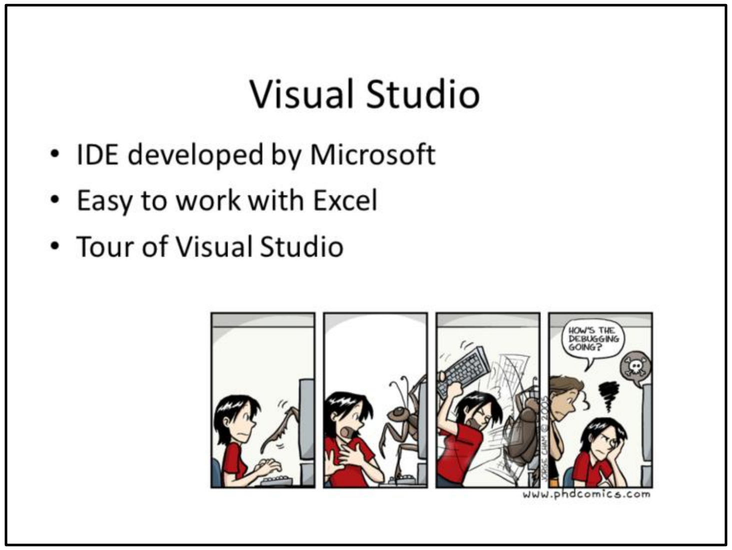
- Component Object Model
- Choice of programming languages
  - Productivity vs power
  - Compiled vs interpreted

1. COM is a standard for exposing functions across executable boundaries
    1. Why is it needed? In order to access data and functionality of a program like GSA, you would typically need access to source code, and the knowledge to write programs in the language that the program is written in (C++ in this case). COM allows software vendors like Oasys to allow users to call code that is embedded within GSA without having to expose the entire source code and to write their code in any language of their choosing. This is made possible by using functionality that is embedded in the Windows OS. GSA uses this functionality to create a "library" of functions that can be called from environments like VBA.
2. Allows programs such as GSA to 'expose' functions.
    1. These functions would allow us to query, create, or edit information in the program
3. Users can write other programs and scripts to 'consume' these functions.
4. Scripts can be written in any 'COM compliant' language. Examples C++, VBA, VB.NET, C, Python, MATLAB
5. C and C++ are very powerful and give a lot of control to the user. They also have a steep learning curve. OTOH VBA and Python are easy to start using but are arguably less flexible.

Note: any differences on opinions about programming languages should be settled in the parking lot.

DEMO – show Python execution vs Visual Studio

We will be using VB.NET in this course today and you have all been introduced to it through the videos and self-assessments.



Visual Studio

- IDE developed by Microsoft
- Easy to work with Excel
- Tour of Visual Studio

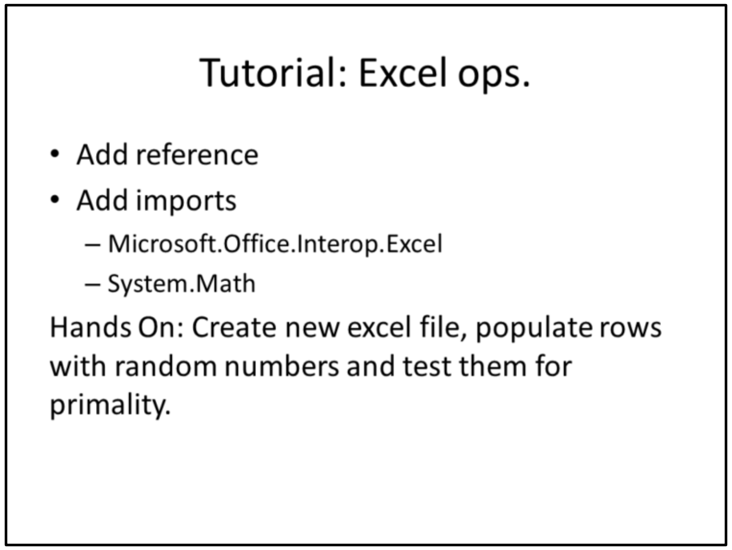
A IDE is an integrated development environment, used for editing, compiling, debugging code. Other capabilities include documentation and source control. Although it is possible to develop software without using IDEs, using one is highly recommended no matter what your level of expertise/experience is as a programmer.

Visual Studio is the market leading IDE –easy to start using and developing code using VS. "very easy" is in a relative sense, in comparison with some other alternatives!

Paid and free versions available. VS Express is a fully function free version – you don't get documentation with it but that is freely accessible from the web.

It is possible to call Excel from VB.NET. You can get edit data in spreadsheets much the same way you would in VBA inside Excel. See comparison chart here https://msdn.microsoft.com/en-us/library/ss11825b(v=vs.110).aspx

(The excel object invocation is similar to the GSA one in that they are both COM based interfaces.)

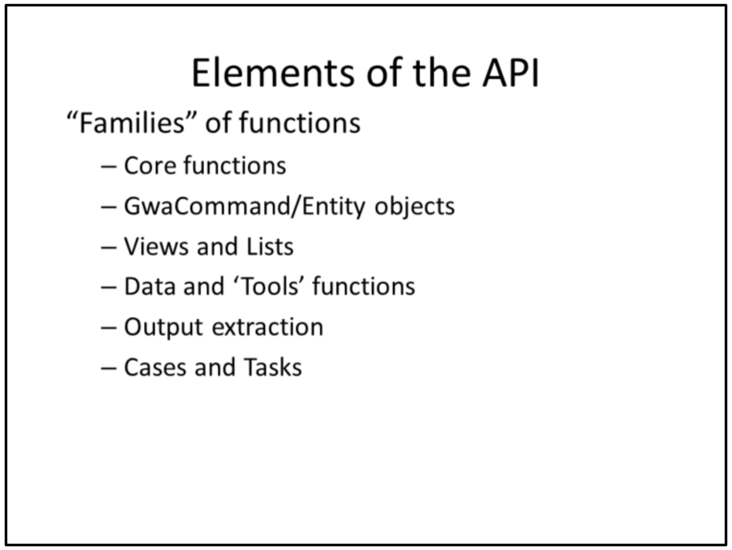

Tutorial: Excel ops.

- Add reference
- Add imports
  - Microsoft.Office.Interop.Excel
  - System.Math

Hands On: Create new excel file, populate rows with random numbers and test them for primality.

```
Dim excel As Microsoft.Office.Interop.Excel.Application = New As Microsoft.Office.Interop.Excel.Application

    Dim wb As Workbook = excel.Workbooks.Add()

    Dim sheet As Worksheet = wb.Worksheets(1)


    If IsNothing(sheet) Then
       Console.Write("sheet is null")
    End If

    For i As Integer = 1 To 100
       sheet.Cells(i, 1).Value = excel.RandBetween(1, 1000)
    Next

    'Console.WriteLine(sheet.Name.ToString())
```

```
For i As Integer = 1 To 100
    Dim n As Integer = sheet.Cells(i, 1).Value
    If IsPrime(n) Then
        sheet.Cells(i, 2).Value = "Prime"
    End If
Next

wb.SaveAs("c:\GSA_training\primes.xlsx")
excel.Quit()

excel = Nothing
```

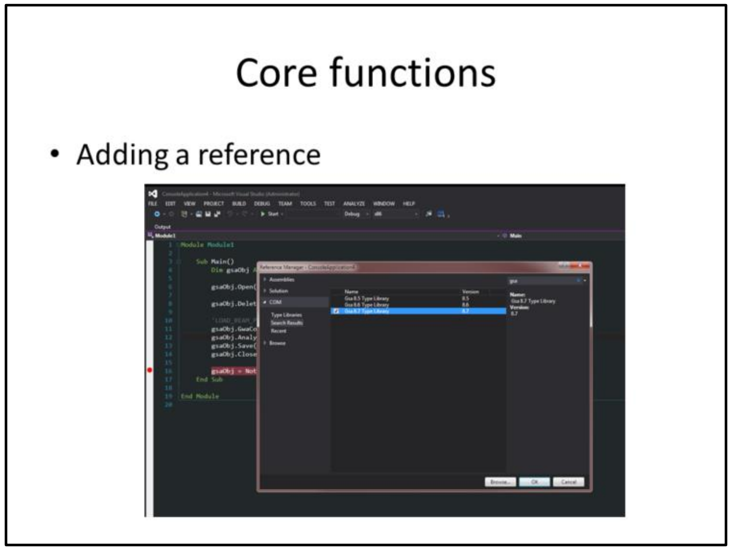


Elements of the API

"Families" of functions

- Core functions
- GwaCommand/Entity objects
- Views and Lists
- Data and 'Tools' functions
- Output extraction
- Cases and Tasks

- The GSA COM object offers functionality that can be grouped in "families". (this is purely for clarity and ease of understanding. The library makes no such distinction.)
    - Core: pertain to basic operations Open/Close files, SaveAs, Analyse etc.
    - View: Deal querying, printing and saving of views.
    - Data and tool functions: For querying, creating and modifying entities like nodes, member-element relationships, generate regions etc.
    - GwaCommand: the main data function for all model editing tasks.
        - Entity objects: alternative way of editing a small subset of entities
    - Output functions are used to retrieve results from analysis and combination cases.
    - Case and Task functions – to interrogate and enumerate analysis cases and analysis tasks. Also to see if results exist for a task etc.



Core functions

- Adding a reference

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Start
Debug
Output
Module1
Main
Reference Manager - ConsoleApplication4
Assemblies
Solution
COM
Type Libraries
Search Results
Recent
Browse
gsa
Browse...
OK
Cancel

```
Module Module1

    Sub Main()
        Dim gsaObj A

        gsaObj.Open(

        gsaObj.Delet

        'LOAD_BEAM_F
        gsaObj.GwaCo
        gsaObj.Analy
        gsaObj.Save(
        gsaObj.Close

        gsaObj = Not
    End Sub

End Module
```

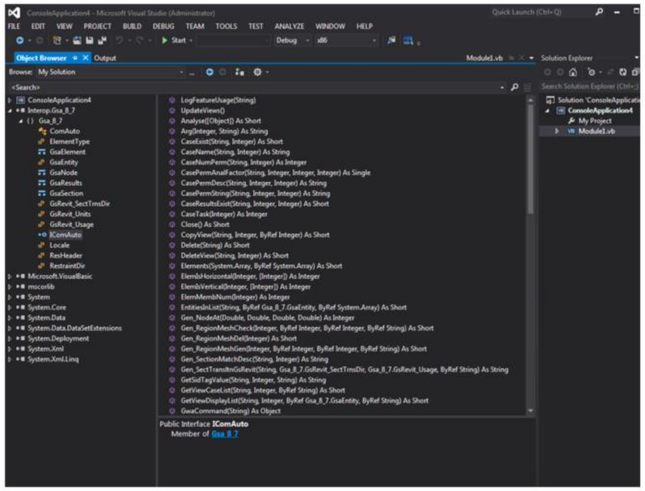
Name	Version
Gsa 8.5 Type Library	8.5
Gsa 8.6 Type Library	8.6
Gsa 8.7 Type Library	8.7

Name:
Gsa 8.7 Type Library
Version:
8.7

The first step to invoking the GSA COM object is to add a reference to it in your visual studio solution.

This is a useful step and even though it’s possible to work without it, it is strongly recommended.

Adding the reference tells your VB.NET project about the existence of the GSA COM object.

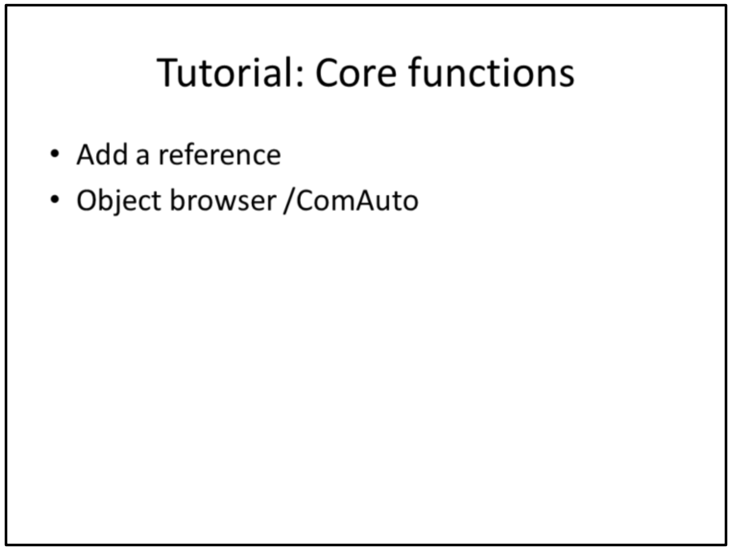

ConsoleApplication4 - Microsoft Visual Studio (Administrator)
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Start
Debug
x86
Object Browser
Output
Module1.vb
Solution Explorer
Browse: My Solution
<Search>
Search Solution Explorer (Ctrl+;)
Solution 'ConsoleApplicati
ConsoleApplication4
My Project
Module1.vb

```
ConsoleApplication4
Interop.Gsa_8_7
  Gsa_8_7
    ComAuto
    ElementType
    GsaElement
    GsaEntity
    GsaNode
    GsaResults
    GsaSection
    GsRevit_SectTrnsDir
    GsRevit_Units
    GsRevit_Usage
    IComAuto
    Locale
    ResHeader
    RestraintDir
Microsoft.VisualBasic
mscorlib
System
System.Core
System.Data
System.Data.DataSetExtensions
System.Deployment
System.Xml
System.Xml.Linq
```

```
LogFeatureUsage(String)
UpdateViews()
Analyse([Object]) As Short
Arg(Integer, String) As String
CaseExist(String, Integer) As Short
CaseName(String, Integer) As String
CaseNumPerm(String, Integer) As Integer
CasePermAnalFactor(String, Integer, Integer, Integer) As Single
CasePermDesc(String, Integer, Integer) As String
CasePermString(String, Integer, Integer) As String
CaseResultsExist(String, Integer, Integer) As Short
CaseTask(Integer) As Integer
Close() As Short
CopyView(String, Integer, ByRef Integer) As Short
Delete(String) As Short
DeleteView(String, Integer) As Short
Elements(System.Array, ByRef System.Array) As Short
ElemIsHorizontal(Integer, [Integer]) As Integer
ElemIsVertical(Integer, [Integer]) As Integer
ElemMembNum(Integer) As Integer
EntitiesInList(String, ByRef Gsa_8_7.GsaEntity, ByRef System.Array) As Short
Gen_NodeAt(Double, Double, Double, Double) As Integer
Gen_RegionMeshCheck(Integer, ByRef Integer, ByRef Integer, ByRef String) As Short
Gen_RegionMeshDel(Integer) As Short
Gen_RegionMeshGen(Integer, ByRef Integer, ByRef Integer, ByRef String) As Short
Gen_SectionMatchDesc(String, Integer) As String
Gen_SectTransltnGsRevit(String, Gsa_8_7.GsRevit_SectTrnsDir, Gsa_8_7.GsRevit_Usage, ByRef String) As String
GetSidTagValue(String, Integer, String) As String
GetViewCaseList(String, Integer, ByRef String) As Short
GetViewDisplayList(String, Integer, ByRef Gsa_8_7.GsaEntity, ByRef String) As Short
GwaCommand(String) As Object
```

```
Public Interface IComAuto
    Member of Gsa_8_7
```

Once a reference is added, you can look at all functions that are exposed by the GSA COM object.

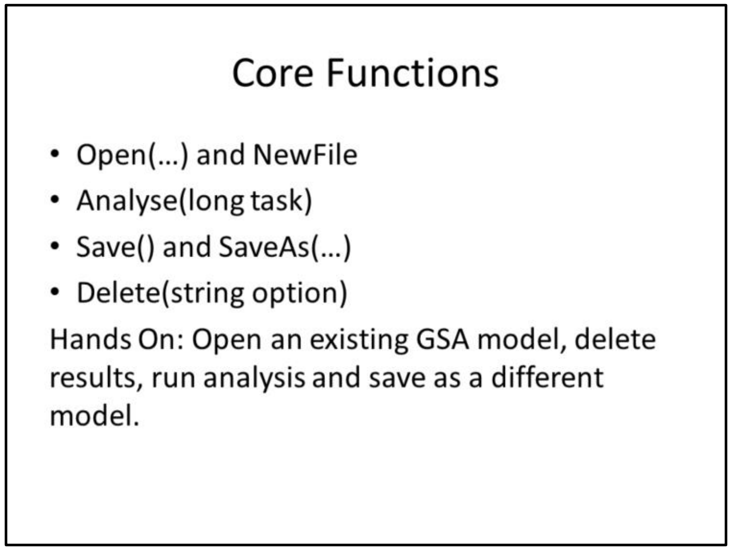

Tutorial: Core functions

- Add a reference
- Object browser /ComAuto

- The first step to invoking the GSA COM object is to add a reference to it in your visual studio solution.
- Copy the following into your projects

```
Dim gsaObj As New gsa_8_7.ComAuto

    gsaObj.Open("c:\GSA_training\Env.gwb")

    gsaObj.Delete("RESULTS_AND_CASES")

    'LOAD_BEAM_POINT.2 | name | list | case | axis | proj | dir | pos |  value
    gsaObj.GwaCommand("LOAD_BEAM_POINT.2,load added by COM,2,4,GLOBAL,NO,Z,-.25,-12")
    gsaObj.Analyse()
    gsaObj.SaveAs("c:\COM-daylong\Env_loaded.gwb")
    gsaObj.Close()
```

```
gsaObj = Nothing
```


Core Functions

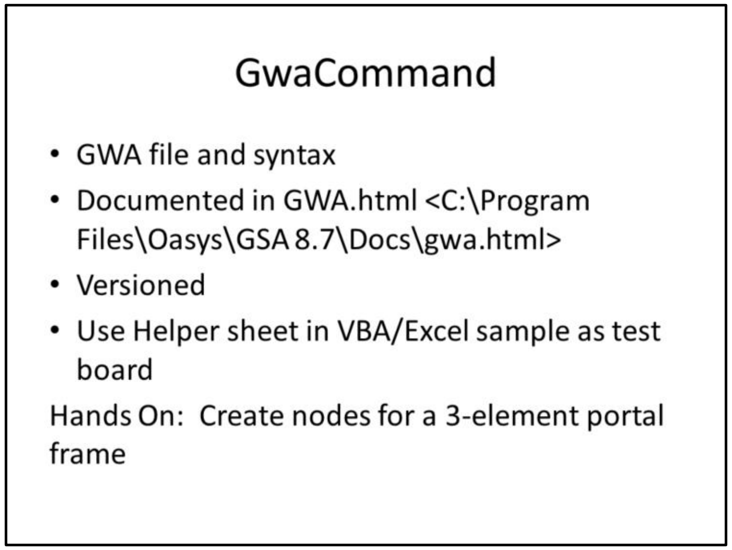
- Open(…) and NewFile
- Analyse(long task)
- Save() and SaveAs(…)
- Delete(string option)

Hands On: Open an existing GSA model, delete results, run analysis and save as a different model.

Open and SaveAs take file names (with full file paths) as string arguments
NewFile and Save() don’t have any arguments

Analyse(…) has a default argument, which when unspecified, means “analyse all tasks”. When specified, it is the task number and it must exist.

Delete deletes results OR results and analysis cases.


GwaCommand

- GWA file and syntax
- Documented in GWA.html <C:\Program Files\Oasys\GSA 8.7\Docs\gwa.html>
- Versioned
- Use Helper sheet in VBA/Excel sample as test board

Hands On: Create nodes for a 3-element portal frame

- GwaCommand : the main data function for all model editing tasks. It is a single function but it enables editing and creation of ALL objects within a GSA model. Operates through the GWA text file syntax. Each “command” is a string (of chars) in the GWA format, that will read from or write data into the GSA model.
  - Show use of GWA copy paste in the UI and GWA text file.
  - Show Gwa.html
  - Show Helper sheet in excel COM sample as an example of GwaCommand call
  - GWA versioning. E.g.
    - JOINT.2 | name | slave | x | y | z | xx | yy | zz | master | stage
    - JOINT.1 | slave | x | y | z | xx | yy | zz | master | stage

Lists

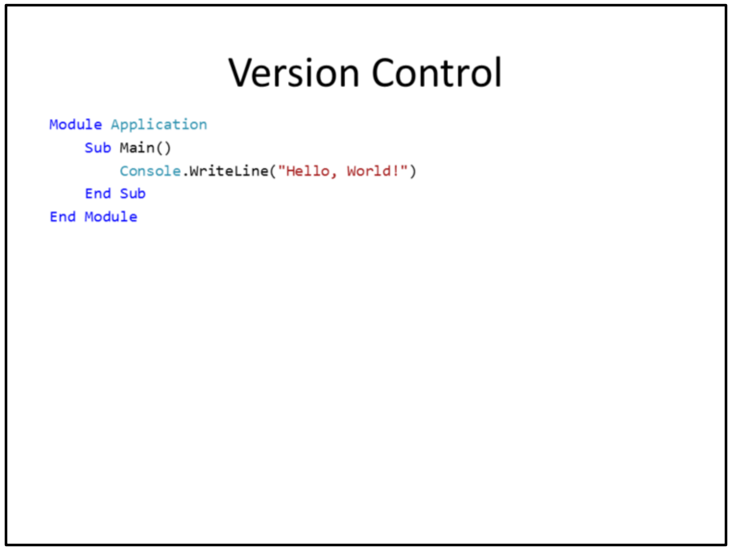
- Saved lists and dynamically created
- Work with an array argument

Hands On – Encastre all ground floor nodes for portal frame

Programming best practices

WE WILL BE ADOPTING THE BEST PRACTICES IN OUR INDUSTRY, JUST LIKE EVERYONE ELSE.
BEST PRACTICES
www.dilbert.com scottadams@aol.com
IF EVERYONE IS DOING IT, BEST PRACTICES IS THE SAME THING AS MEDIOCRE.
© 2008 Scott Adams, Inc./Dist. by UFS, Inc.
STOP MAKING MEDIOCRITY SOUND BAD!
SORRY.

We will look at version control, code reuse via functions, classes and libraries.

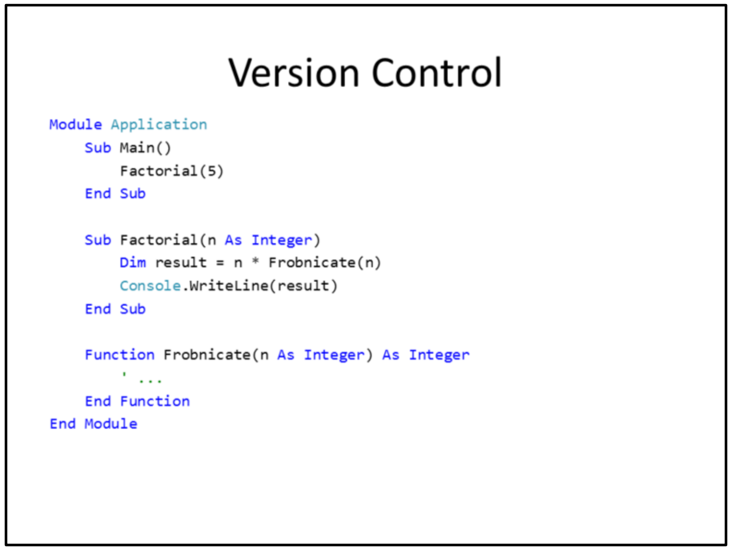


Version Control

```
Module Application
    Sub Main()
        Console.WriteLine("Hello, World!")
    End Sub
End Module
```

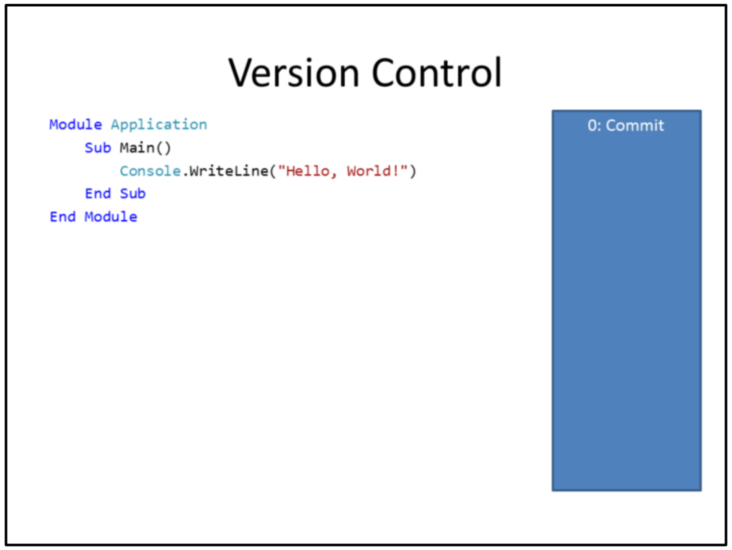
Version Control

```
Module Application
    Sub Main()
        Factorial(5)
    End Sub

    Sub Factorial(n As Integer)
        Dim result As Integer = 1

        While n > 1
            result = result * n
            n = n - 1
        End While

        Console.WriteLine(result)
    End Sub
End Module
```



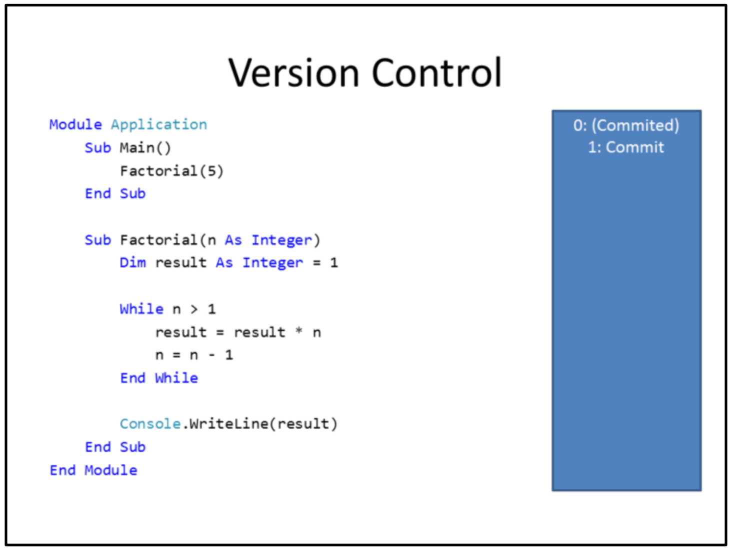
Version Control

```
Module Application
    Sub Main()
        Factorial(5)
    End Sub

    Sub Factorial(n As Integer)
        Dim result = n * Frobnicate(n)
        Console.WriteLine(result)
    End Sub

    Function Frobnicate(n As Integer) As Integer
        ' ...
    End Function
End Module
```



Version Control

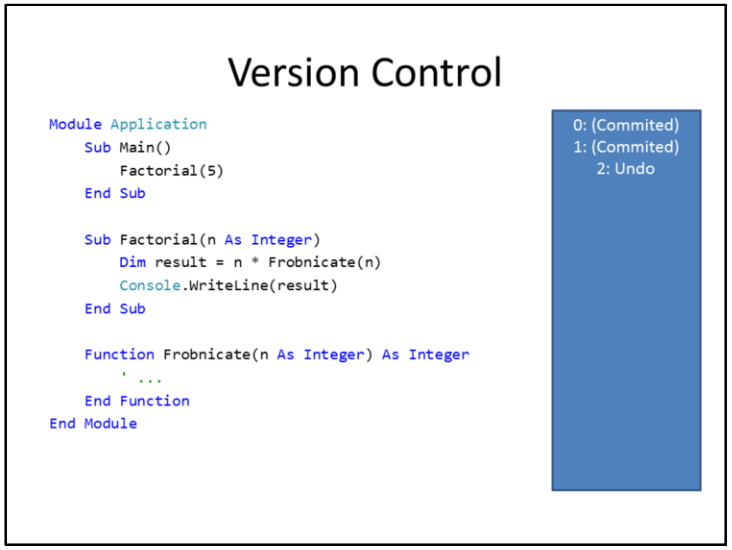
```
Module Application
    Sub Main()
        Console.WriteLine("Hello, World!")
    End Sub
End Module
```



Version Control

```
Module Application
    Sub Main()
        Factorial(5)
    End Sub

    Sub Factorial(n As Integer)
        Dim result As Integer = 1

        While n > 1
            result = result * n
            n = n - 1
        End While

        Console.WriteLine(result)
    End Sub
End Module
```

0: (Commited)
1: Commit



Version Control

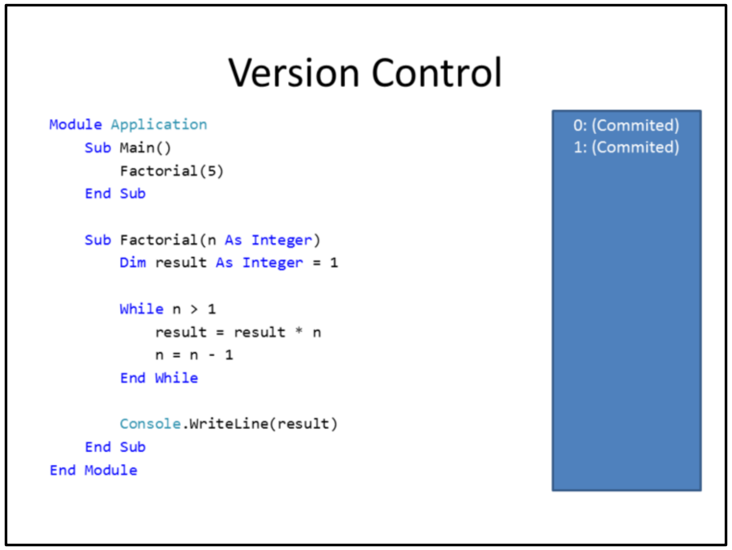
```
Module Application
    Sub Main()
        Factorial(5)
    End Sub

    Sub Factorial(n As Integer)
        Dim result = n * Frobnicate(n)
        Console.WriteLine(result)
    End Sub

    Function Frobnicate(n As Integer) As Integer
        ' ...
    End Function
End Module
```

0: (Commited)
1: (Commited)
2: Undo

Version Control

```
Module Application
    Sub Main()
        Factorial(5)
    End Sub

    Sub Factorial(n As Integer)
        Dim result As Integer = 1

        While n > 1
            result = result * n
            n = n - 1
        End While

        Console.WriteLine(result)
    End Sub
End Module
```

0: (Commited)
1: (Commited)



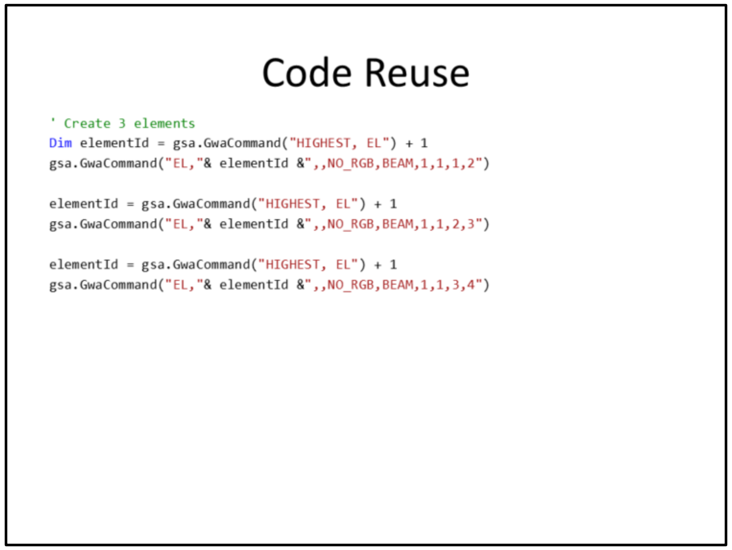
Version Control

```
Module Application
    Sub Main()
        Factorial(5)
    End Sub

    Sub Factorial(n As Integer)
        Dim result As Integer = 1

        While n > 1
            result = result * n
            n = n - 1
        End While

        Console.WriteLine(result)
    End Sub
End Module
```

0: (Commited)
1: (Commited)

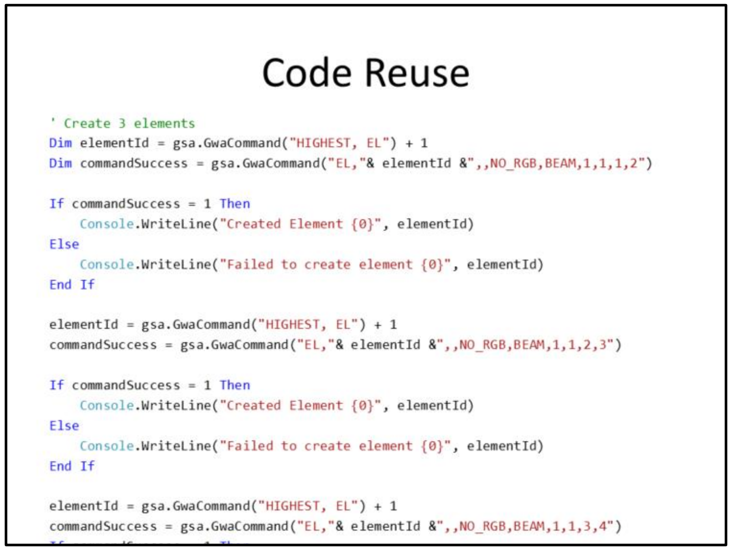
Code Reuse

```
' Create 3 elements
gsa.GwaCommand("EL,1,,NO_RGB,BEAM,1,1,1,2")
gsa.GwaCommand("EL,2,,NO_RGB,BEAM,1,1,2,3")
gsa.GwaCommand("EL,3,,NO_RGB,BEAM,1,1,3,4")
```



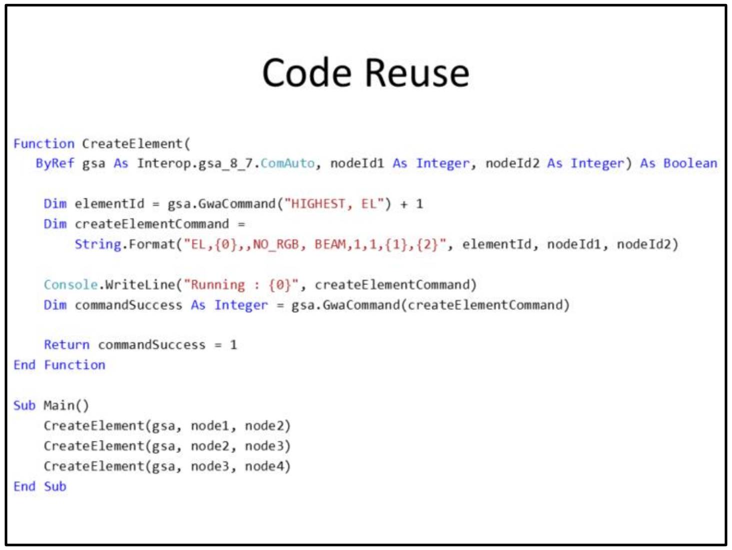
Code Reuse

```
' Create 3 elements
Dim elementId = gsa.GwaCommand("HIGHEST, EL") + 1
gsa.GwaCommand("EL,"& elementId &",,NO_RGB,BEAM,1,1,1,2")

elementId = gsa.GwaCommand("HIGHEST, EL") + 1
gsa.GwaCommand("EL,"& elementId &",,NO_RGB,BEAM,1,1,2,3")

elementId = gsa.GwaCommand("HIGHEST, EL") + 1
gsa.GwaCommand("EL,"& elementId &",,NO_RGB,BEAM,1,1,3,4")
```



Code Reuse

```
' Create 3 elements
Dim elementId = gsa.GwaCommand("HIGHEST, EL") + 1
Dim commandSuccess = gsa.GwaCommand("EL,"& elementId &",,NO_RGB,BEAM,1,1,1,2")

If commandSuccess = 1 Then
    Console.WriteLine("Created Element {0}", elementId)
Else
    Console.WriteLine("Failed to create element {0}", elementId)
End If

elementId = gsa.GwaCommand("HIGHEST, EL") + 1
commandSuccess = gsa.GwaCommand("EL,"& elementId &",,NO_RGB,BEAM,1,1,2,3")

If commandSuccess = 1 Then
    Console.WriteLine("Created Element {0}", elementId)
Else
    Console.WriteLine("Failed to create element {0}", elementId)
End If

elementId = gsa.GwaCommand("HIGHEST, EL") + 1
commandSuccess = gsa.GwaCommand("EL,"& elementId &",,NO_RGB,BEAM,1,1,3,4")
```

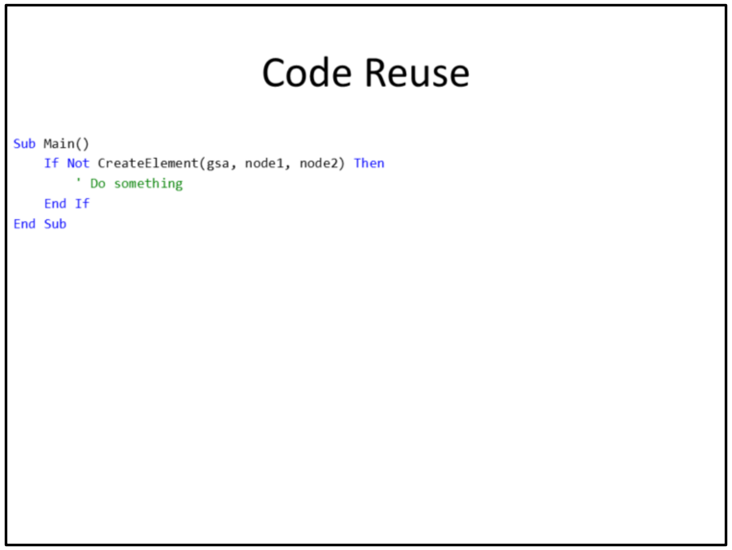


Code Reuse

```
Function CreateElement(
   ByRef gsa As Interop.gsa_8_7.ComAuto, nodeId1 As Integer, nodeId2 As Integer) As Boolean

    Dim elementId = gsa.GwaCommand("HIGHEST, EL") + 1
    Dim createElementCommand =
        String.Format("EL,{0},,NO_RGB, BEAM,1,1,{1},{2}", elementId, nodeId1, nodeId2)

    Console.WriteLine("Running : {0}", createElementCommand)
    Dim commandSuccess As Integer = gsa.GwaCommand(createElementCommand)

    Return commandSuccess = 1
End Function

Sub Main()
    CreateElement(gsa, node1, node2)
    CreateElement(gsa, node2, node3)
    CreateElement(gsa, node3, node4)
End Sub
```

Looking at that code, how can we improve it, make it more reusable and future-proof?

It is possible to encapsulate the Element GWA code in the previous example into a function

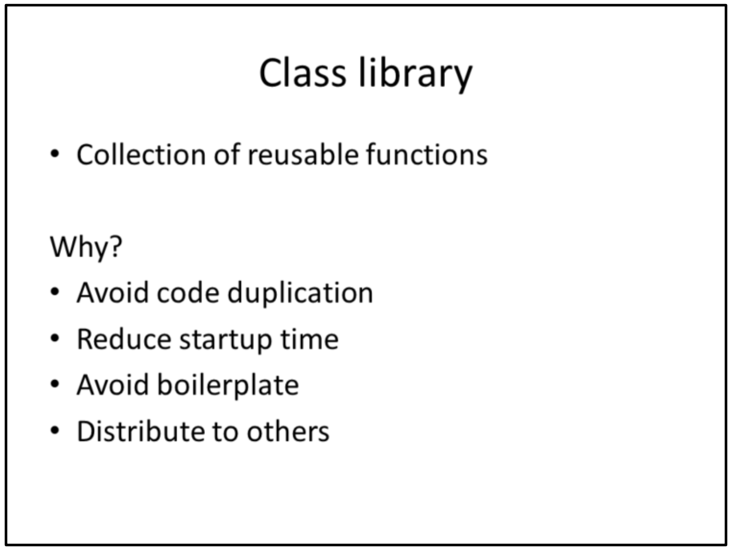


Code Reuse

```
Sub Main()
    If Not CreateElement(gsa, node1, node2) Then
        ' Do something
    End If
End Sub
```

Looking at that code, how can we improve it, make it more reusable and future-proof?

It is possible to encapsulate the Element GWA code in the previous example into a function



Class library

- Collection of reusable functions

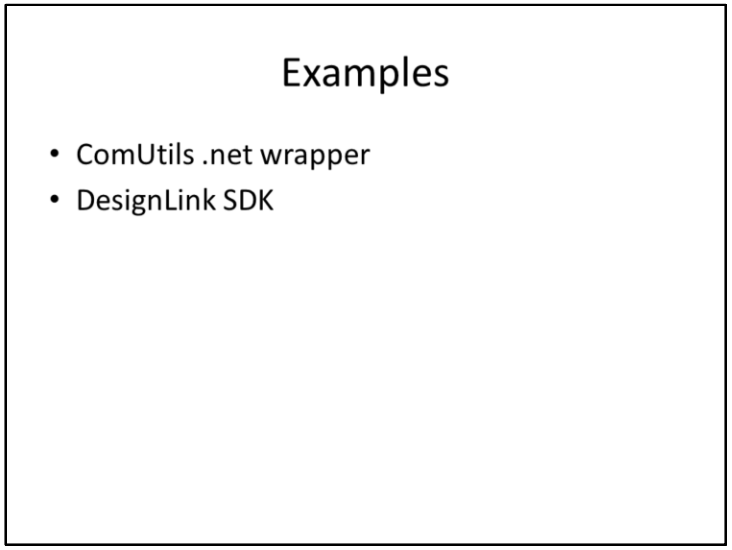
Why?

- Avoid code duplication
- Reduce startup time
- Avoid boilerplate
- Distribute to others

It’s possible to collect functions that work with input and output into a wrapper class.
This class could then be distributed and reused.
The advantages code:

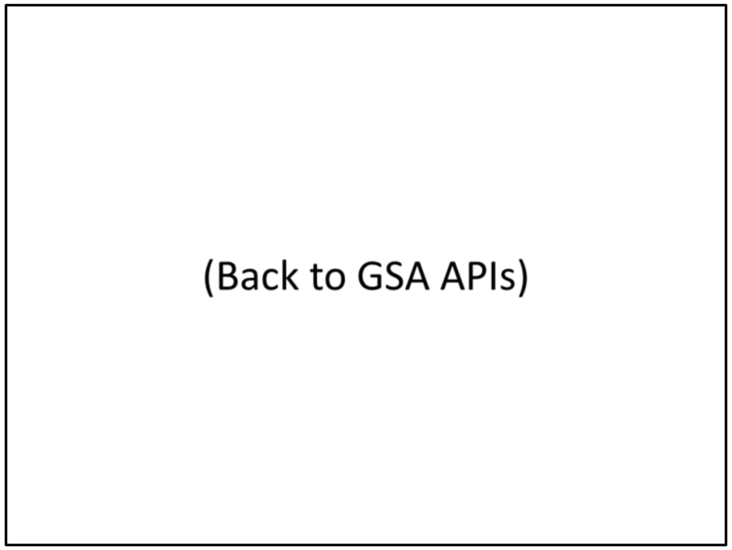
- Fix bugs in only one place – avoid code duplication.
- Save time on projects getting started
- Avoid boiler plate code through reuse
- Distribute across the firm – instant stardom

Classes allow encapsulation. Keep all data and functions in the “same place”

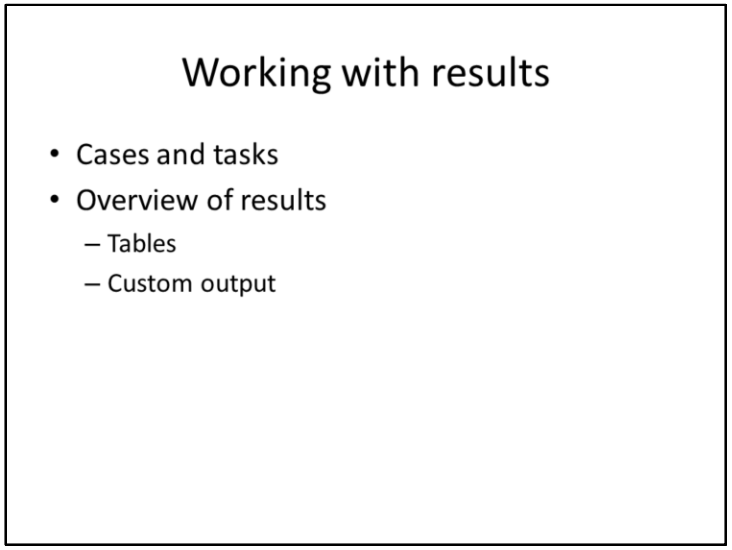


Examples

- ComUtils .net wrapper
- DesignLink SDK



(Back to GSA APIs)



Working with results

- Cases and tasks
- Overview of results
  - Tables
  - Custom output

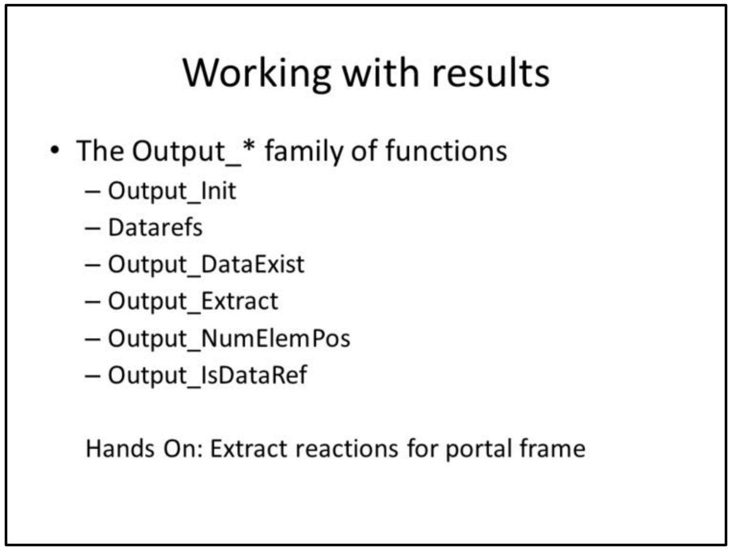
Review API functions for cases and tasks

- Overview of results:

In GSA results in output views are broadly of two categories (from a software developer’s perspective): tabulated results and custom tables

Tabulated results are all results that are presented in tables or are available for diagrams or contours. E.g. nodal displacements, beam forces, steel utilizations

Custom tables are those that are presented as a piece of text in output views. Example dynamic details or steel design verbose output


Working with results

- The Output_* family of functions
  - Output_Init
  - Datarefs
  - Output_DataExist
  - Output_Extract
  - Output_NumElemPos
  - Output_IsDataRef

Hands On: Extract reactions for portal frame

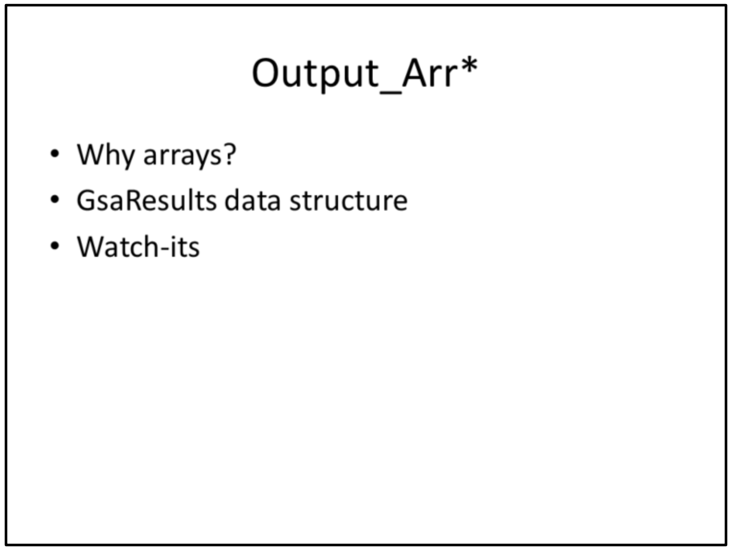
Tabulated results can be extracted using the Output_ family of functions.

The first step is to understand what DATAREFs are – demonstrate this by opening the file and showing the equivalent output in the UI.

Using output_ family of functions is a 3-step process –

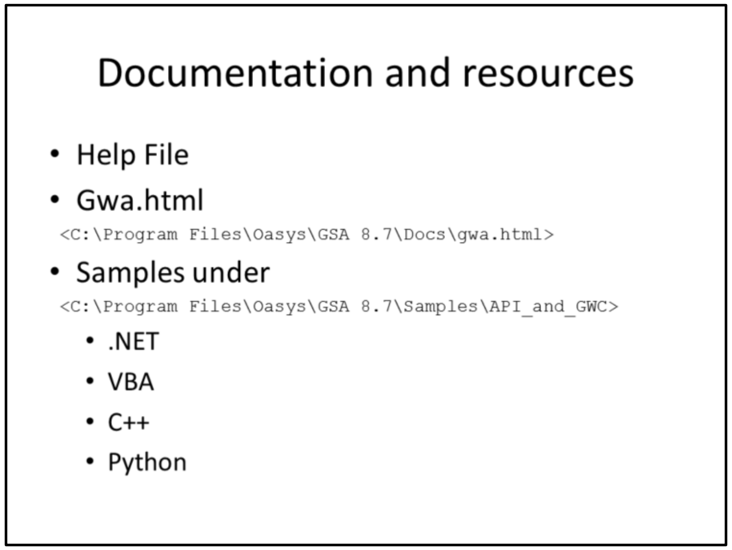
1. Output_Init
2. Output_Exist
3. Output_Extract

Things to remember: check for error codes and use the DataExist, CaseExist etc to carry out sanity checks



Output_Arr*

- Why arrays?
- GsaResults data structure
- Watch-its



Documentation and resources

- Help File
- Gwa.html

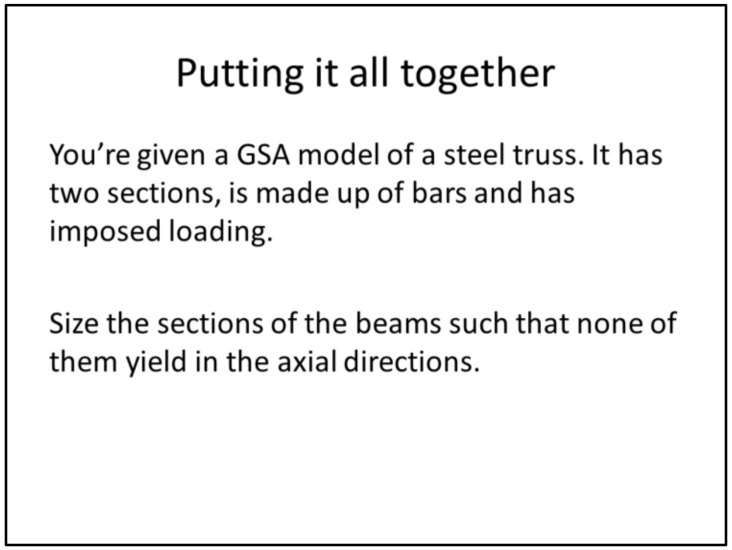
```
<C:\Program Files\Oasys\GSA 8.7\Docs\gwa.html>
```

- Samples under

```
<C:\Program Files\Oasys\GSA 8.7\Samples\API_and_GWC>
```

  - .NET
  - VBA
  - C++
  - Python

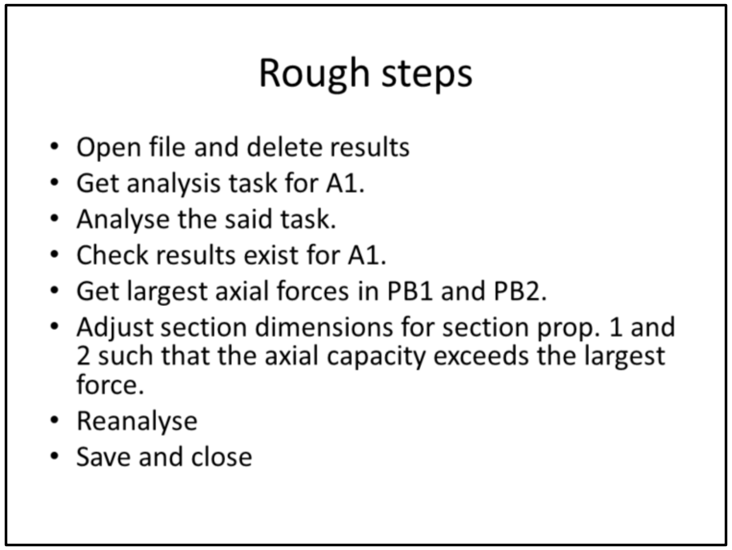
(Take a look at <C:\Program Files\Oasys\GSA 8.7\Samples\API_and_GWC\readme.txt>)


Putting it all together

You’re given a GSA model of a steel truss. It has two sections, is made up of bars and has imposed loading.

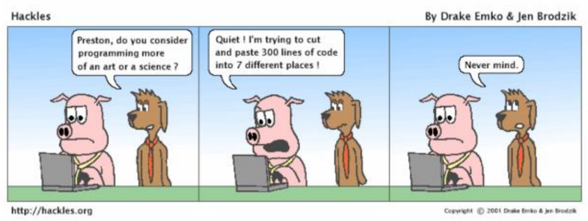
Size the sections of the beams such that none of them yield in the axial directions.

This is the final hands on for the course. The exercise is to size a simple truss model (tension/compression only). Use the model ‘truss.gwb’ in the ‘models’ folder.



Rough steps

- Open file and delete results
- Get analysis task for A1.
- Analyse the said task.
- Check results exist for A1.
- Get largest axial forces in PB1 and PB2.
- Adjust section dimensions for section prop. 1 and 2 such that the axial capacity exceeds the largest force.
- Reanalyse
- Save and close


Hackles
By Drake Emko & Jen Brodzik
Preston, do you consider programming more of an art or a science ?
Quiet ! I'm trying to cut and paste 300 lines of code into 7 different places !
Never mind.
http://hackles.org
Copyright © 2001 Drake Emko & Jen Brodzik