LUSAS

LUSAS Programmable Interface (LPI)

Customisation and Automation Guide

LUSAS Programmable Interface (LPI) Customisation and Automation Guide

LUSAS Version 24.0 : Issue 1

LUSAS
Forge House, 66 High Street, Kingston upon Thames,
Surrey, KT1 1HN, United Kingdom

Tel: +44 (0)20 8541 1999
Fax +44 (0)20 8549 9399
Email: info@lusas.com
http://www.lusas.com

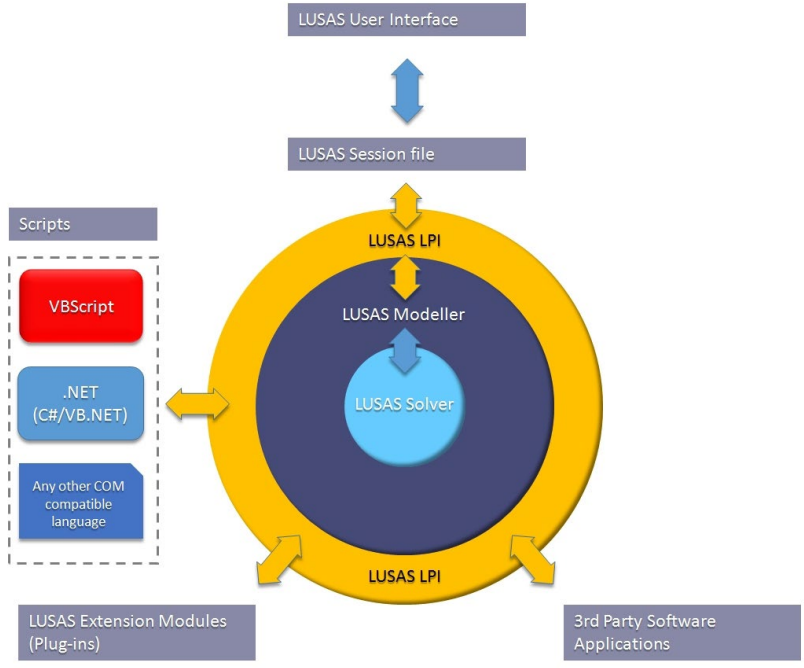
Distributors Worldwide

Table of Contents

Introduction

Introduction

LUSAS software is highly customisable. The built-in LUSAS Programmable Interface (LPI) allows the customisation and automation of modelling and results processing tasks and creation of user-defined menu items, dialogs and toolbars as a means to access those user-defined resources. It can also be used for transferring data between LUSAS and other software applications, and to control other programs from within LUSAS Modeller, or control LUSAS Modeller from other programs.



With LPI, any user can automate the creation of complete structures, either in LUSAS or from third-party software, carrying out design checks, optimising members and outputting graphs, spreadsheets of results and custom reports. Because everything carried out by a user is recorded in a LUSAS Modeller session file, anything that LUSAS can do, can also be controlled by another application via the LUSAS

Programmable Interface. This means that you can view and edit a recorded session, parameterise those commands, turn them into sub-routines, add loops and other functions to the scripts and create a totally different application or program - using the proven core technology of LUSAS.

In addition to the accessing and customising LUSAS Modeller via the LUSAS Programmable Interface, user-defined material models (written in Fortran) can be compiled and built into a customised LUSAS Solver executable by using the LUSAS Material Model Interface (LUSAS MMI).

Examples of capabilities

By using any ActiveX compliant scripting language, such as VB.Net, C#, VBScript, C++, Python, Perl, JScript etc. to access LUSAS facilities and functionality, you can:

- Create user-defined menu items, dialogs and toolbars
- Interrogate all aspects of a LUSAS model
- Customise modelling operations
- Create parameterised models
- Automate repetitive tasks
- Import CAD geometry and properties
- Make direct links to Microsoft Word / Excel, or other programs for import or export of data
- Perform simple / codified design checks and, when used with automated iterative analysis, optimise structural member sizes and configurations, slab reinforcement quantities, etc.

Scripts

In their simplest form script files can be used to store a sequence of LUSAS commands for later playback. Some examples of use include the creation of start-up templates to pre-load the Attributes Treeview of the LUSAS Modeller user interface with selected attributes for a particular analysis; the setting of default mesh or material types, or preferred colour schemes; or defining specific model orientations for use when saving model views for use in reports.

When LUSAS is run, a session file is created recording each step of the model generation in Visual Basic Script (.vbs or .lvb file extension from version 20 onwards) - one of the most commonly used and easily understood languages. Editing of a session file can be used to define a similar model with new parameters. When the script is re-run in LUSAS, a new user-defined model can be easily and rapidly generated from the parameters defined. A Macro Recorder facility in LUSAS also provides the means to record a sub-set of commands for a task, for saving and re-use. User-generated scripts

can be controlled by creating dialogs that may include parametric variables, check boxes, drop-downs etc.

Varied uses of scripts include reading of geometric data, such as column dimensions, section properties and span lengths / storey heights etc., from a spreadsheet to automatically build multi-span bridge or building models; rapid generation of parametrically-idealised wind farm base structures, or for automating the creation of numerous load combinations and envelopes.

A set of example scripts are provided in LUSAS to assist in the understanding of standard concepts including file handling, how to access LUSAS geometry / attribute data, and how to import / export data from / to Microsoft Word or Excel, or other programs.

Topics covered in this guide

The aim of this guide is to help you locate and use the supplied tools which will enable you to write scripts and work more efficiently. No programming experience or knowledge is needed to complete the examples shown. The guide covers:

- **Getting started with the LUSAS Programmable Interface (LPI)**
- **Identifying LPI Functions**
- **Customising the interface**
- **Getting started with VBS**
- **A simple example script**
- **Creating your own menus**

LPI Developer Guide

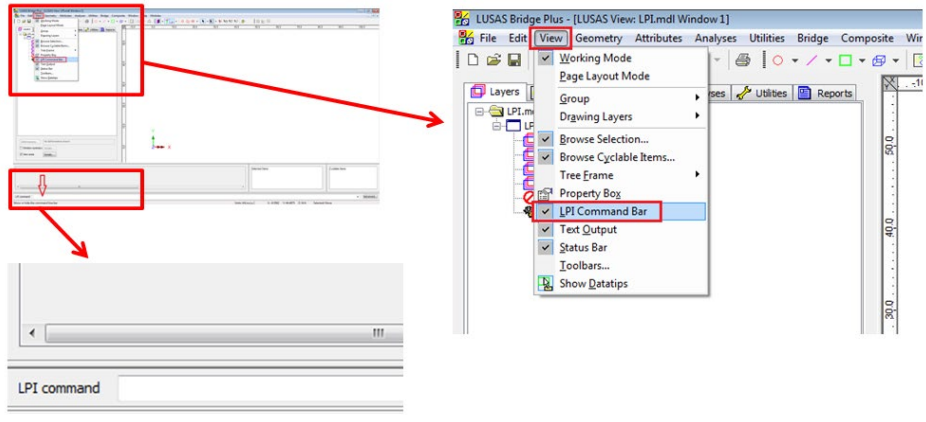
A separate LPI Developer Guide is also available covering more advanced topics:

- **Creating dialogs using VB.NET**
- **LUSAS via COM**
- **LUSAS Material Model Interface**

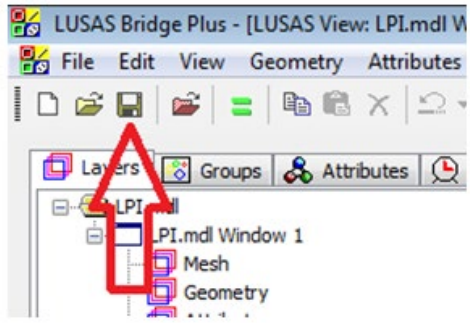
Getting started with the LUSAS Programmable Interface (LPI)

LPI Command Bar

The LPI Command Bar can be added to the user interface by selecting the menu item **View> LPI Command Bar**



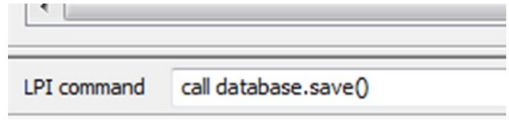
All actions performed by menu items within LUSAS Modeller can be performed by typing commands into the LPI Bar. For instance, when the **Save** button is selected in LUSAS Modeller, it is actually calling the function **database.save()**



Therefore, to save a model, you can type in the LPI command bar:

```
call database.save()
```

and then press Enter:



Commands can also be concatenated using a colon (:) character.

For example:

```
Txt = "Hello World" : call msgbox(txt)
```

Identifying LPI functions

There are two ways of identifying which LPI function corresponds to an operation carried out within LUSAS:

- **By writing LPI functions to a file**
- **By writing LPI functions to the Text Output window.**

Writing LPI functions to a file

1. Select the menu item **File > Script > Start Recording...**
2. Pick menu item(s), for example select **Tools > Vertical Axis** and click **OK.**
3. Select the menu item **File > Script > Stop Recording**

A .vbs file (.lvb file from version 20 onwards) will be saved to a chosen location. This file can then be edited with a text editor to see the LPI commands. An example follows:

```
$ENGINE=VBScript
' LUSAS Modeller session file
' Created by LUSAS 21.0-0c1 - Modeller Version 21.0.1601.44691
' Created at 14:17 on Thursday, September 28 2023
' (C) Finite Element Analysis Ltd 2023
'
call setCreationVersion("21.0-0c1, 21.0.1601.44691")
'
'*** Settings/Options/Properties change
call database.setVerticalDir("Y")
```

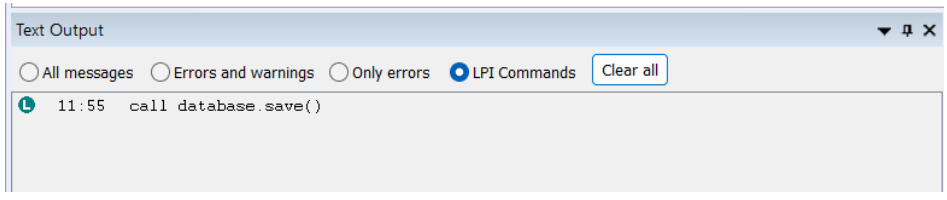
Re-running commands with Modeller

- Commands can be re-run within Modeller by selecting **File > Script > Run Script**... and choosing the previously saved .vbs file.
- The **Run Script**  button can also be selected to run scripts.

LPI functions in the Text Output window

For every operation carried out within LUSAS Modeller, the corresponding command will be written to the Text Output window – LPI Commands view.

For example: If you click on the **Save** button, you will see the following:

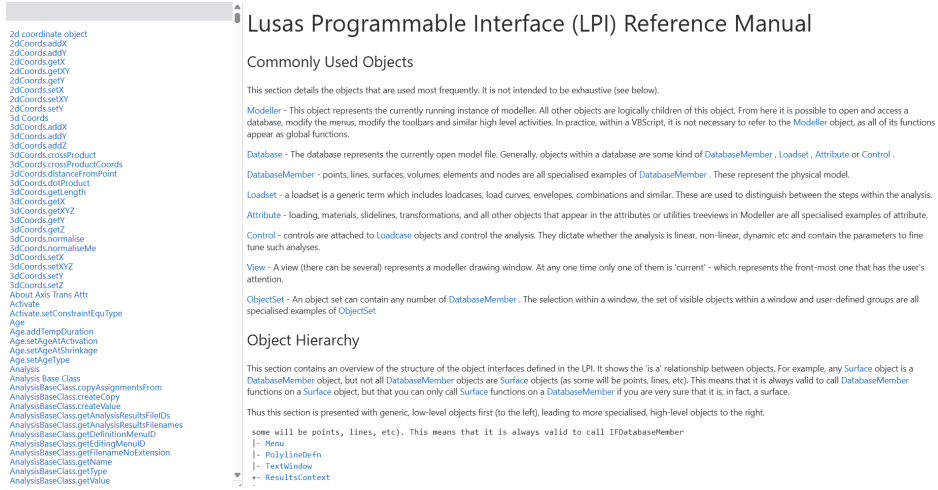


Details of LPI functions

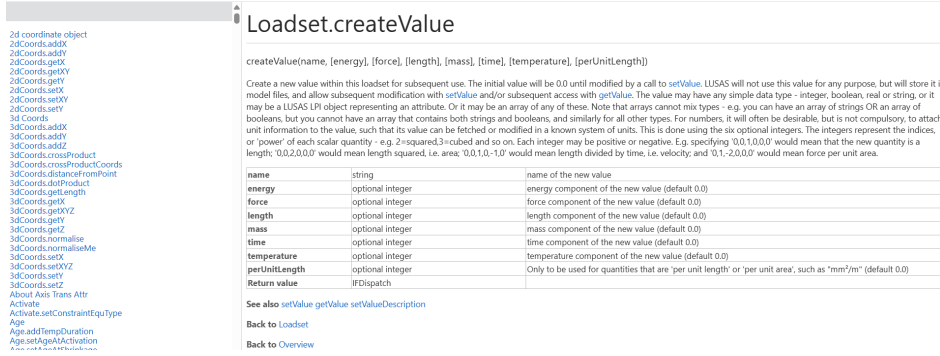
Details of all LPI functions along with an explanation of what they do, the arguments they take (if any), the returned values, etc, can be found by selecting the LUSAS menu item **Help > LPI Reference Manual**

LUSAS Programmable Interface (LPI) online help

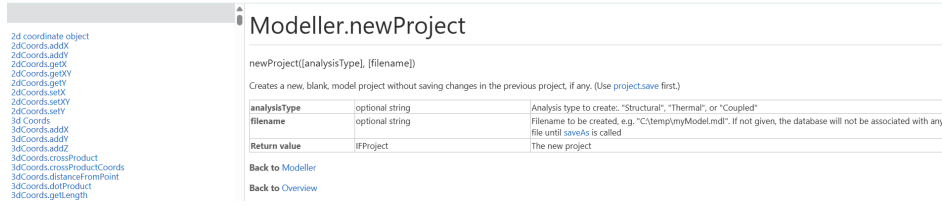
The left-hand pane of the LUSAS Programmable Interface help system contains a filtered list of all classes and functions that can be accessed within Modeller.



Clicking one of these classes or functions gives a description of the function with its input arguments and return values in the right-hand pane. Any argument shown in square brackets is optional.

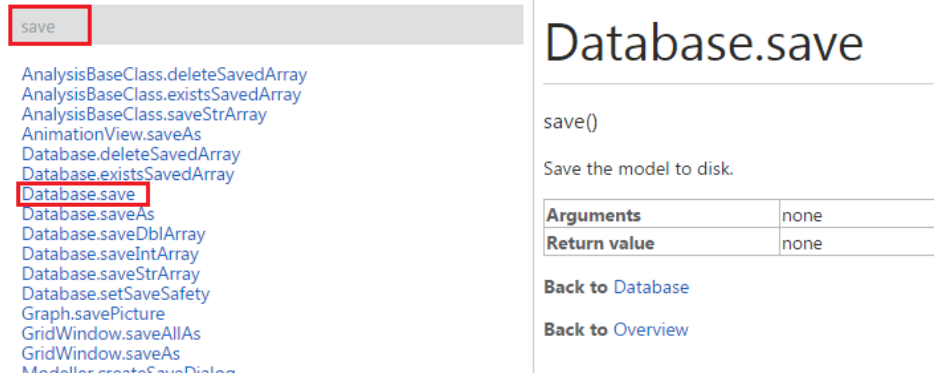


The creation of a new model will require a call to the **Modeller.newProject** function and from this other functions may be called to add to, manipulate or interrogate the state of the objects in the Modeller database.



Searching LPI help

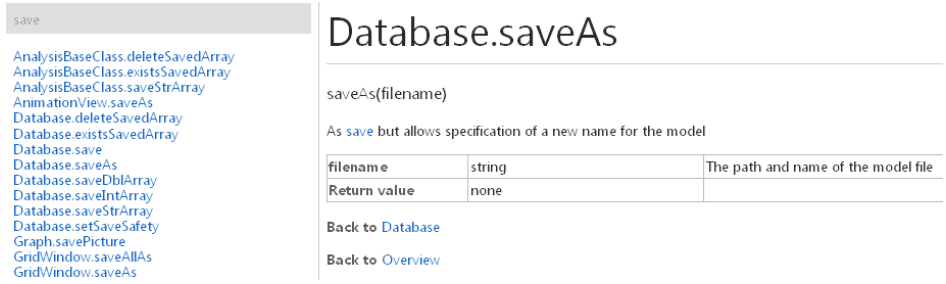
Searching of LPI help is possible. For example, a search for “save” will return results that include the database.save() function:



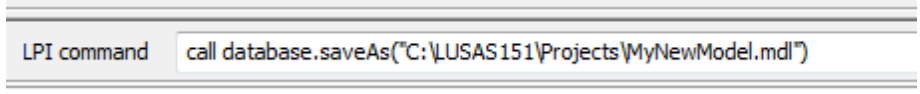
Some function basics

Note that the above **database.save()** function it does not take any parameters and does not return anything. It simply saves the model.

The **database.saveAs(filename)** function is an example of a function that does take a parameter (in this case, just one) comprising a string with the path and file name of the new model:

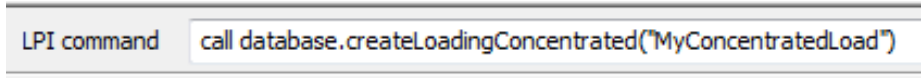


To specify that you are passing a string to the function and not any other type of data, strings need to be placed between double quotes: (“ ”)

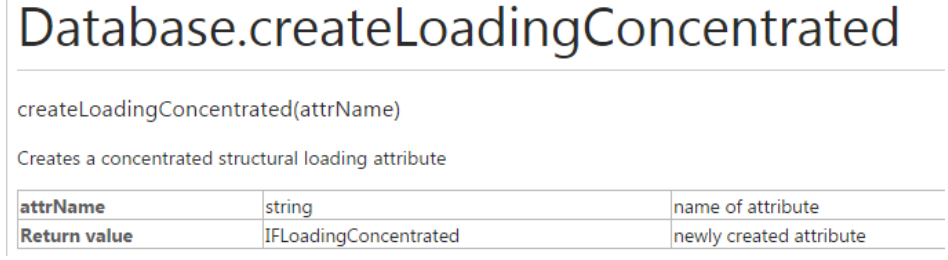


An example of a function that takes one parameter and returns a named attribute:

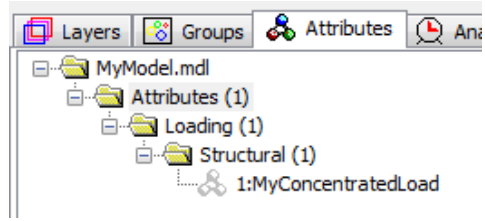
```
database.createLoadingConcentrated("MyConcentratedLoad")
```



This function creates a concentrated load attribute. The parameter it takes is the name of the load attribute (for example: MyConcentratedLoad), and it returns an object of the class ‘Loading Concentrated’:



The corresponding entry in the Attributes Treeview is shown as follows:



At the moment this load has a value of 0 for all its components. If you want it to be a load of, say, 10 units in the X direction, you also need to use one of the functions of the Loading Concentrated class:

Loading Concentrated

Base Class: Loading

Derived Classes: None

Description

Concentrated structural loading attribute

Available Functions:

setConcentrated(px, py, [pz], [mx], [my], [mz], [loof1], [loof2], [pore])

```
LPI command   call database.createLoadingConcentrated("MyConcentratedLoad").setConcentrated(10.0,0.0)
```

As stated previously, the parameters in square brackets [ ] are optional, so do not need to be defined, hence just px and py are specified.

Customising the user interface

Capabilities

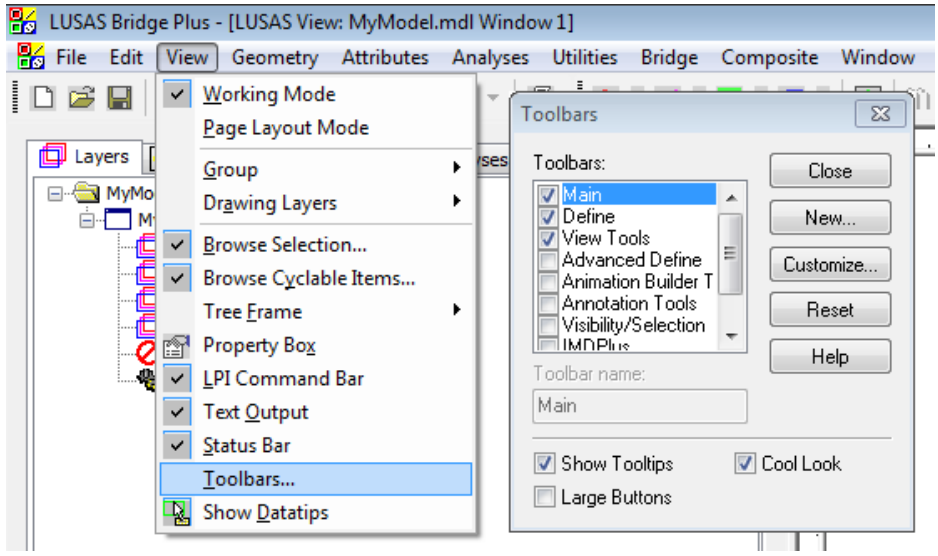
With LPI you can create user-defined menu items, dialogs and toolbars. Dialogs are covered in the LUSAS Programmable Interface (LPI) Developer Guide. This section covers customisation of:

- **Modifying standard toolbars**
- **Customised and User toolbar buttons**
- **Start-up templates**

Modifying standard toolbars

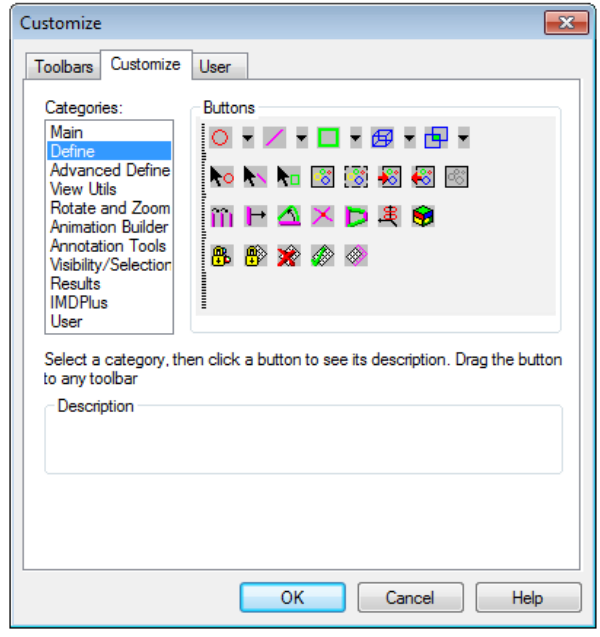
Toolbars consist of buttons which can be used to drive the software.

Within Modeller these can be customised from the **View> Toolbars** menu item.



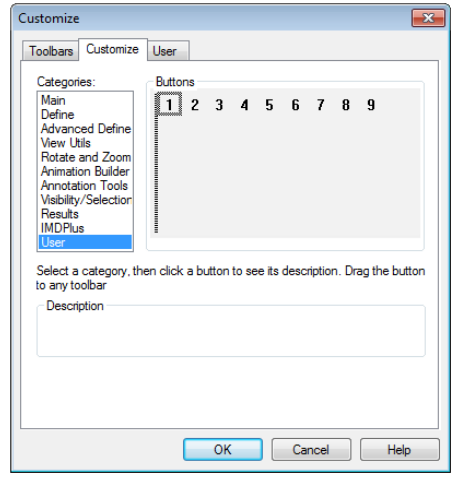
Customized toolbar buttons

Pre-programmed buttons can be added to the toolbars from the **View> Toolbars > Customize > Customize** tab, by simply dragging and dropping buttons as required.



User toolbar buttons

User toolbar buttons can be added and programmed to carry out user defined actions by navigating to selecting **View > Toolbars > Customize > Customize tab** and then selecting **User** from the Categories list



The bitmaps on the toolbar buttons may be changed by modifying the file **C:\<LUSAS Installation Folder>Programs\Config\userToolbar.bmp**

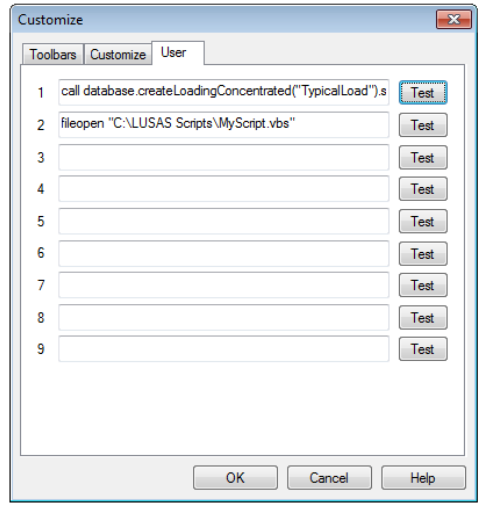
Calling functions from user buttons:

The actions carried out when a button on a user tool bar is chosen are defined on the **View > Toolbars > Customize > User** dialog.

For example, if you often need to define a concentrated load of, say, 10 in X, 20 in Y, and 30 in Z, you can type the LPI function in the user button 1 text box, so that every time that button was selected, that load attribute would be created in the Attribute Treeview:

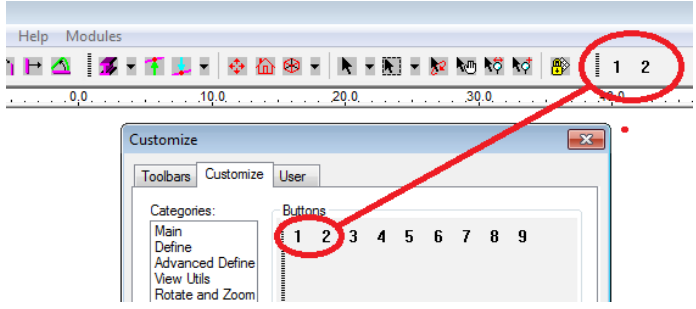
```
Call database.createLoadingConcentrated("TypicalLoad").set

Concentrated(10,20,30,0,0,0,0,0.0)
```

Or, if you often want to run a script called MyScript.vbs, then you would type: **fileopen "C:\LUSAS Scripts\MyScript.vbs"** user button 2 text box as shown below:



Adding User buttons to toolbar menus

User buttons can be added to the toolbar menu by dragging and dropping into place.

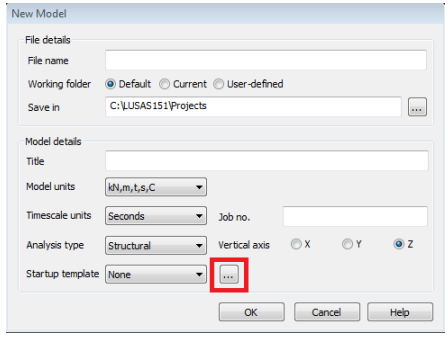


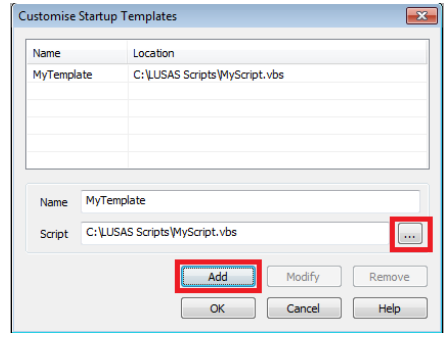
Startup templates

Startup templates can be used to pre-load the Attributes Treeview with selected attributes for a particular analysis, set default mesh or material types, or define preferred colour schemes - to name just a few uses.

User-defined startup templates are created by recording the setting of a variety of selections and then associating the recording with a template name.

You can use any VBS file as a template, and you can also add templates from the “New Model” form:





Now every time that you create a new blank model, you will be able to choose this template, which will run the script just after creating the model.

Advanced operations

For more advanced operations a macro facility is available to enable commonly used commands to be grouped together or abbreviated.

Macro functions should be written in Visual Basic and saved in a file. For example:

```
sub create_line(x1,y1,z1,x2,y2,z2)

    set geometry_data = geometryData().setAllDefaults()
    call geometry_data.setLowerOrderGeometryType("coordinates")
    call geometry_data.setCreateMethod("straight")
    call geometry_data.addCoords(x1, y1, z1)
    call geometry_data.addCoords(x2, y2, z2)
    call database.createLine(geometry_data)

end sub
```

The macro file is registered from the **Advanced** button on the LPI command bar.

The functions in the macro file may then be activated from the LPI command bar by typing the function name and arguments e.g. **create_line 1,2,3,4,5,6** or **call create_line(1,2,3,4,5,6)**

Getting started with VBS

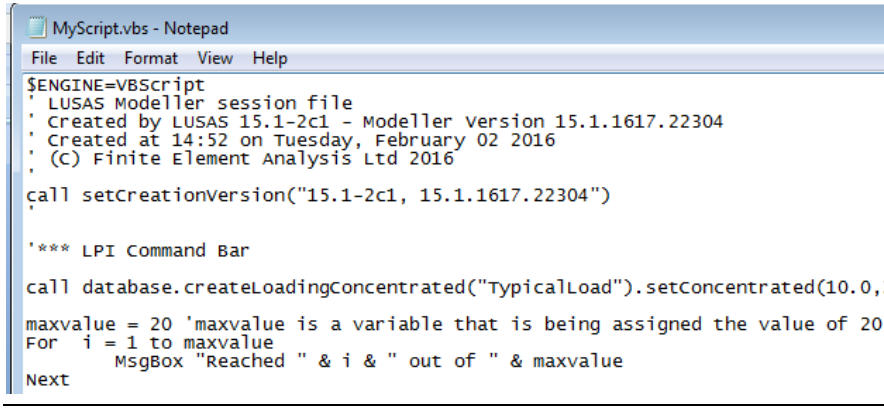
LUSAS Modeller records every operation that it carries out in a session file. This file contains standard calls to LUSAS LPI function in a Visual Basic Script syntax. The file can be replayed to carry out exactly the same actions again. Alternatively the file can be modified to carry out different actions.

The procedure is as follows:

1. To start by recording a script select **File > Script > Start Recording...**
2. Carry out a series of operations.
3. Stop recording by selecting **File > Script > Stop Recording**
4. Edit the .vbs file (.lvb file from version 20 onwards) to cover the cases required.

With a little programming syntax knowledge **loops** can be used to make the script more "tidy" and **variables** can be added to make the script more "flexible"

Editing can be carried out with the standard Windows Notepad (accessible from **Start> All Programs> Accessories> Notepad**) or 3rd party products such as Notepad++ or VSCode.



Programming syntax

Some simple rules

- First line of the visual basic script file must be **$ENGINE=VBSCRIPT**
- Lines to be treated as comments only must start with an apostrophe (‘)

Basic Operators allowed include:

- Arithmetic: +, -, /, *
- Comparison: =, >, <, >=, <=, <>
- Concatenation: &
- Logical: Not, And, Or

Conditionals

- If ... Then ... Else

```
If a > b Then
    MsgBox "a was greater than b"
Else
    MsgBox "a was not greater than b"
End If
```

- Loops: For ... Next

```
maxvalue = 20 ' max value is a variable that is assigned the value of 20
For i = 1 to maxvalue
    MsgBox "Reached " & i & "out of " & maxvalue
Next
```

Variables

- Can be strings, numbers etc
- Names must begin with a letter
- Names must not contain an embedded full-stop (period) “.”
- Names must not exceed 255 characters
- Names must be unique
- There is no need to “declare” variables

Arrays

- Can contain strings or numbers etc
- Can be “called” individually
- Always use (0) as the first index of the array.

Example:

```
Dim MyArray(2)
    MyArray(0) = 10
    MyArray(1) = 20
    MyArray(2) = 30
```

This is a one dimensional array with 3 items. The first element has been assigned a value of 10, the second 20, and the third 30.

Note that the **UBound** function returns the largest available subscript of an array:

```
call msgbox("Upper bound of array=" & UBound(MyArray))
```

So the UBound function returns 2 for the array in the example above.

Visual Basic Script online tutorials

More detailed online tutorials showing how to write visual basic script can be found on the internet. Here are just a few examples (with links applicable at the time of writing):

- English: http://www.tizag.com/vbscriptTutorial/
- English: http://www.tutorialspoint.com/vbscript/index.htm

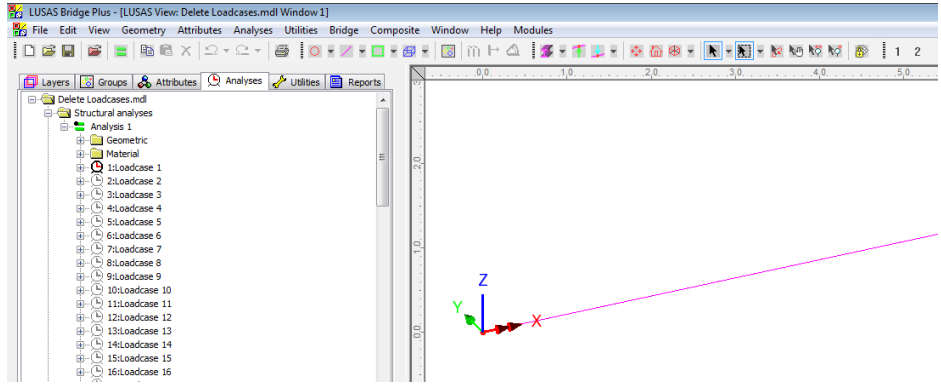
Example VB scripts

Simple user script example

Deleting a range of loadcases

A simple user script is to be written to illustrate how a range of loadcases can be deleted from the Analyses Treeview.

In Version 15.2 of LUSAS the only way to delete loadcases via the user interface was to click on each of them in turn in the Analyses Treeview and press the delete key. In later versions of LUSAS this is no longer the case, and loadcases can be deleted from the Analyses Treeview singly or by multiple selection, so the script example does not provide any more functionality for these versions. However, it does provide a good introduction to generating user scripts.



First make a recording

1. Choose **File > Start Recording...**
2. Specify a file name
3. Delete **loadcase 1** manually.
4. Choose **File > Stop Recording**
5. Open the script file created:

```
1  $ENGINE=VBScript
2  ' LUSAS Modeller session file
3  ' Created by LUSAS 15.1-2c1 - Modeller Version 15.1.1617.22304
4  ' Created at 09:21 on Wednesday, February 03 2016
5  ' (C) Finite Element Analysis Ltd 2016
6  '
7  call setCreationVersion("15.1-2c1, 15.1.1617.22304")
8  '
9
10 '*** Delete loadcase/control
11
12 call database.deleteLoadset("Loadcase 1")
```

In this file note that:

- Line 1: This line is common in all scripts. Do not remove or modify this line.
- Line 7: Specifies the version of LUSAS used to generate the script. This line is common in all scripts. Do not remove or modify this line.
- Line 12: This is the line that deletes Loadcase 1
- Lines other than those above: These lines are comments: they are ignored. Comments always start with an apostrophe (‘)

If the Loadcase names are of the form Loadcase 1, Loadcase 2 etc and you wanted to delete Loadcase 2 to Loadcase 50 you need to edit the previous script and insert a **For ... Next** loop as seen at the bottom of this next image:

```
1  $ENGINE=VBScript
2  ' LUSAS Modeller session file
3  ' Created by LUSAS 15.1-2c1 - Modeller Version 15.1.1617.22304
4  ' Created at 09:21 on Wednesday, February 03 2016
5  ' (C) Finite Element Analysis Ltd 2016
6  '
7  call setCreationVersion("15.1-2c1, 15.1.1617.22304")
8  '
9
10 '*** Delete loadcase/control
11
12 For i = 2 To 50
13     call database.deleteLoadset("Loadcase " & i)
14 Next
```

Now you are actually calling the deleteLoadset function 49 times, taking the argument Loadcase 2, Loadcase 3, Loadcase 4, etc.

Note. The function **deleteLoadset** is used instead of **deleteLoadcase** because the deleteLoadset function also deletes combinations and envelopes.

If you look for information about this function in the LPI online help, you will see that the loadset can be specified by Loadset name (which is how it has been done in this example) but it can also be specified by Loadset ID.

Database.deleteLoadset

deleteLoadset(loadset)

deleteLoadset(name, [resFile], [eigen], [harm])

deleteLoadset(ID, [resFile], [eigen], [harm])

Delete the specified loadset. Note that it is not possible to delete results loadcases (close the file instead) or the last remaining pre-processing loadcase. The loadset can be specified in several ways, by name, by ID or by type and name/ID. In each case, additionally specifying the results file name/ID, eigenvalue ID and harmonic ID will clarify to LUSAS which loadset is required. Alternatively, an object may be passed in, which requires no further clarification. This same principle applies to all functions that input single loadsets, and the examples below reflect this. Each input form is legal in each circumstance

By ID it would be easier to write the script as follows:

```
12  For i = 2 To 50
13      call database.deleteLoadset(i)
14  Next
```

And if in the original model you wanted to delete all the even loadcases you would add ‘Step 2’ to line 12:

```
12  For i = 2 to 100 Step 2
13      call database.deleteLoadset(i)
14  Next
```

Alternatively if you wanted to delete all the even-numbered loadcases you could append ‘2T50I2’) to the main LPI command:

call database.deleteLoadsets(2T50I2)

This would delete from Loadcase 2 to Loadcase 50 in increments of 2.

Save the file as delete_loadcases.vbs

Running a script

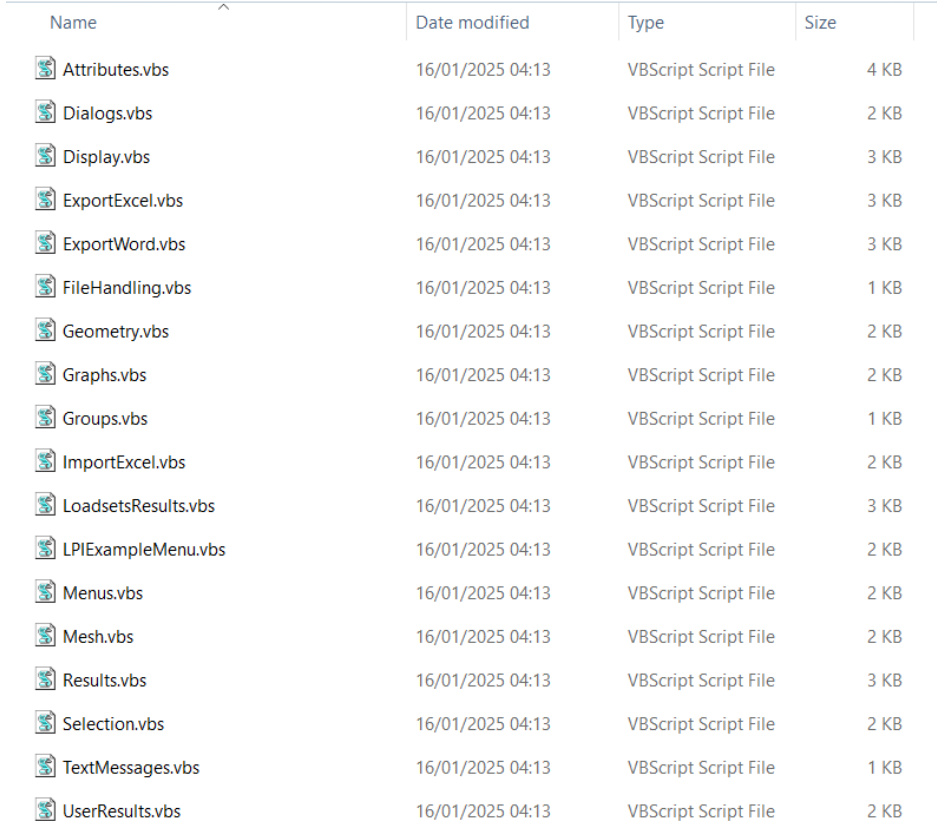
A script can be run within LUSAS Modeller as follows:

1. Choose **File > Script > Run Script**
2. Browse for and select **<script_name.vbs>**

Supplied script examples

LUSAS supplies many script examples (that are installed as part of a software installation) which demonstrate how to carry out various functions and tasks. These may be found at this location:

C:\<LUSAS Installation Folder>\Programs (x86)\Scripts\LPIExamples



Name	Date modified	Type	Size
Attributes.vbs	16/01/2025 04:13	VBScript Script File	4 KB
Dialogs.vbs	16/01/2025 04:13	VBScript Script File	2 KB
Display.vbs	16/01/2025 04:13	VBScript Script File	3 KB
ExportExcel.vbs	16/01/2025 04:13	VBScript Script File	3 KB
ExportWord.vbs	16/01/2025 04:13	VBScript Script File	3 KB
FileHandling.vbs	16/01/2025 04:13	VBScript Script File	1 KB
Geometry.vbs	16/01/2025 04:13	VBScript Script File	2 KB
Graphs.vbs	16/01/2025 04:13	VBScript Script File	2 KB
Groups.vbs	16/01/2025 04:13	VBScript Script File	1 KB
ImportExcel.vbs	16/01/2025 04:13	VBScript Script File	2 KB
LoadsetsResults.vbs	16/01/2025 04:13	VBScript Script File	3 KB
LPIExampleMenu.vbs	16/01/2025 04:13	VBScript Script File	2 KB
Menus.vbs	16/01/2025 04:13	VBScript Script File	2 KB
Mesh.vbs	16/01/2025 04:13	VBScript Script File	2 KB
Results.vbs	16/01/2025 04:13	VBScript Script File	3 KB
Selection.vbs	16/01/2025 04:13	VBScript Script File	2 KB
TextMessages.vbs	16/01/2025 04:13	VBScript Script File	1 KB
UserResults.vbs	16/01/2025 04:13	VBScript Script File	2 KB

Note. In the printed versions of the two scripts that follow word wrapping has taken place. Only lines that are preceded by an apostrophe (‘) are comment lines. Other lines containing VB script should not be word-wrapped.

Example script: Attributes.vbs

This supplied script creates a single planar surface and then creates and assigns to that surface: a regular mesh, material and geometry. A support is created and assigned to a line; a concentrated load is created and assigned to a point as loadcase 1; and a face load is created and assigned to a line as loadcase 2.

```
$ENGINE=VBScript
' Create and assign attributes
'-----------------------------------------------------------------
' Create new database
call newdatabase()

' Create Surface
call geometryData.setAllDefaults()
call geometryData.setCreateMethod("planar")
call geometryData.addCoords(0.0, 0.0, 0.0)
call geometryData.addCoords(40.0, 0.0, 0.0)
call geometryData.addCoords(40.0, 20.0, 0.0)
call geometryData.addCoords(0.0, 20.0, 0.0)
call database.createSurface(geometryData)

' Create Attribute : Surface Mesh 1
call database.createMeshSurface("Plane Stress").setRegular("QPM8",
0, 0, false)

' Modify selection
call selection.add("Surface", "1")

' Attribute : Plane Stress : Assign to Primary selection :
call assignment.setAllDefaults().setLoadset("Loadcase 1")
call database.getAttribute("Mesh", "Plane
Stress").assignTo(selection, assignment)
call database.updateMesh()
```

```
' Create Attribute : Isotropic Material 1
set attr = database.createIsotropicMaterial("Mild Steel", 200.0E3,
0.3, 7.8E3)
set attr = nothing

' Attribute : Mild Steel : Assign to Primary selection :
call assignment.setAllDefaults()
call database.getAttribute("Material", "Mild
Steel").assignTo(selection, assignment)

' Create Attribute : Surface Geometric 1
call database.createGeometricSurface("Thickness=1").setSurface(1.0,
0.0)

' Attribute : Thickness=1 : Assign to Primary selection :
call assignment.setAllDefaults()
call database.getAttribute("Geometric",
"Thickness=1").assignTo(selection, assignment)

' Create Attribute : Fixed in XY
call database.createSupportStructural("Fixed in
XY").setStructural("R", "R")

' Modify selection
call selection.add("Line", "4")

' Attribute : Fixed in XY : Assign to Primary selection :
call
assignment.setAllDefaults().setSelectionNone().addToSelection("Line"
)
call database.getAttribute("Supports", "Fixed in
XY").assignTo(selection, assignment)

' Create Attribute : Concentrated Load 1
call database.createLoadingConcentrated("Concentrated Load
1").setConcentrated(0.0, -100.0)
```

```
' Modify selection
call selection.add("Point", "3")

' Attribute : Concentrated Load 1 : Assign to Primary selection :
call
assignment.setAllDefaults().setSelectionNone().addToSelection("Point
").setLoadset("Loadcase 1")
' assign load to selected point in loadcase 1
call database.getAttribute("Loading", "Concentrated Load
1").assignTo(selection, assignment)

' Define face load
call database.createLoadingFace("Distributed 1").setFace(0.0, 10.0,
0.0)
' Select top face
call selection.remove("All")
call selection.add("Line", "3")

' Create new loadcase and set active
call database.createLoadcase("Loadcase 2", "Structural")
set loadset = database.getLoadset("Loadcase 2", "model")
call view.setActiveLoadset(loadset)
set loadset = nothing

' set assignment object with selected face hof and loadcase 2
set hof0 = database.getObject("Surface", "1")
call assignment.setAllDefaults().setLoadset("Loadcase
2").addHof(hof0)
' assign face load to top face in loadcase 2
call database.getAttribute("Loading", "Distributed
1").assignTo(selection, assignment)
```

Example script: Results.vbs

This supplied script writes a set of results to the text window for a prior selection of features made in the Modeller view window.

```
$ENGINE=VBScript
' Extracting Results
'-----------------------------------------------------------------
  set textWindow = getTextWindow()
' set results type
  entity="Displacement"
' get array of results component names
  component=view.getResultsComponentNames(entity)
' extract array of selected nodes
  nodes=selection.getObjects("Node","All")
' check nodes in selection
  if ubound(nodes) >= 0 then
' loop selected nodes
    for i = 0 to ubound(nodes)
' get node object
     set node = nodes(i)
' get element Number
     num=node.getID()
' write line to text window
     text = "Node=" & num
     for j=0 to ubound(component)
' get averaged nodal result
        res=node.getResults(entity,component(j))
        text=text & " " & component(j) & " = " & res
     next
     textWindow.writeLine(text)
    next
' set results type
    entity ="Force/Moment - Thick Shell"
' get array of results component names
    component=view.getResultsComponentNames(entity)
' extract array of selected elements
```

```
    elements= selection.getObjects("Element","All")
' loop selected elements
    for i = 0 to ubound(elements)
' set element object
      set elt = elements(i)
' get element Number
      num=elt.getID()
' extract array of element nodes
      nodes = elt.getNodes()
' loop element nodes
      for k = 0 to ubound(nodes)
' get node object
        set node = nodes(k)
' extract node data
        nnum = node.getID()
        node.getXYZ x,y,z
' get vector of element nodal results
        vecRes=elt.getNodeVectorResults(k,entity)
        nc=ubound(vecRes)
' build output text string
        text = "Elt=" & num & " Node=" & nnum & " x=" & x & " y=" & y & "
z=" & z
        for j = 0 to nc
           text=text & " " & component(j) & " = " & vecRes(j)
        next
        textWindow.writeLine(text)
      next

' extract number of Gauss points
      ngp = elt.countGaussPoints()
' loop gauss points
      for k = 0 to ngp-1
        vecRes = elt.getGaussVectorResults(k,entity)
        nc=ubound(vecRes)
' build output text string
        text = "Elt=" & num & " GP=" & k
```

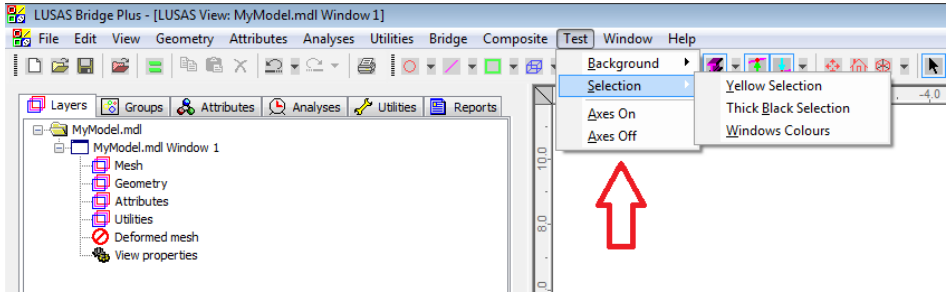
```
        for j = 0 to nc
           text=text & " " & component(j) & " = " & vecRes(j)
        next
        textWindow.writeLine(text)
      next
    next
  else
    AfxMsgBox "Run an analysis and select some nodes as input to this
script"
  end if
```

Running a script from a menu

When a number of related scripts have been created it is often more convenient to add a Modeller menu item to access those scripts, rather than by opening the scripts using the **File > Script > Run Script** menu item.

To allow this, one supplied script contains the code to create a menu containing all of the other supplied scripts. To add the menu item to Modeller's main menu:

1. Choose **File > Script > Run Script**
2. Browse to the **C:\<LUSAS Installation Folder>\Programs (x86)\scripts\LPIExamples folder**
3. Select **LPIExamplesMenu.vbs**



This menu script example adds a Test menu name to the main menu, and has a number of menu items with sub-menus that each trigger a script.

Keeping the LPI menu visible

To keep the LPI menu visible:

1. Open the file **C:\<LUSAS Installation Folder>\Programs (x86)\Config\afterNewModel.vbs**
2. Add these lines to the bottom of the file:

```
Scripts= getSystemString("scripts")

call fileopen(scripts&"\LPIExamples\LPIExampleMenu.vbs")
```

Adding a user menu

A user menu can be added by editing this file in your user folder:

"%USERPROFILE%\Documents\LUSAS210\UserScripts\" -> C:\User\<Your username>\Documents\LUSAS210\UserScripts\Usermenu.vbs

More advanced scripts

Many more advanced scripts can be downloaded from Lusas GitHub repository:

https://github.com/LUSAS-Software/LUSAS-API-Examples

Using Python

Python has become the scripting language of choice for the scientific and engineering communities due to its simplicity, extensive libraries and useful development tools such as debuggers. Setting up Python for automating workflows in LUSAS is a little more advanced than using VBScript and this is due to the active development of Python and its libraries resulting in many different versions. Unlike running VBScripts "Inside" LUSAS Modeller, Python can be run "Outside" of LUSAS Modeller sending commands to LUSAS Modeller LPI via the COM.

To use Python to drive LUSAS Modeller externally, the following steps are required:

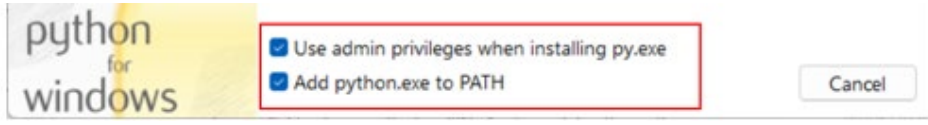
- Install Python
- Install Python package pywin32 version 308
- Install VSCode (Technically this step is not required but it is recommended to make developing and running scripts much simpler)

Install Python

There are many versions of Python and there are many versions of extension modules that provide a Python "Environment". Here we will show the most straightforward route to getting started but more experienced users may choose to handle installations through management systems such as 'conda' or 'miniforge'.

Download and install the version of Python you wish to use from https://www.python.org/downloads/windows/

It is recommended to install both 64-Bit and 32-Bit versions of Python and in both cases include the options to use admin privileges for py.exe and add python.exe to the PATH. Note the default version will be the last one installed, so it is recommended to install 32-Bit first.



Note. It is possible to download and use a Python environment from your file system without installing. https://winpython.sourceforge.net/ provides packages containing additional tools such as Spyder IDE and lots of default packages that are useful. This is left for the advanced user.

Install package pywin32

To install the pywin32 packages for both 32 and 64-Bit versions open a command prompt window and paste in the following instructions, taking care to change the version numbers to the version of Python you installed.

```
set version=3.13

Rem install pywin for 64Bit application
py -V:%version% -m pip install --upgrade pywin32==308

Rem install pywin for 32Bit application
py -V:%version%-32 -m pip install --upgrade pywin32==308
```

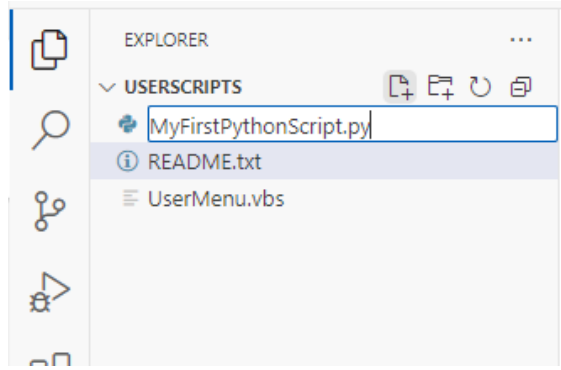
Note. The pywin32 package is under active development and it has been found that some versions will not work with LUSAS. Here the specific version 308 is installed which is known to work.

Install VSCode

The installation of VSCode is not technically required to run Python scripts with LUSAS. Advanced users can simply write scripts and execute them with the Python executable. However, it is recommended to use VSCode which is free and open source and offers lots of useful features. Download and install VSCode from here: https://code.visualstudio.com/download

Once installed, run VSCode and select “Open a folder” from the start page, navigate to Documents/Lusas230/UserScripts (or any other folder you choose to contain your scripts)

In the explorer view, add a new file and give it a name with a .py file extension



Double click the file to edit it.

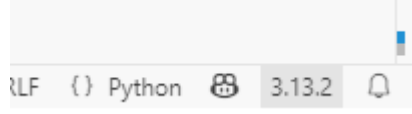
Note. You may notice VSCode offering to install extensions for Python development. It is recommended that you install these tools which will help writing scripts offering syntax highlighting and code completion.

Write the following code in your file. Note that if you are using other versions of LUSAS Modeller the correct version number should be included

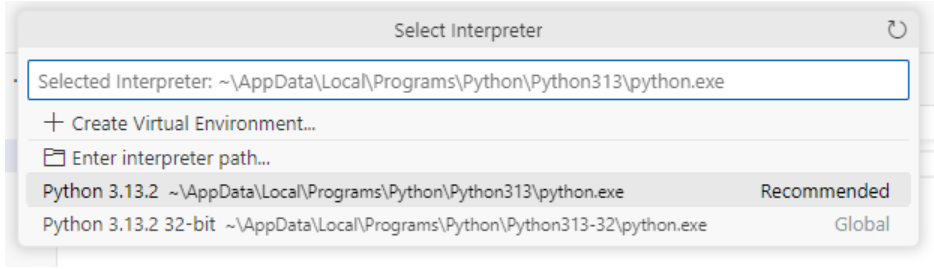
```
import win32com.client as win32
modeller = win32.dynamic.Dispatch("Lusas.Modeller.24.0")

modeller.AfxMsgBox("Test")
```

Note. If this code is run without LUSAS Modeller running, the code will start a version in the background if it can and this will consume one of your licences. You can check in the task manager if any versions of LUSAS Modeller are running that you didn't intend to run and close them there. Before running this code, it is recommended that you start the correct version of LUSAS Modeller you which to use and that way you will be able to see the effects of the script being executed.

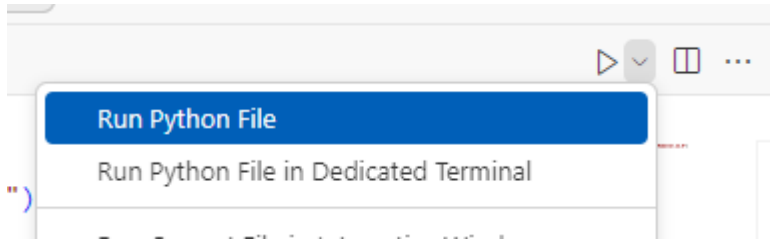
In the bottom right corner of VSCode the Python interpreter which will be used to run your code should be listed. If you do not see a version number here, select it to display a list of available interpreters to use.



Then choose the interpreter to use from the displayed list. The 64-Bit version is required to run with LUSAS Modeller 64-Bit.



Now, with LUSAS Modeller running and the correct Python interpreter selected, go to the top right-hand corner of VSCode, select the “play” button and choose “Run Python File”



You should see a message box displaying your text. If not carefully review the steps again, check that the LUSAS Modeller version number matches the intended use, check in the task manager that there are no hidden LUSAS Modeller’s created by mistake and try again.

Once you see the message box you are ready to automate your workflows with the power of Python and LUSAS combined.

Many examples of the use of the LPI with Python are provided in the LUSAS repository on GitHub. https://github.com/LUSAS-Software/LUSAS-API-Examples

Note. It is recommended to use win32.dynamic.Dispatch method to connect to LUSAS Modeller however is also possible to use win32.gencache.EnsureDispatch which will create a cached Python wrapper of all the LPI functions to improve performance. Unfortunately, this cache can sometimes fail to locate the correct functions and so the dynamic method is preferred. If the gencache method is used and errors occur, they can often be resolved by deleting the cache at **%localAppData%\Temp\gen_py** and allowing it to be rebuilt automatically on the next run.

Performance

Executing a Python script externally typically runs more slowly than a VBScript run internally. This is because LUSAS Modeller can perform optimisations when it knows a whole script is running. There are several things that can be done to improve performance of external scripts however as follows:

Disable the UI

To improve the speed of the Python scripts running externally you can disable the user interface as follows:

```
6
7   if modeller.getMajorVersionNumber() >= 22:
8       modeller.enableUI(False)
9   else:
10      modeller.setVisible(False)
11
12  # Script body
13
14
15  if modeller.getMajorVersionNumber() >= 22:
16      modeller.enableUI(True)
17  else:
18      modeller.setVisible(True)
```

Note that prior to version 22 the UI must be set to be invisible, but in version 22 onwards it can simply be disabled. Care should be taken here however when a script fails before completion the user interface will remain invisible or disabled, and another script will need to be executed containing just the bottom lines to restore the UI.

Batch commands

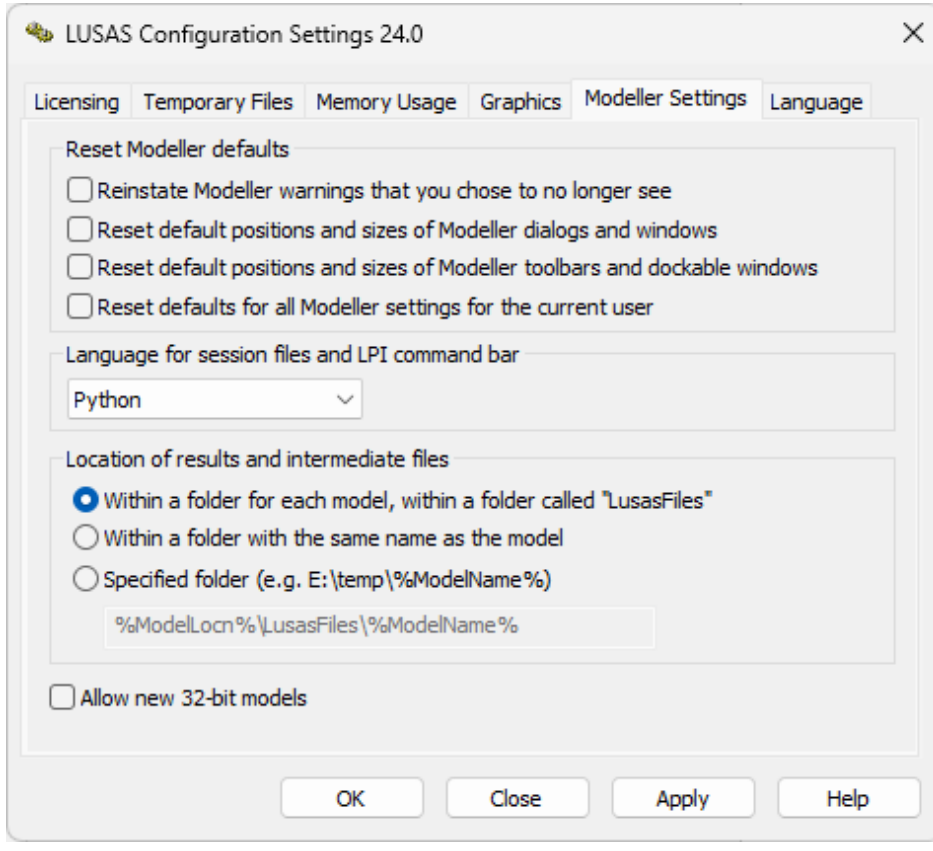
Wrapping your code between calls to begin and close the command batch will tell LUSAS Modeller that you intend to issue lots of commands and so it will not perform any updates until you have finished. This also allows the actions of your script to be undone.

```
modeller.getDatabase().beginCommandBatch("My Commands")

# Script body

modeller.getDatabase().beginCommandBatch()
```

Python session files

LUSAS Version 23.0 and later releases come with a custom version of Python which can be used to generate session files and run Python scripts “inside” Lusas Modeller.

To generate session files and display LPI commands in Python, go to the LUSAS Configuration Utility and select Python from Language option.



The Python version supplied with LUSAS is separate from any installed versions as described previously. It is recommended that an installed version of Python is used such that additional Python packages can be added and removed as required.

Python IntelliSense

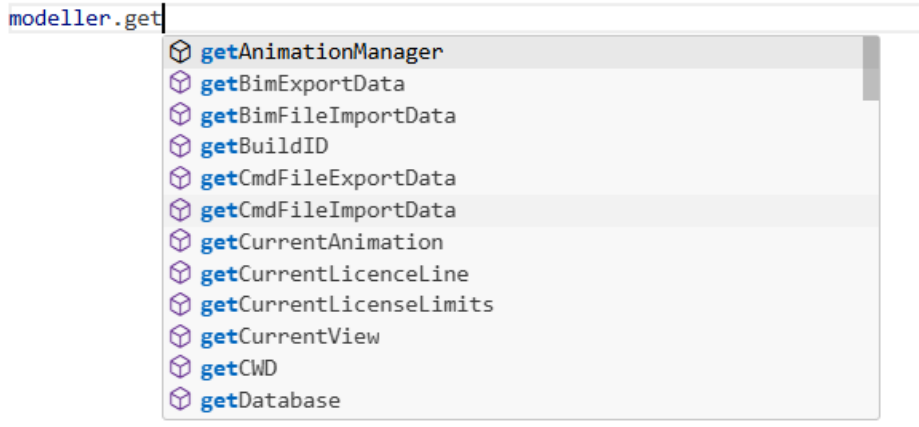
When using an IDE such as VSCode, the editor will provide auto complete and inline help for function names and arguments as you type. To enable inline help for LPI functions the LPI.py module can be imported and used.

Copy the file **C:\Program Files\LUSAS240\Programs\scripts\Python\LPI.py** to the folder containing your Python script.

You can now use the following to connect to LUSAS Modeller (Note that this will connect to the version of LUSAS that it was installed for.)

```
from LPI import*
modeller = get_lusas_modeller()
```

As you type VSCode will display lists of available LPI functions and help on their use



There are many examples of using this LPI with this approach on our GitHub account.

https://github.com/LUSAS-Software/LUSAS-API-Examples

LUSAS

LUSAS, Forge House, 66 High Street, Kingston upon Thames, Surrey, KT1 1HN, UK
Tel: +44 (0)20 8541 1999 | Fax: +44 (0)20 8549 9399 | info@lusas.com | www.lusas.com