NextFEM

Designer
Developer's manual

For version 2.0 and above

Rev. G

Index

1 Plugin basic interface

NextFEM Designer can host plugins to interact with model and viewport. Plugins must be written in any of the **.NET Framework** supported languages (eg. C#, VB.NET, F#, …). Advised version is v4.7.1.

NextFEM gives the public interface ***IPlugin*** in *NextFEMplugin.dll*, that must be implemented inside the plugin you're developing. In addition, the plugin must reference *NextFEMapi.dll*. Both *NextFEMapi.dll* and *NextFEMplugin.dll* can be found in the installation folder of NextFEM Designer, typically *C:\Program Files\NextFEM\NextFEM Designer 64bit* for 64bit installations.
Both references must have the *Copy locally* flag set to False in the Visual Studio reference properties. To be found by the plugin, please insert the following code in your *app.config*.

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <probing privatePath=".." />
  </assemblyBinding>
</runtime>
```

Implement a **base class** like the samples below.

1.1 C# language – Iplugin interface

```
using NextFEMplugin;

public class plugin : IPlugin
{
        public void LaunchMe(IntPtr hnd)
        {
                TrussDrawer frm = new TrussDrawer();
                frm.caller = this;
                frm.Text = this.PluginName;
                frm.Show(System.Windows.Forms.Control.FromHandle(hnd));
        }

        public string PluginAuthor
        {
                get
                { return "NextFEM Support Team"; }
        }

        public string PluginName
        {
                get
                { return "Truss Drawer"; }
        }

        public string iconFile
        {
                get
                { return "NextFEM.ico"; }
        }

        public event EventHandler updateModel;
        internal void RequestScreenUpdate(object sender, bool resize, ViewState Vstate)
        {
                updateModel(sender, resize, Vstate);
        }
        public event EventHandler undoCall;
        internal void RequestUndo(undoOps undoProperty)
```

```
        {
                undoCall(undoProperty);
        }

        public NextFEMapi.API API {get; set;}
        public string currentLoadcase {get; set;}
        public string currentTime {get; set;}
}
```

1.2 VB.NET language – *Iplugin interface*

```
Imports NextFEMplugin

Public Class plugin
    Implements IPlugin

    Public Sub LaunchMe(hnd As IntPtr) Implements IPlugin.LaunchMe
        Dim frm As New TrussDrawer
        frm.caller = Me
        frm.Text = Me.PluginName
        frm.Show(System.Windows.Forms.Control.FromHandle(hnd))
    End Sub

    Public ReadOnly Property PluginAuthor As String Implements IPlugin.PluginAuthor
        Get
            Return "NextFEM Support Team"
        End Get
    End Property

    Public ReadOnly Property PluginName As String Implements IPlugin.PluginName
        Get
            Return "Truss Drawer"
        End Get
    End Property

    Public ReadOnly Property iconFile As String Implements IPlugin.iconFile
        Get
            Return "NextFEM.ico"
        End Get
    End Property

    Public Event updateModel(sender As Object, resize As Boolean, Vstate As ViewState) Implements
IPlugin.updateModel
    Friend Sub RequestScreenUpdate(sender As Object, resize As Boolean, Vstate As ViewState)
        RaiseEvent updateModel(sender, resize, Vstate)
    End Sub
    Public Event undoCall(undoProperty As undoOps) Implements IPlugin.undoCall
    Friend Sub RequestUndo(undoProperty As undoOps)
        RaiseEvent undoCall(undoProperty)
    End Sub

    Public Property API As NextFEMapi.API Implements IPlugin.API
    Public Property currentLoadcase As String Implements IPlugin.currentLoadcase
    Public Property currentTime As String Implements IPlugin.currentTime
End Class
```

Implementation is discussed on VB.NET code. Below all the fields of the interface are described:

- **PluginName** : name of the plugin as string. The same name can be used to call the plugin by starting Designer via command-line. Eg. *designer64.exe –plugin "Truss Drawer"*
- **PluginAuthor** : string for the author of the plugin;
- **iconFile** : it sets the filename of the image to be loaded into the *Tools/Plugins* menu in NextFEM Designer. This is optional, if not used please return an empty string.

- **LaunchMe** : procedure for starting the plugin application (eg. a form). It must initialize NextFEM API (otherwise) and pass the current instance of the base class to the plugin application form. In the sample above, it instantiates and launches the TrussDrawer class. The variable *caller* is declared in the form as: `Friend caller as plugin`
  As final step, this procedure launched the form, passing the handle to be treated as Designer mask: `frm.Show(System.Windows.Forms.Control.FromHandle(hnd))`
- **updateModel** : event to be called for requesting a viewport update. Since events can be called only in the same class, the function *RequestScreenUpdate* is implemented and has to be called from the plugin app form.
- **undoCall** : it follows the same working mechanism as *updateModel*.
- **API :** an instance of NextFEM API class, retaining all the methods to operate in model and results. You don't need to implement it in your base class.
- **currentLoadCase :** string indicating the loadcase showed on screen for loading or results views.
- **currentTime :** string indicating the time or the mode showed on screen for results view.

Plugins can operate in every way in the model. There are some good practice rules to ensure their optimal functionality:

1) to load other DLLs in your plugin, please add the following code inside the form code:

```
    ' for loading dlls in the same folder - IMPORTANT: must be Private!
    Private currentDomain As AppDomain = AppDomain.CurrentDomain
    Private Function MyResolveEventHandler(ByVal sender As Object, ByVal args As ResolveEventArgs) As
System.Reflection.Assembly
        ' this handler is called only when the CLR tries to bind to the assembly and fails
        Dim strTempAssmbPath As String = ""
        Dim MyAssembly, objExecutingAssembly As System.Reflection.Assembly
        objExecutingAssembly = System.Reflection.Assembly.GetExecutingAssembly()
        ' seek in the same folder
        Dim name As String = args.Name.Substring(0, args.Name.IndexOf(","))
        strTempAssmbPath = My.Application.Info.DirectoryPath & "\plugins\" & name & ".dll"
        If IO.File.Exists(strTempAssmbPath) = False Then Return Nothing
        ' load the assembly from the specified path
        MyAssembly = System.Reflection.Assembly.LoadFrom(strTempAssmbPath)

        ' return the loaded assembly
        Return MyAssembly
    End Function
```

In addition, a handler to this event must be added to the *MyBase.Load* procedure:

```
        ' use DLL loading for load assemblied from the same plugin folder (or set subfolder in MyResolveEventHandler)
        AddHandler currentDomain.AssemblyResolve, AddressOf MyResolveEventHandler
```

Remember, in *MyBase.Closing*, to remove the handler:

```
         RemoveHandler currentDomain.AssemblyResolve, AddressOf MyResolveEventHandler
```

Complex plugins with many DLLs may use a subdirectory of "plugins" folder.

2) Every operation on the model have to be preceded by an "*Undo request*" to Designer.

```
        ' call undo
        caller.RequestUndo(NextFEMplugin.undoProps.Normal)
```

and followed by a request for updating screen:

```
        ' refresh the view
        caller.RequestScreenUpdate(caller, True, NextFEMplugin.ViewState.NoOperation)
```

3) to update the viewport in Designer, you may choose between a variety of commands, all put together in the ViewState enumerator.

```
Eg.: caller.RequestScreenUpdate(caller, False, NextFEMplugin.ViewState.NoOperation)
```

This line request to Designer a screen update, *resize* is set to false hence not "View all" operation is called, and the ViewState is set to *NoOperation*, that means no operation required.

Here you are a list of visual operations you can call:

ViewState enumerator	**Action and remarks**
Reset	Viewport is reset to default (no results and no loading views)
NoOperation	No operation is performed, leave viewport as it is.
NodesVisible	Show nodes. Recall it to disable visible nodes.
NodesNumber	Show nodes numbers. Recall it to disable view.
ElementsNumber	Show elements numbers. Recall it to disable view.
ExtrudedView	Show extruded views. Recall it to disable view.
ColorElements_Uniform	Show uniform color in elements. Call Reset to disable view.
ColorElements_BySection	Color elements by sections. Call Reset to disable view.
ColorElements_ByMaterial	Color elements by materials. Call Reset to disable view.
ColorElements_ByGroup	Color elements by groups. Call Reset to disable view.
ShowLocalAxes	Show element local axes. Recall it to disable view.
GetScreenshot	Get screenshot of the current view. Screenshot is stored in Clipboard.
ShowAllLoads	Show all loads in mdoel. Call reset to disable view.
ShowDisplResults	Show displacement results. Call Reset to disable view.
HighlightSelectedNodes	Highlight selected nodes. Call ClearNodesHighlight to disable view.
HighlightSelectedElems	Highlight selected element. Call ClearElementsHighlight to disable view.
ClearNodesHighlight	Clear the nodes highlighting
ClearElementsHighlight	Clear the elements highlighting
ShowNodesCheckResults	Show nodes check results. Call Reset to disable view.
ShowElementsCheckResults	Show elements check results. Call Reset to disable view.
ShowFrameForcesResults	Show frame forces results. Call Reset to disable view.
ShowReactionsResults	Show element reactions results. Call Reset to disable view.
ShowAreaForcesResults	Show area elements forces. Call Reset to disable view.
ShowSoilPressure	Show soil pressure. Call Reset to disable view.

For results and loading views, you should set also *caller.currentLoadcase* and, mostly for modal or non-linear analyses, also *caller.currentTime*.

4) prior to do any action in the model (adding nodes, element, etc.) you should request an undo operation to the Designer. Otherwise, the user cannot revert what plugin did in the model.

```
Eg.: caller.RequestUndo(NextFEMplugin.undoOps.Normal)
```

undoOps enumerator	**Action and remarks**
Normal	User will be prompted to clear results, if present.
NormalDontAsk	User will not be prompted to clear results.
NoUndo	No operation is added to undo stack.

1.3 Debugging a plugin

Finally, for debugging purposes, you can copy the whole NextFEM Designer installation directory (typically *C:\Program Files\NextFEM\NextFEM Designer 64bit* for 64bit installations) anywhere in your hard-disk, and target its subfolder "*plugins*" as output directory for your program.
IMPORTANT: remember that the "plugins" folder is not writable.
Plugins DLLs can be freely developed and distributed without including files or parts of NextFEM Designer assemblies. Always remember to distribute the DLL together with the corresponding *.dll.config* file (which contains the *app.config* copy).

Each plugin must be a DLL build on the base of the sample project below, which is a simple program to draw truss structures in Designer. You can freely use this as a base for your plugin.

1.4 Get element type

When using the method

```
getElementProperty(ID, "type")
```

to check the type of an element, you get an integer value as listed below:

```
unk = 0
line = 1
tria = 2
quad = 3
hexa = 4
wedge = 5
tetra = 6
user = 10
line3 = 20
quad8 = 21
hexa16 = 22
hexa20 = 23
tetra10 = 24
tria6 = 25
wedge15 = 26
spring2nodes = 40
```

In addition, to get the length of a beam, use

```
getElementProperty(ID, "lun")
```

In other words, this method reflects all the properties as they are reported in the XML format.

1.5 Saving plugin data

To save plugin data you can use a custom field stored just inside the model. See API reference for the following functions. The key used to store data should be the plugin name.

```
addOrModifyCustomData

getCustomData

removeCustomData
```

1.6 Resources

Download VB.NET sample project

Developers' forum (for all public enquiries)
API reference
NextFEM Github repositories

2 Advanced plugin interface

In addition to *IPlugin* interface, since v2.0 NextFEM Designer can host plugins developed with the new interfaces *IPluginWithCommands**X***, which is an inherited interface that gives a deeper control in the program.

You can choose between:

- *IPluginWithCommands* interface, which adds the ability to open Designer's tools
- *IPluginWithCommands2* interface, which inherits *IPluginWithCommands* and will be loaded into a new ribbon tab.

This interface is not fully and publicly documented, and if you're going to implement your plugin onto it, you should contact support@nextfem.it for further support.

Let's have a look to a sample VB.net code.

2.1 VB.NET language – IpluginWithCommands2 interface

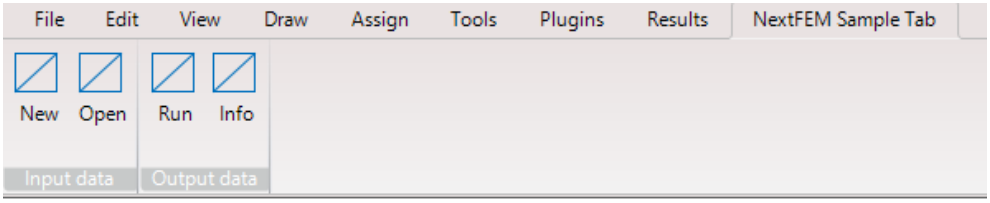
```
Imports NextFEMplugin

Public Class plugin
    Implements IPluginWithCommands2

    Dim frm As TrussDrawer ' global, as it will be used like in Winforms
    Public Sub LaunchMe(hnd As IntPtr) Implements IPlugin.LaunchMe
        ' since this is no more a single command plugin, this section is used as initialization
if tabSections returns proper buttons

        ' initilization
        frm = New TrussDrawer
        frm.caller = Me ' needed for API and model linking
        ' add function to Functions window
        RaiseEvent commandOps(Me, Commands.AddFunctionHandler, "MyFunc")
    End Sub

    Public ReadOnly Property PluginAuthor As String Implements IPlugin.PluginAuthor
        Get
            Return "NextFEM Support Team"
        End Get
    End Property

    Public ReadOnly Property PluginName As String Implements IPlugin.PluginName
        Get
            Return "NextFEM Sample Tab"
        End Get
    End Property

    Public ReadOnly Property iconFile As String Implements IPlugin.iconFile
        Get
            Return "truss.ico"
        End Get
    End Property

    Public Event updateModel(sender As Object, resize As Boolean, Vstate As ViewState) Implements
IPlugin.updateModel
    Friend Sub RequestScreenUpdate(sender As Object, resize As Boolean, Vstate As ViewState)
        RaiseEvent updateModel(sender, resize, Vstate)
    End Sub
    Public Event undoCall(undoProperty As undoOps) Implements IPlugin.undoCall
    Friend Sub RequestUndo(undoProperty As undoOps)
        RaiseEvent undoCall(undoProperty)
```

```
    End Sub
    Public Event commandOps As IPluginWithCommands.commandOpsEventHandler Implements
IPluginWithCommands2.commandOps

    Public Sub LaunchPluginCommand(hnd As IntPtr, commandName As String) Implements
IPluginWithCommands2.LaunchPluginCommand
        If frm.IsDisposed Then frm = New TrussDrawer : frm.caller = Me
        ' call single commands or single functions (added by Commands.AddFunctionHandler)
        Select Case commandName
            Case "Info"
                MsgBox("Sample plugin by " & Me.PluginAuthor)
            Case Else
                If Not frm.Visible Then frm.Show(System.Windows.Forms.Control.FromHandle(hnd))
        End Select
    End Sub

    Public Function returnData(functionName As String) As Object() Implements
IPluginWithCommands2.returnData
        returnData = {} ' default value

        ' series type enumerator:
        'Displacement_TH = 0
        'Velocity_TH = 1
        'Acceleration_TH = 2
        'Acceleration_Spectrum_TH
        'Displacement_Spectrum_TH
        'Lateral_Forces = 5
        'Vertical_Forces
        'Time_Temperature
        'Strength_Temperature
        'User = 9

        If functionName = "MyFunc" Then
            ' return list of PointF
            'MsgBox("Code for function " & functionName)
            Dim out As New List(Of Drawing.PointF)
            out.Add(New Drawing.PointF(0, 0)) : out.Add(New Drawing.PointF(1, 1))
            Return {out, 5, "kN", "custom function"}

        End If
    End Function

    Public Property currentLoadcase As String Implements IPlugin.currentLoadcase
    Public Property currentTime As String Implements IPlugin.currentTime
    Public Property API As NextFEMapi.API Implements IPlugin.API
    ' this is called only once, so the Get part can be used as initilization
    Public ReadOnly Property tabSections As List(Of tabSection) Implements
IPluginWithCommands2.tabSections
        Get
            'Return Nothing ' default value
            ' set ribbon buttons
            Dim out As New List(Of NextFEMplugin.tabSection)
            ' first section
            Dim s1 As New tabSection With {.tabName = "Input data"}
            s1.tabCommands.Add("New") : s1.tabIcons.Add("SampleIcons\truss.ico")
            s1.tabCommands.Add("Open") : s1.tabIcons.Add("SampleIcons\truss.ico")
            out.Add(s1)
            ' second section
            Dim s2 As New tabSection With {.tabName = "Output data"}
            s2.tabCommands.Add("Run") : s2.tabIcons.Add("SampleIcons\truss.ico")
            s2.tabCommands.Add("Info") : s2.tabIcons.Add("SampleIcons\truss.ico")
            out.Add(s2)

            Return out
        End Get
    End Property
End Class
```

2.2 Support for plugin ribbon bar

With *IPluginWithCommands2* interface, a single plugin can host an entire new ribbon tab to represent plugin commands. Buttons and sections are defined in the `Public ReadOnly Property tabSections`.

The previous code generates the ribbon tab depicted below.



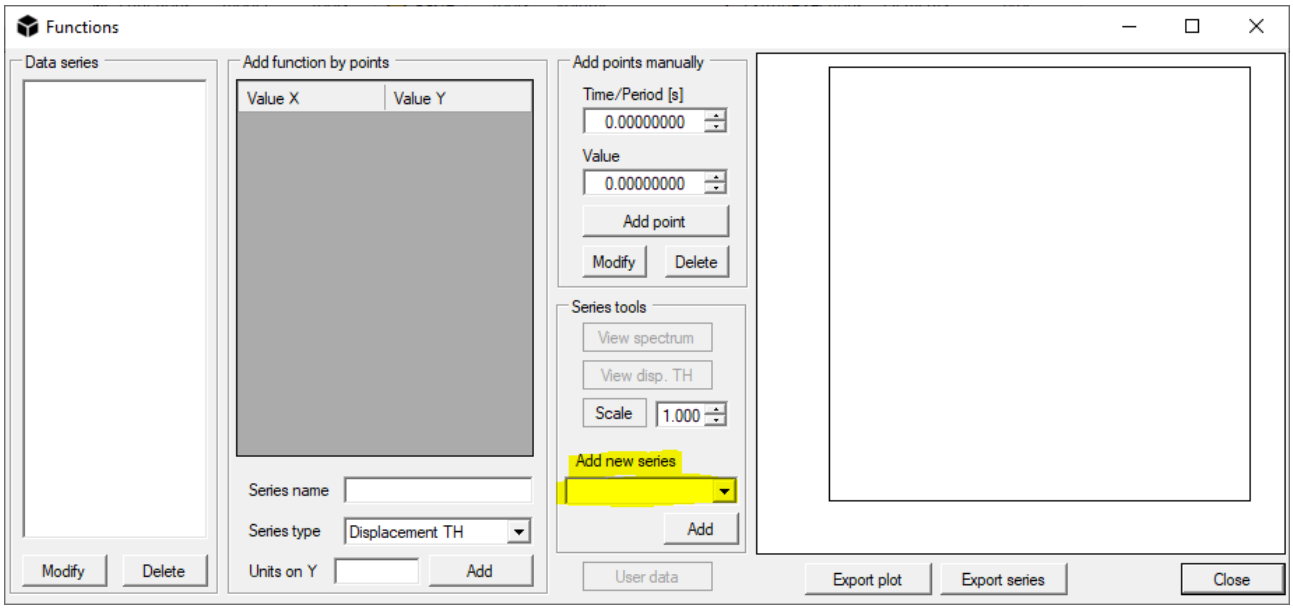
Please note that both lists `tabCommands` and `tabIcons` should have the same number of items. The latter should contains the part of the path after *installationFolder**plugins*\ directory.

It is important to notice that, since you don't have a single form like in the *Iplugin* interface, the old (inherited) `LaunchMe` procedure has been used to initialize the plugin.

2.3 Adding functions to Functions window

Functions window is the main tool to add and manage functions (accelerograms, response spectra, lateral force patterns, etc.) in NextFEM Designer.

The advanced plugin interface *IPluginWithCommands2* is able to add functions handlers for your custom function code in the *Add new series* dropdown menu.



You can add a new function by calling, during plugin initialization, this code:

```
' add function to Functions window
RaiseEvent commandOps(Me, Commands.AddFunctionHandler, "MyFunc")
```

In this way, an item called "MyFunc" will be added to the dropdown menu.

In order to return values for the function, you must use `Public Function returnData`, which returns an array of Objects that in this case is of size 4. In order, you should provide:

- list of *Drawing.PointF* for function series
- an integer representing the series type (see enumerator in code comments)
- the units of measure of values in Y axis
- the suggested name for the function.

2.4 Program commands

You can open window and tools from inside NextFEM Designer by calling them from the plugin. You should simply raise the event `RaiseEvent commandOps(sender, ops, data)`.

By now, the *ops* and *data* parameters can be one of the `Commands` enumerator and object of the listed type, respectively.

Commands enumerator	**Action and remarks**	**Passed object data type**
KeyForwarding	Forward the pressed keys to Designer	KeyEventArgs
RunModel	Run the model within Designer	-
RunSelectedLCs	Run only selected loadcases	Array of string (loadcases)
ExportDXFsection	Export DXF of selected sections	Array of string (section IDs)
ExportDXFelement	Export DXF of selected elements	Array of string (element IDs)
ShowReportMask	Show Report mask	-
ShowSectionAnalyzer	Show Section Analyzer window	Integer (section ID)
GetViewportImage	Get a screenshot of viewport	Bitmap (return value)
StartOtherPlugin	Start the specified plugin	String (plugin name)
ShowTabularInputMask	Show an input mask with table	Dictionary(Of String, String)
OpenFunctionsWindow	Open the Functions window	String (optional, selected in new series menu)
AddFunctionHandler	Add a function handler to Functions mask	String (function name)
AddDrawingGrid	Add a grid in viewport	Array of string (oX,oY,oZ,B,W,d1,d2,plan)
ClearDrawingGrid	Clear all grids in viewport	-
CreateDocXreport	Create a DocX file with selected text	Array of string (DocXpath, [template path or “”], line1, line2, ...)
ProcessDocXkeys	Process a DocX file replacing keywords	String with full path of the file (DocXpath)

2.5 Compatibility

In the case you want to use the program commands without ribbon tab support, you should implement the *LaunchMe* function as in the *Iplugin* sample, and return a null (or Nothing) value from *tabSections* property.

2.6 Considerations on referenced DLLs and redistribution

A plugin must have a unique name. Before to redistribute you plugin please contact support@nextfem.it to communicate plugin name.

You can reference all open source packages used in NextFEM Designer, by simply using the DLLs in the *plugins* upper directory (the installation root folder). For instance, you can use *ZedGraph* in your plugin without adding your own copy. Please remember to include attributions and/or licenses.
Since the plugin templates have an *app.config* file that specifies that the search folder for referenced

libraries is the upper one, you can always use other locations by using the following code (which is necessary, for instance, when you're using language-specific resources).

In the code excerpt below, "it" and "es" resources are supposed to be used.

```
    ' for loading dlls in the same folder
    Friend currentDomain As AppDomain = AppDomain.CurrentDomain
    Friend Function MyResolveEventHandler(ByVal sender As Object, ByVal args As ResolveEventArgs)
As System.Reflection.Assembly
        Dim MyAssembly, objExecutingAssembly As System.Reflection.Assembly
        MyAssembly = Nothing
        objExecutingAssembly = System.Reflection.Assembly.GetExecutingAssembly()
        ' seek in the same folder
        'Dim name As String = args.Name.Substring(0, args.Name.IndexOf(","))
        Dim name As String = args.Name
        Dim cInd As Integer = args.Name.IndexOf(",")
        If cInd > -1 Then name = args.Name.Substring(0, cInd)
        Dim pathL As New List(Of String)
        pathL.Add(My.Application.Info.DirectoryPath & "\plugins\" & name & ".dll")
        If args.Name.Contains("=it") Then
            pathL.Add(My.Application.Info.DirectoryPath & "\plugins\it\" & name & ".dll")
        ElseIf args.Name.Contains("=es") Then
            pathL.Add(My.Application.Info.DirectoryPath & "\plugins\es\" & name & ".dll")
        End If
        For Each path In pathL
            If IO.File.Exists(path) Then MyAssembly = System.Reflection.Assembly.LoadFrom(path)
        Next
        'Return the loaded assembly
        Return MyAssembly
    End Function
```

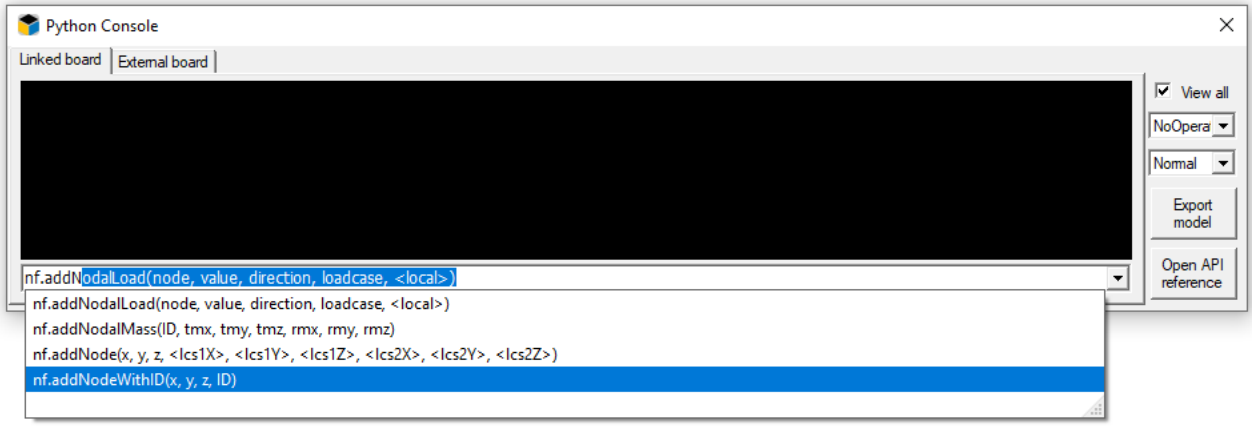
In plugin initialization:

```
' use DLL loading for load assemblied from the same plugin folder (or set subfolder in
MyResolveEventHandler)
        AddHandler currentDomain.AssemblyResolve, AddressOf MyResolveEventHandler
```

To the same email address, you can request installer source code template for your plugin, which guarantees that NextFEM Designer is installed before to install the plugin.

3 NextFEM Python Console

NextFEM APIs can be used with Python as well. This plugin helps developers in finding and testing every API instruction, directly in Designer and with immediate effect in the model.



This plugin works only with 64bit version of NextFEM Designer and needs Python 3.8 64bit installed in system. Users must have Python 3.8 available in the PATH environment variable.

Direct link for downloading Python 3.8 64bit

This plugin is formed by two boards:

1. a **Linked board** in which every API method affects the model and the viewport. This has Python syntax and some basic functions, but it's not a complete Python console. In the lower textbox, start typing "***nf.***" and the available commands will be suggested, as well as their arguments. Arguments between "<" and ">" are optional. API reference can be recalled by pressing *Open API reference*.
2. An **External board**, which is a complete Python terminal. The instance of API (always called **nf**) is not linked to the model here. You can export the model from linked to external board by the button *Export model*.

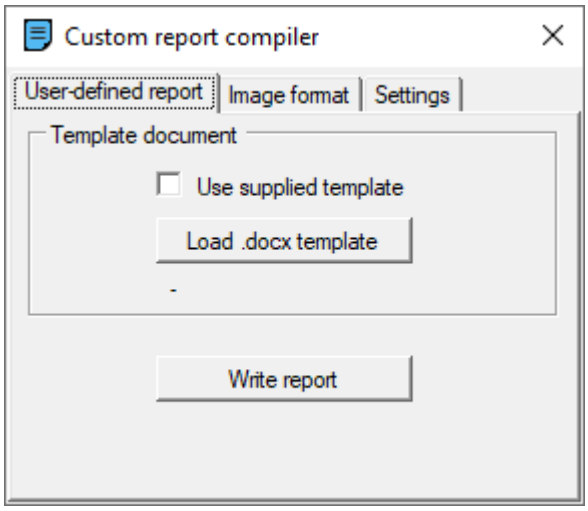
The other controls in this form are associated to the *ViewState* and *undoOps* enumerator and with the *resize* boolean variable as explained above.

4 NextFEM Custom Report

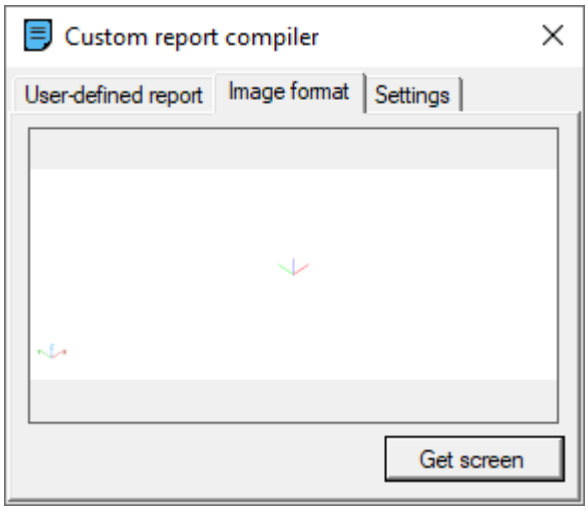
Custom Report is a plugin for NextFEM Designer able to insert in a DocX document file (typically made in *Microsoft Word(R)*) images, text and tables about model and results.
By now, General Design module license is required to use this plugin.

Custom Report automatically seeks in the folder where model is saved if a DocX template is available. File must be called "`reportTemplate.docx`". If there's such file, it will be loaded automatically.

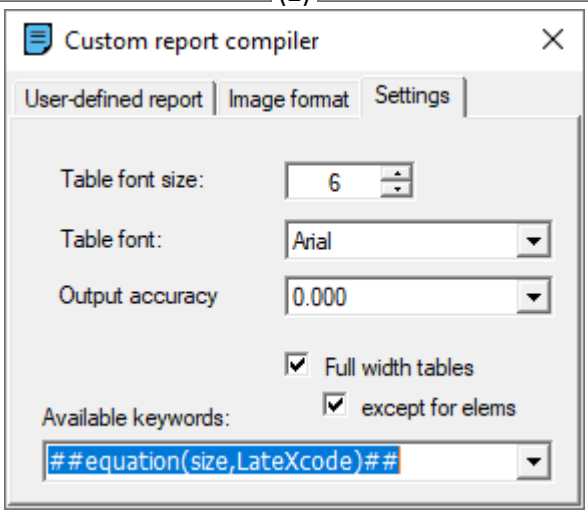
(1)



(2)



(3)



Let's see which commands are available inside this plugin:

1. **tab 1** contains commands to load your .docx template and for writing report;
2. **tab 2** allows to see graphically how viewport is screen-shot inside the report. Please position your model in the best way possible. To insert a screenshot of the model as it, use keyword #image.asis# in the Word document;
3. **tab 3** contains options: you can specify the font for tables and its size. Keywords to be included in the docx template are listed. Read the following for their explanation.

The docx template is a simple document containing **keywords**, that will be replaced with the requested contents. The available keywords are:

Comments can be inserted by embracing them between ###.

Keyword	Description
##text.name##	Name of the current model file
##text.nodesnumber##	Number of nodes
##text.elementsnumber##	Number of elements
##text.length##	Length unit
##text.force##	Force unit
##text.beamdiagram(ID,loadcase,time,type,station)##	Beam diagram value for current element ID, loadcase, time, type is 1=N, 2=Vy, 3=Vz, 4=Mt, 5=My, 6=Mz; station can be 1=I, 2, 3, 4 or 5=J
##table.nodes##	Node tables with coordinates
##table.elements##	Element tables with connectivity
##table.checking(loadcase,time)##	Checking results table for current loadcase and time
##table.materials##	Table of materials with main properties
##table.sections##	Table of sections with main properties
##table.combinations##	Table of combinations set in the model
##image.asis##	Image as displayed in viewport
##image.extruded##	Image of extruded model
##image.nodesnumber##	Image of the model with nodes numbers
##image.elementsnumber##	Image of the model with elements numbers
##image.showloads(loadcase)##	Image of applied loads for selected loadcase
##image.beamdiagram(loadcase,time,type)##	Image of beam diagrams for the current loadcase, time and type (1=N, 2=Vy, 3=Vz, 4=Mt, 5=My, 6=Mz)
##image.areaforce(loadcase,time,type)##	Image of area forces for the current loadcase, time and type (1=fxx, 2=fyy, 3=fxy, 4=mxx, 5=myy, 6=mxy, 7=qxz, 8=qyz, 9=mxWAbot, 10=myWAbot, 11=mxWAtop, 12=myWAtop, 13=mxWA, 14=myWA)
##image.soilpressure(loadcase,time)##	Image of soil pressure for selected loadcase and time
##image.path(imagePath,sizeRatio)##	Image from external path. SizeRatio is the percentage of the page width
##image.elementsection(elemID,station,sizeRatio)##	Insert image of the element section. SizeRatio is the percentage of the page width
##equation(size,LateXcode)##	Insert equation in LateX format. First parameter is the font size.

5 Material library customization

The material library can be expanded by writing a CSV file (i.e. a text file using the semicolon separator), as in the following examples.

- *Steel*

```
Name;Code;E [MPa];G [MPa];n_;fyk [MPa];ftk [MPa];Wd [kN/m^3];Md [ton/m^3];CheckType
S235;UNI EN 10025-2;210000;80769;0.3;235;360;78.6;0.801;1
S275;UNI EN 10025-2;210000;80769;0.3;275;430;78.6;0.801;1
```

- *Concrete*

```
Name;Code;E [MPa];G [MPa];n_;fck [MPa];Wd [kN/m^3];Md [ton/m^3];CheckType
C8/10;NTC2008;25331;10555;0.2;8;25;254.841998
C12/15;NTC2008;27085;11285;0.2;12;25;254.841998
```

The column *CheckType* specifies the verifications associated with the material. Insert:

- 1 for *Steel*
- 2 for *Aluminium/Alloy*
- 3 for *Concrete*
- 4 for *Timber*
- 0 for other materials.

For other types of materials, all the columns are mandatory, except for “Md” – mass density, which will be calculated by the program in consistent units from Wd – weight density.

Natively supported fields are:

- E: Young modulus
- n_: Poisson ratio
- G: shear modulus
- fk: characteristic strength
- Wd: weight density
- a_T: thermal coefficient for linear deformation
- K: heat conductivity
- Cp: specific heat capacity

The property name can be followed by square brackets containing the unit of measure (eg. “E [MPa]”).

The file must be placed in the “data” folder and its extension must be **.nfm*.

6 Section library customization

The section library can be expanded by writing a CSV file (i.e. a text file using the semicolon separator) as follows:

```
Name;Code;h [mm];b [mm];tw [mm];tf [mm];r [mm];A [cm^2];Jy [cm4];Wey [cm^3];Wpy [cm^3];iy [cm];Avz [cm^2];Jz [cm^4];Wez [cm^3];Wpz [cm^3];iz [cm]
IPE A 80;UNI;78;46;3.3;4.2;5;6.375401837;64.37774091;16.50711305;18.97743881;3.177708713;3.070001837;6.852668856;2.979421242;4.692462888;1.036754887
…
```

The first four columns are mandatory.
Natively supported fields are:

- h: height
- b: base
- d: diameter
- t: thickness (for pipe sections)
- tw: web thickness
- tf: flange thickness
- A: area of the section
- Jz: moment of inertia with respect to horizontal axis
- Jy: moment of inertia with respect to vertical axis
- Wez: elastic strength modulus with respect to horizontal axis
- Wey: elastic strength modulus with respect to vertical axis
- Wpz: plastic strength modulus with respect to horizontal axis
- Wpy: plastic strength modulus with respect to vertical axis
- Avz: shear area with respect to horizontal axis (local z)
- Avy: shear area with respect to horizontal axis (local y)
- gap: horizontal distance for double sections (eg. double L or double C).

The property name can be followed by square brackets containing the unit of measure (eg. “h [cm]”).

The *Type* columns specifies the shape of the section. Insert:

- 0 for unknown or generic
- 1 for rectangular
- 2 for circular
- 3 for C shape
- 4 for T shape
- 5 for double-T o I
- 6 for L shape
- 7 for box
- 8 for pipe
- 9 for double L
- 10 for double C.

The file must be placed in the “data” folder and its extension must be **.nfs*.

6.1 Library of sections defined by points

Sections defined by points can be inserted in a **.nfs* file like in the previous case, but with the following fields:

```
Name;Code;PointsXf1 [mm];PointsYf1 [mm];PointsXe2 [mm];PointsYe2 [mm];PointsXf3 [mm];PointsYf3 [mm];Class;OffsetX [mm];OffsetY
[mm];alphaLT;alphay;alphaz;Jw [mm^4]
TestSect;custom;0,20,20,0;0,0,20,20;4,8,8,4;4,4,8,8;0;0;3;0;0;0.76;0.76;0.76;0
…
```

First 2 columns are mandatory, and value of Code column must be set to “custom”. Series of points can be whichever needed, by following this codification for columns names:

PointsX*pN* [*units*];PointsY*pN* [*units*];

in which *p* is equal to “f” for filled figures and “e” for holes; *N* is the progressive number of series, the same for filled and empty shapes. All the columns after section offsets are inserted in section custom values (e.g. *alphaLT;alphay;alphaz;Jw*).

7 Custom verifications

It is possible to customize the list of verifications with the notation described in the following. The text file containing checking can be placed in the "*verification*" folder in the program installation directory and must have the "*.nvv*" extension (see for example *trusses.nvv*).

Such file can be edited with:

- Notepad++, with the setting for the syntax highlight that can be found in the "*verification*" folder (file *NextFEMVerifications.xml*);

- Visual Studio Code with Custom Coloring plugin for syntax highlight. Setting for this plugin are in the file *NextFEMverification_vsCode.json* contained in the folder "*verification*" as well.

The checking engine supports the code execution in blocks. Each block is delimited by the identifiers as in the following example and must the named with a floating point number (0.6).

$$0.6

 # this is a comment

execif(SecType==1,1.0)

$!

To call each block, the following keywords are available:

- *exec(0.6)* : executes the code block named 0.6
- *execif(condition, 0.6)*: executes the code block named 0.6 is *condition* is true;
- *execwhile(condition, 0*.6): executes the code block named 0.6 while condition is true. The variable *exitdo* equal to 1 within the code block forces the exit. Limited to 10000 cycles.

Formulas can be written using the following operators/functions:

Addition: **+**
Subtraction: **-**
Multiplication: *****
Division: **/**
Modulo: **%**
Exponentiation: **^**
Less than: **<**
Less than or equal: **<=** or **≤**
More than: **>**
More than or equal: **>=** or **≥**
Equal: **==**
Not Equal: **!=** or **≠**
Sine: **sin**
Cosine: **cos**
Arcsine: **asin**
Arccosine: **acos**
Tangent: **tan**
Cotangent: **cot**
Arctangent: **atan**
Arc cotangent: **acot**
Natural logarithm: **loge**

Common logarithm: **log10**
Logarithm: **logn**
Square root: **sqrt**
Conditional key: **if(var<var2,1,0)**
Boolean operator: **and(cond1,cond2)**
Boolean operator: **or(cond1,cond2)**
Boolean operator: **not(cond)**

The hardcoded variables are:

- **Model units handling**
  - **unitconv**: converts between units. Usage: *unitconv(oldUnits,newUnits,Value)*.
    Example: Eps=sqrt(235/*unitconv(model_S,MPa,fk)*)
    converts fk from *stress units in the model* to *MPa*
  - **rcsect**: calculates resisting moments of a RC section, storing them in Mry and Mrz.
    Usage: *rcsect(N,Myy,Mzz)*
  - **skipItem**: if =1, skips the subsequent checking. To be used only in the time-dependent load cases (for example, linear dynamic analysis)
  - **model_L**: placeholder for the length unit in the model
  - **model_F**: placeholder for the force unit in the model
  - **model_FL**: placeholder for the force per length unit in the model
  - **model_T**: placeholder for the temperature unit in the model
  - **model_M**: placeholder for the mass unit in the model
  - **model_S**: placeholder for the stress unit in the model
  - **isVarDefined(*var*)**: checks if a variable is defined (1) or not (0)
  - **Halt**: stops the execution;
  - **addUtable:** adds an user table at runtime, and returns the ID of the table. Usage: addUtable(numColumns, columnsHeaders, data by row including row header).
    Example: tid=addUtable(3,1,2,3,100,4,5,6,101,4,5,6,102,4,5,6,103,4,5,6)
    adds the following table:

	1	**2**	**3**
100	4	5	6
101	4	5	6
102	4	5	6
103	4	5	6

  - **UtableAt**: returns the value at i, j of a table. Usage: UtableAt(table ID,i,j)
  - **UtableInterpValues:** gives the bilinear interpolation for the table. Usage: results=UtableInterpValues(table ID, value on row headers, value on column headers).
    Example: res1= UtableInterpValues(tid,101.3,2.5) gives 5.5 as a result.
  - **addRebar**: adds longitudinal rebar to the current element.
    The function returns 0 if it fails or design material cannot be found.
    Usage: addRebar(zCoord, yCoord, rebarArea, design material ID, startAt [0,1), endAt (0,1]).
    Example: addRebarL(40,40,201.0,2,0,1)
    adds a rebar of area 201.0 at position 40, 40 from the bottom left corner of the section, for the whole element.

- **addStirrups**: adds stirrups to the current element.
  The function returns 0 if it fails or design material cannot be found.
  Usage: addStirrups(legs in Y, legs in Z, single bar area, spacing, design material ID, startAt [0,1), endAt (0,1]).
  Example: addStirrups(2, 2, 50.0, 200, 2, 0,1)
  adds 2-by-2 legs stirrups, each one of area 50.0, at 200 of spacing, for the whole element.
- **clearRebar:** clear all rebar, including stirrups, in the current element.
- **getBarDiam**: gets the minimum diameter of a longitudinal bar required to satisfy the given area. The output is in mm. Example: getBarDiam(2.01) # for a model in cm, returns 16.
- **getStirrupDiam**: gets the minimum diameter of a stirrup bar required to satisfy the given area. The output is in mm. Example: getStirrupDiam(0.5) # for a model in cm, returns 8.
- **TranslateMomentZZ**: gives maximum and minimum moments around z local axis for the given position, translated by a quantity *delta*. Syntax: TranslateMomentZZ(position, delta). The argument *position* is contained in the built-in dataset for elements in the variable *pos*. This function returns or overwrite the variables *Mzzmax* and *Mzzmin*.
- **TranslateMomentYY**: gives maximum and minimum moments around y local axis for the given position, translated by a quantity *delta*. Syntax: TranslateMomentYY(position, delta). The argument *position* is contained in the built-in dataset for elements in the variable *pos*. This function returns or overwrite the variables *Myymax* and *Myymin.*
- **FireSectionStrength**: loads a thermal map of a section and use it to estimate element strength. It is available in the FireSafe module. The filename containing thermal results for the station must be saved inside the "fireSect" key in custom properties of the element. Syntax: FireSectionStrength(station,N,Myy,Mzz). The argument *station* is the checking station, typically from 1 to 5.
- **round**: round a floating point number to the nearest integer. Ex. round(12.6) returns 13.0.
- **ceil**: round up a floating point number. Ex. ceil(12.3) returns 13.0.
- **floor**: round down a floating point number. Ex. floor(12.6) returns 12.0.

- **Built-in general dataset:**
  - **ServType:** indicates if the current serviceability combination is characteristic (rare) (1), frequent (2) or quasi-permanent (2). The value is 0 if the combination is not of serviceability type.
  - **Seismic**: indicates if the current combination is seismic (1) or not (0). The value 1 is always associated to ServType=0.
  - **time:** current time.

- **Built-in dataset for elements:**
  - **A**: Area
  - **Jz**: Moment of inertia around z-axis
  - **Jy**: Moment of inertia around y-axis
  - **Jmin**: Minimum moment of inertia
  - **Jt**: Torsional Inertia

- **D**: Diameter of circular cross sections
- **Di**: Inner diameter of pipe cross sections
- **te**: Thickness of pipe cross sections
- **b**: Base for any other cross sections
- **h**: Height for any other cross sections
- **tw**: web thickness
- **tf1**: thickness of bottom flange
- **tf2**: thickness of upper flange
- **t**: thickness for planar sections
- **N**: Axial force of the current section of a beam
- **Vy**: Shear force along y direction of the current section of a beam
- **Vz**: Shear force along z direction of the current section of a beam
- **Mt**: Twisting moment of the current section of a beam
- **Myy**: Moment around y local axis of the current section of a beam
- **Mzz**: Moment around z local axis of the current section of a beam
- **rMyIJ**: ratio between Myy at end I and at end J of the beam
- **rMzIJ**: ratio between Mzz at end I and at end J of the beam
- **MmaxY**: maximum moment around y axis for the whole element or member
- **MmaxZ**: maximum moment around z axis for the whole element or member
- **MminY**: minimum moment around y axis for the whole element or member
- **MminZ**: minimum moment around z axis for the whole element or member
- **Em**: material Young modulus
- **Gm**: material shear modulus
- **NIm**: material Poisson's ratio
- **fk**: material characteristic strength
- **WelZ**: section modulus for z axis
- **WelY**: section modulus for y axis
- **WplZ**: plastic section modulus for z axis
- **WplY**: plastic section modulus for y axis
- **iz**: radius of inertia for z axis
- **iy**: radius of inertia for y axis
- **imin**: minimum radius of inertia
- **SecType**: 1=beam, 2=planar, 0=unknown
- **SecBeamType**: 0=unknown, 1=rectangular, 2=circular, 3=Cshape, 4=Tshape, 5=DoubleTshape, 6=Lspahe, 7=box, 8=pipe
- **MatType**: type of material. Steel 1, Aluminium 2, Concrete 3, Timber 4, Masonry 5, Fragile in tension 6, unknown 0.
- **dx**: axial relative displacement along beam axis
- **dy**: transversal deflection in local direction y
- **dz**: transversal deflection in local direction z
- **L0yy**: buckling length for bending around y local axis
- **L0zz**: buckling length for bending around z local axis
- **L0**: maximum buckling length between L0yy and L0zz
- **pos**: of the current section of a beam, in model units
- **LvyI**: contraflexure length from end I, ratio Myy/Vz in station 1
- **LvyJ**: contraflexure length from end J, ratio Myy/Vz in station 5

- **LvzI**: contraflexure length from end I, ratio Mzz/Vy in station 1
- **LvzJ**: contraflexure length from end J, ratio Mzz/Vy in station 5
- **fxx**: force in x local direction (element centre)
- **fyy**: force in y local direction (element centre)
- **fxy**: shear force in xy direction (element centre)
- **fzz**: force in z local direction (element centre)
- **fxz**: shear force in xz direction (element centre)
- **fyz**: shear force in yz direction (element centre)
- **mmxx**: moment around x local direction (element centre)
- **mmyy**: moment around y local direction (element centre)
- **mmxy**: moment around xy direction (element centre)
- **mmzz**: moment around z local direction (element centre)
- **mmxz**: moment around xz direction (element centre)
- **mmyz**: moment around yz direction (element centre)
- **time**: current time step
- **temp**: average element temperature in the current step
- **isWall**: if wall groups is defined for the current planar element returns 1, 0 otherwise.
- **Column**: if the current beam element is vertical returns 1, 0 otherwise. It is not defined for other types of elements.

- **Built-in dataset for nodal results:**
  - **dx**: nodal displacement in X direction
  - **dy**: nodal displacement in Y direction
  - **dz**: nodal displacement in Z direction
  - **rx**: nodal rotation around X axis
  - **ry**: nodal rotation around Y axis
  - **rz**: nodal rotation around Z axis
  - **vx**: nodal velocity in X direction
  - **vy**: nodal velocity in Y direction
  - **vz**: nodal velocity in Z direction
  - **vrx**: nodal velocity around X axis
  - **vry**: nodal velocity around Y axis
  - **vrz**: nodal velocity around Z axis
  - **ax**: nodal acceleration in X direction
  - **ay**: nodal acceleration in Y direction
  - **az**: nodal acceleration in Z direction
  - **arx**: nodal acceleration around X axis
  - **ary**: nodal acceleration around Y axis
  - **arz**: nodal acceleration around Z axis
  - **Rex**: nodal reaction in X direction
  - **Rey**: nodal reaction in Y direction
  - **Rez**: nodal reaction in Z direction
  - **Rerx**: nodal reaction around X axis
  - **Rery**: nodal reaction around Y axis
  - **Rerz**: nodal reaction around Z axis
  - **sxx**: nodal stress in X direction

- **syy**: nodal stress in Y direction
- **sxy**: nodal shear stress in XY direction
- **szz**: nodal stress in Z direction
- **sxz**: nodal shear stress in XZ direction
- **syz**: nodal shear stress in YZ direction
- **fxx**: nodal force in X direction
- **fyy**: nodal force in Y direction
- **fxy**: nodal shear force in XY direction
- **fzz**: nodal force in Z direction
- **fxz**: nodal shear force in XZ direction
- **fyz**: nodal shear force in YZ direction
- **mxx**: nodal moment around X direction
- **myy**: nodal moment around Y direction
- **mxy**: nodal moment around XY direction
- **mzz**: nodal moment around Z direction
- **mxz**: nodal moment around XZ direction
- **myz**: nodal moment around YZ direction
- **time**: current time step
- **temp**: nodal temperature in the current step.

7.1 Documenting checks

Since v2.0.3.2, comments, checking formulae and images can be added to a verification report. This is the suggested workflow:

1. Run checking as usual with a properly commented script;
2. Obtain the verification log (available as plain text for a single station);
3. Create a .docx document and paste the content of the verification log;
4. Render the .docx document in Custom Report plugin or via API.

A properly commented script can be written as follows.

```
# this is a comment - Verification file for truss elements
# lines can be separated by |. Literals are wrapped by ~. Variables can be used between »
§A§ ###| Section properties |~ ##image.elementsection(»ElNum»,»Station»,0.35)##~
§s_str§ ###| Axial check |~ ##equation(14,{\sigma _{str}} = \frac{{\left| N \right|}}{A})##~
s_str=abs(N)/A
# convert 220MPa in model stress units
s_adm=unitconv(MPa,model_S,220)
# Check the formulas below
@Axial
rN=s_str/s_adm
```

Let's see which are the effects of the line:

```
§s_str§ ###| Axial check |~ ##equation(14,{\sigma _{str}} = \frac{{\left| N \right|}}{A})##~
```

The first part (§s_str§) indicates to which variable refer the comment. This means that the comment will be placed before the variable named *s_str* introduced in the code. Newlines are represented by the character "|". Hence, the previous line will render as:

```
###
Axial check
##equation(14,{\sigma _{str}} = \frac{{\left| N \right|}}{A})##
```

The last line is compatible with Custom Report processor, and asks the compiler to replace that line with an equation image. The equation is described in LaTeX format and it will be rendered like in the following picture.

Let's have a look also to the other line:

```
§A§ ###| Section properties |~ ##image.elementsection(»ElNum»,»Station»,0.35)##~
```

This is a reference to a built-in variable (*A*, which is the area of the transversal section). In this case we ask the DocX compiler to add an image of the element's section. To do this, the keyword should report the number of the element.

In order to have the correct output, e.g. for element 120, station 1 (I-end):

```
###
Section properties
##image.elementsection(120,1,0.5)##
```

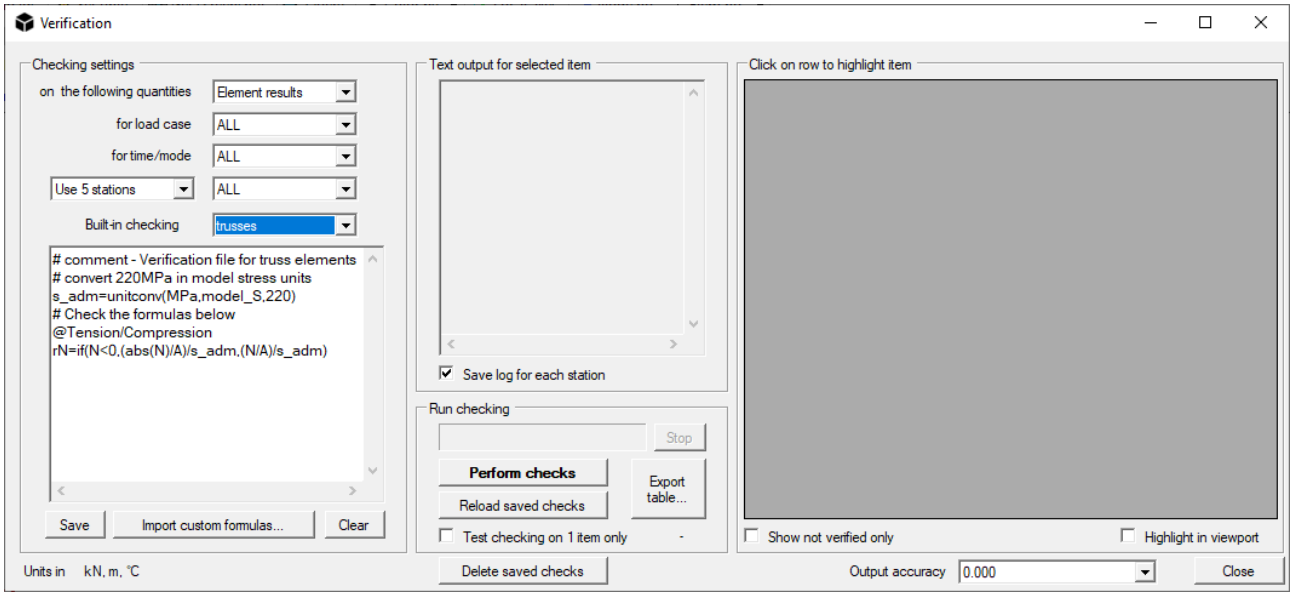
We recalled the element number by using directly one of the variable in checking (ElNum), written between the "»" character. This is a simple variable replacement; operations in variables are not permitted but can be demanded to the script itself before the comment.

7.2 Custom checking sample

The syntax is very simple and straightforward, you can even copy your formulas from a spreadsheet since the most common math functions are supported.

The script must be loaded in the Verification window (*Results/Verifications…* command); to do so, create a text file and place it into the subfolder "verification" in the program installation directory (typically it's *C:\Program Files\NextFEM\NextFEM Designer\verification*).

The file must have the extension *.nvv* to be read automatically and to appear in the drop-down list as below.

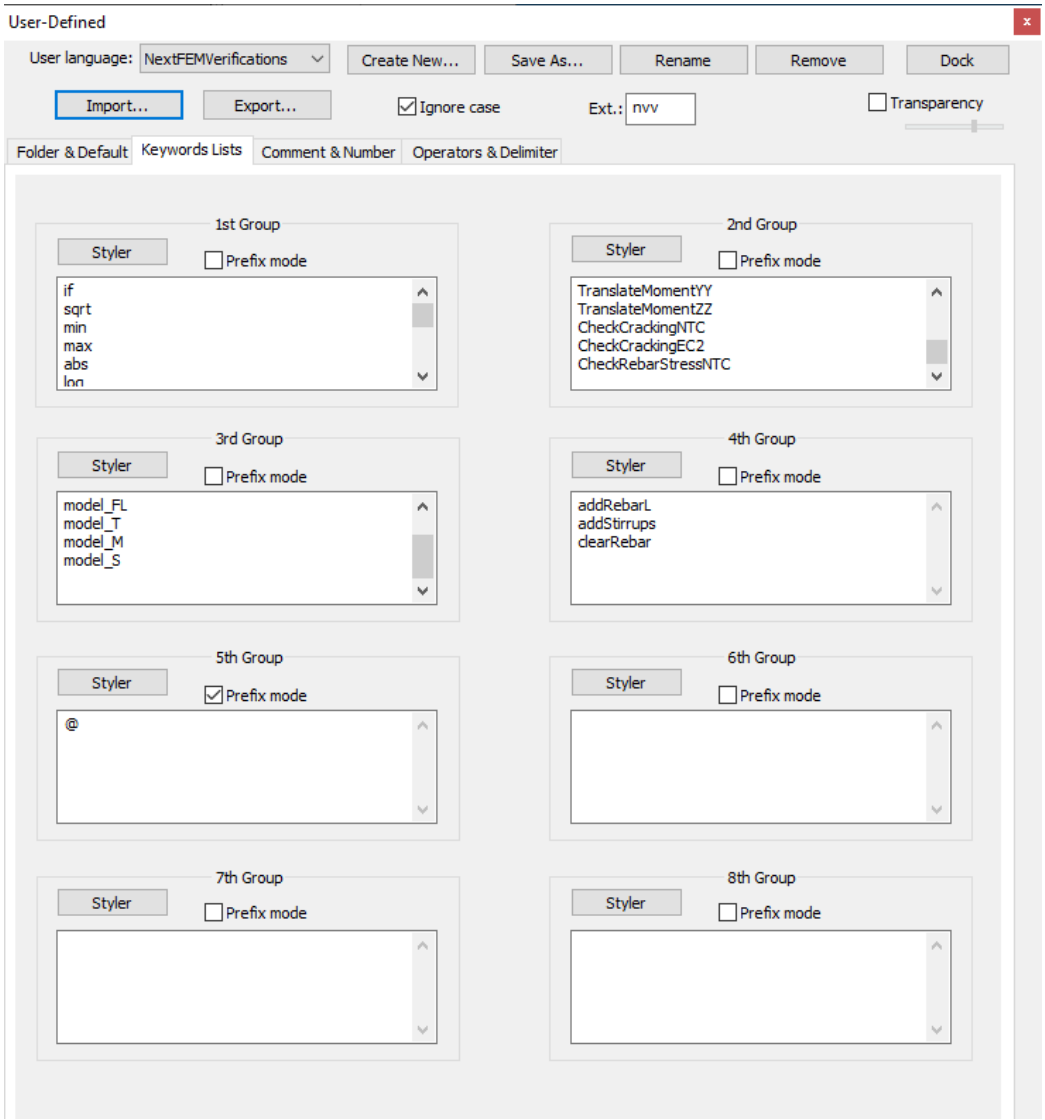


You can also edit your script in the text area of Verification window, or you can modify the file outside the program.

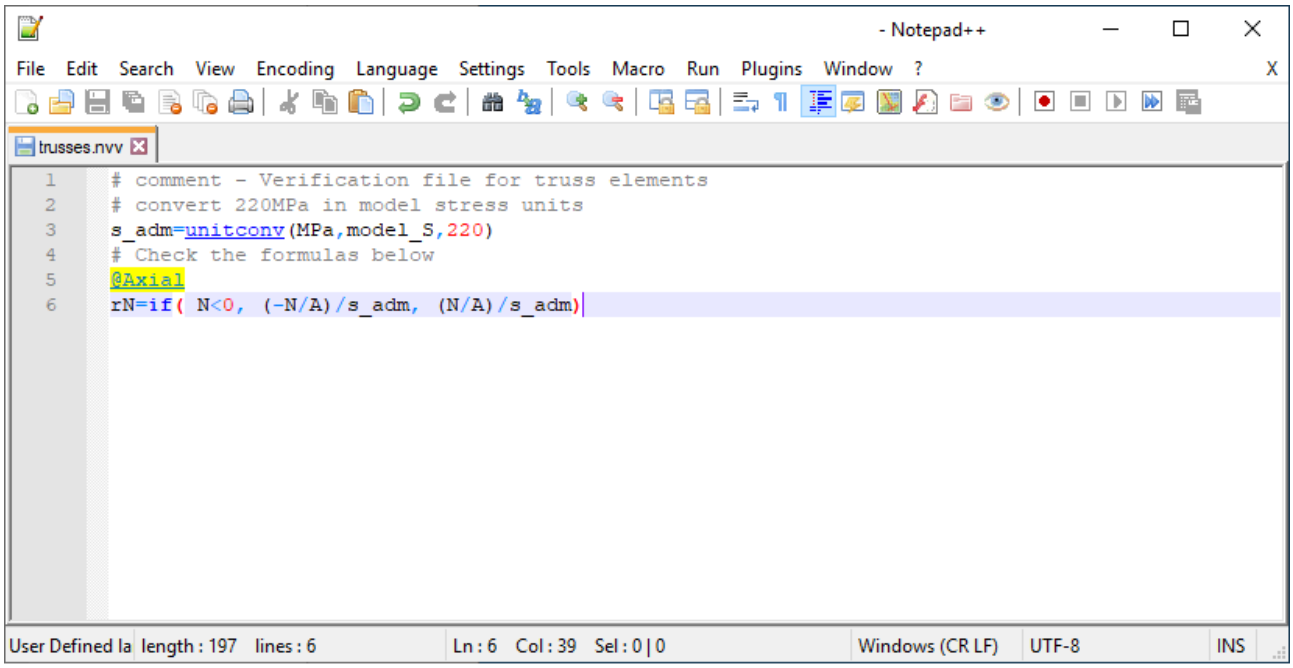
Writing checks

We recommend an editor with syntax highlight for writing your script, such as Notepad++ (https://notepad-plus-plus.org/) or Visual Studio Code (see chapter 7).

A setting for the syntax highlight for Notepad++ can be found in the "*verification*" folder (*NextFEMVerifications.xml*). From inside Notepad++, it can be loaded with the command *Language/Define your language…* and by clicking on *Import…* . Then, you'll find the entry "NextFEMVerifications" in the top dropdown menu. Also, in the tab *Keyword Lists*, you can see all the supported commands for scripting that will be highlighted during editing.



Open the sample script called "trusses.nvv" supplied with the program, in the "verification" folder.



The few lines contained are self-explanatory. Note the keyword "unitconv" to convert units ("model_S" is the current stress units in the model), the @ character to create a column as output and the ***if****(condition, true, false)* structure.

In this tutorial, we'll gradually see how to develop a more complex script that can suit your needs.

Using blocks

The checking engine supports the code execution in blocks. Each block is delimited by the identifiers as in the following example and must be named with a floating point number (for example, 0.6).

$$0.6
 # this is a comment – we are inside the block named 0.6
 # the block named 1.0 will be executed only if the variable SecType is equal to 1 (==)
execif(SecType==1,1.0)
$!

To call each block for execution, the following keywords are available:

- *exec(0.6)* : executes the code block named 0.6
- *execif(condition, 0.6)*: executes the code block named 0.6 is *condition* is true.

Customizing output

Output coming from your checking is organized in columns containing numbers or Boolean values (true or false). To declare a column, use the character @ before the column name, the write the output quantity in the following line. For example:

@MomentCheck
MEd/MRd

This will create a columns named "*MomentCheck*" and will insert the value resulting from the expression *MEd/MRd* in the row corresponding to the node or the element to be checked.

Script example

Variables don't need to be declared; simply initialize them with a value or an expression. For example:

```
# design yield strength 391.3 MPa
fyd=391.3
# 500 mm section height
h=450
# 50 mm concrete cover
c=50
# 628 mm2 lower bars - 2phi20
AsInf=400
# 308 mm2 upper bars - 2phi14
AsSup=280
# 20x10^6 Nmm = 20.0 kNm Design Moment
MEd=2.0E7
```

By default, all variables are double, but you can always use them as a lower type:

```
# variable used as a boolean: 1 is true, 0 is false
positiveMoment=if(MEd>=0,1,0)
```

Let's see in action a simple script to estimate the rebar area needed in a concrete beam. Blocks can be inserted freely at any line of the code.

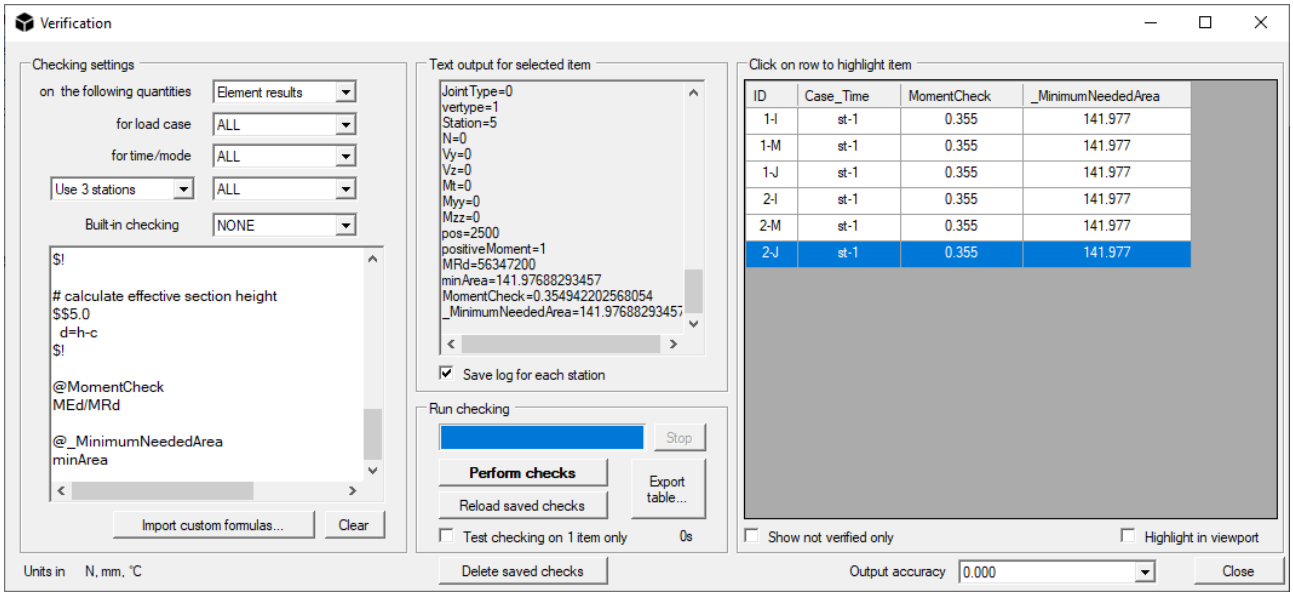
```
# design yield strength 391.3 MPa
fyd=391.3
# 500 mm section height
h=450
# 50 mm concrete cover
c=50
# 628 mm2 lower bars - 2phi20
AsInf=400
# 308 mm2 upper bars - 2phi14
AsSup=280
# 20x10^6 Nmm = 20.0 kNm Design Moment
MEd=2.0E7

# variable used as a boolean: 1 is true, 0 is false
positiveMoment=if(MEd>=0,1,0)

# call the function 5.0
exec(5.0)

# conditional function call: positiveMoment is true so 2.0 will be called
execif(not(positiveMoment),1.0)
execif(positiveMoment,2.0)

# Negative Resistant Moment estimate
$$1.0
```

```
  MRd=-0.9*d*fyd*AsSup
  minArea=MEd/(-0.9*d*fyd)
$!

# Positive Resistant Moment estimate
$$2.0
  MRd=0.9*d*fyd*AsInf
  minArea=MEd/(0.9*d*fyd)
$!

# calculate effective section height
$$5.0
  d=h-c
$!

@MomentCheck
MEd/MRd
@_MinimumNeededArea
minArea
```

To start, you must have results in our model, otherwise the checking will not be executed.

First of all, copy the sample script in the text box. Then choose the set of data you need in the first dropdown menu (*Element results*), and the number of stations (*Use 3 stations*). To see all the values for the elements given by default, please check the scripting reference in Chapter 4 of the users' manual.

Finally press *Perform check*, the table on the left will be populated with 3 rows for each element.



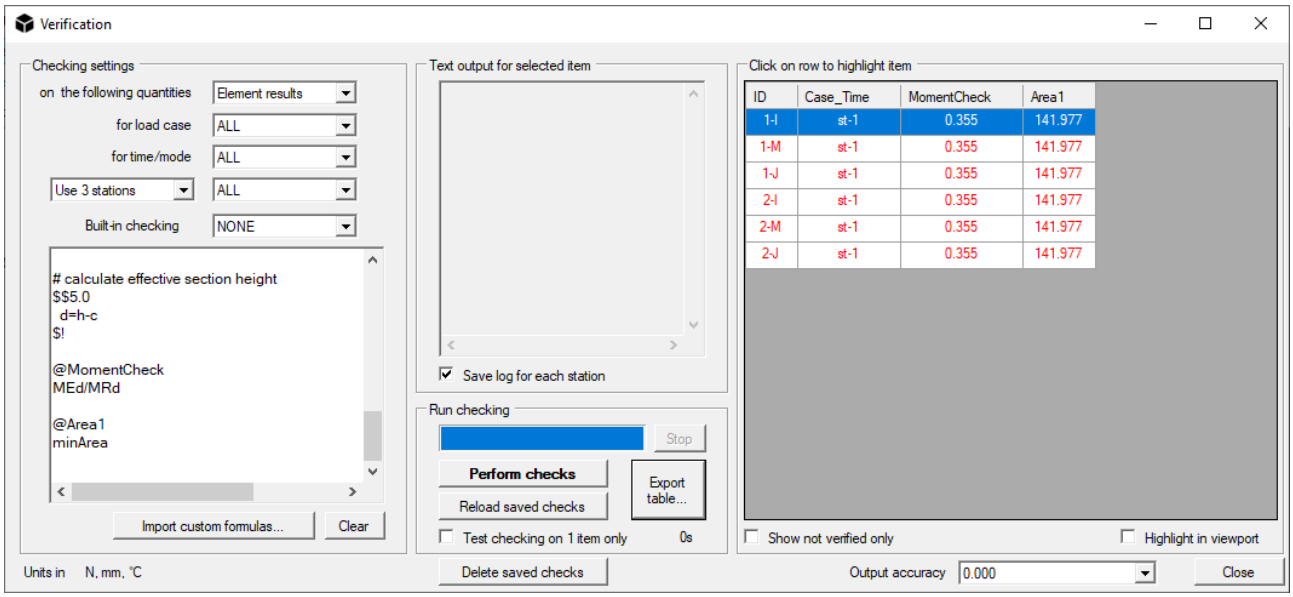
Columns highlight

If a column contains a values that is greater than 1, the entire row will be automatically highlighted in red.

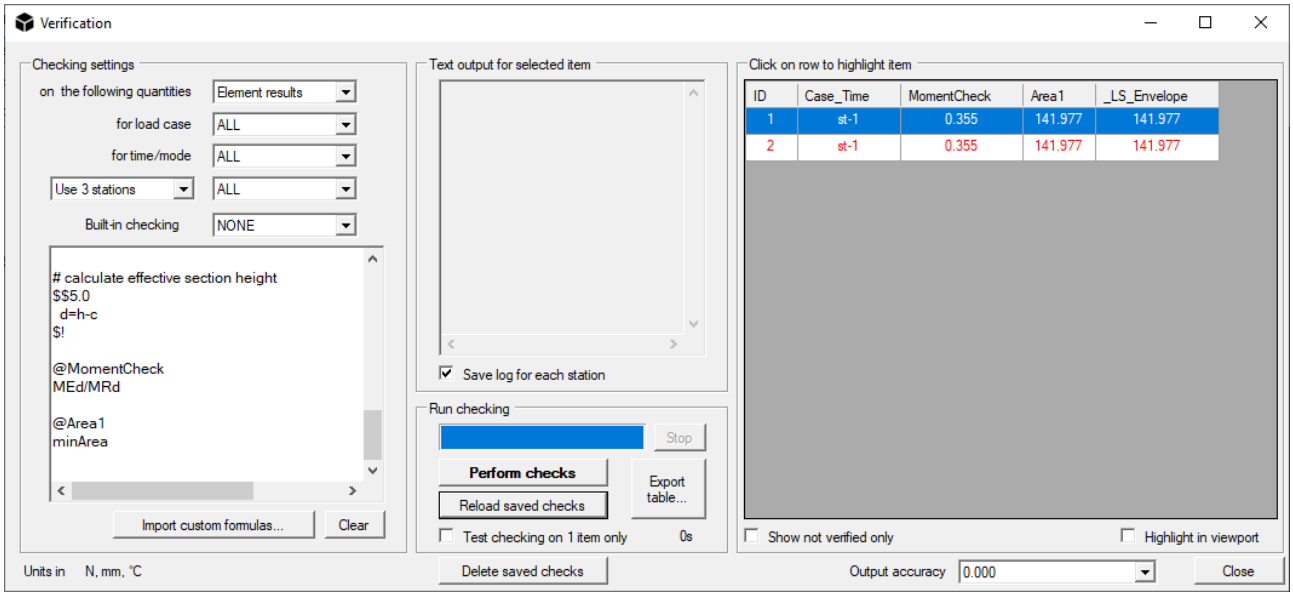
Try to append the following code to the previous script:

```
@Area1
```

minArea

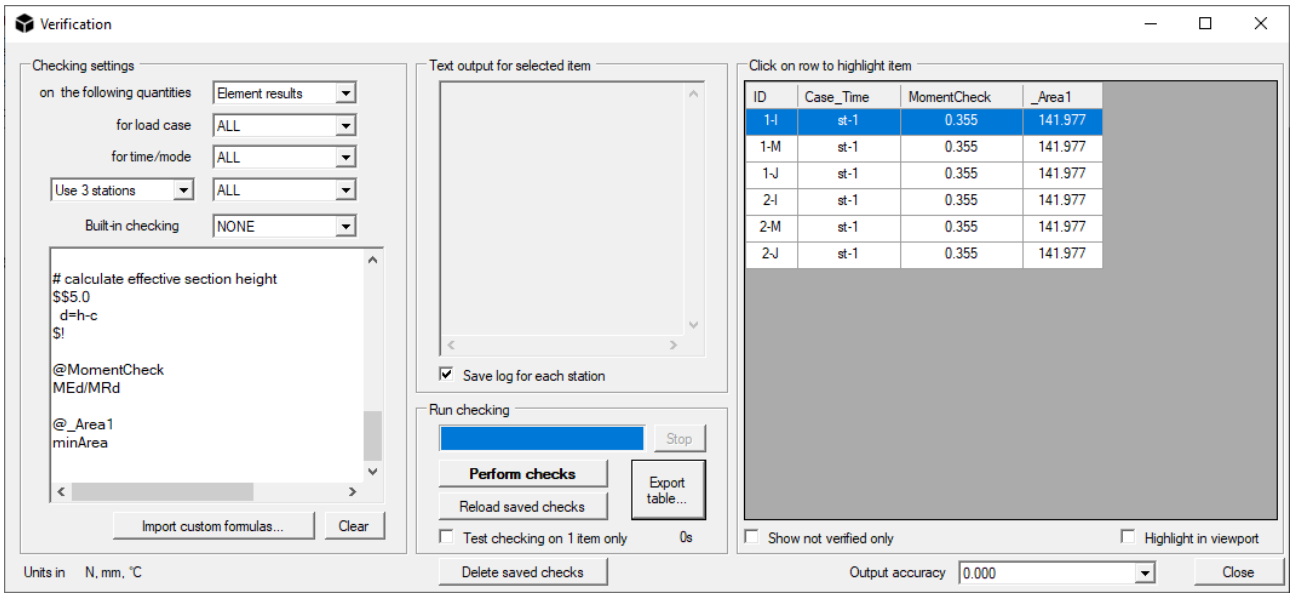


If you press the button "*Reload saved checks*", the program will compact the rows by using the maximum for each element.



To avoid highlighting in a certain column, declare it with "@_". For example:

@_Area1
minArea



Dimensioning concrete members

Using some special commands, it's possible to dimension concrete members by adding longitudinal and shear reinforcements on the base of custom relationships.

In the following, you can find a sample to dimension concrete beams by calculating the needed shear area under pure flexural behaviour.

 # dimensioning of rebar in a beam part of a rigid diaphragm – clear all rebar in current element
clearRebar
 # long. rebar strength fydl
fydl=unitconv(MPa,model_S,391.3)
 # stirrup strength
fyds=fydl
 # design moment - take maximum and minimum over beam
Mpos=MmaxZ
Mneg=MminZ

 # set rebar cover to 40mm
c=unitconv(mm,model_L,40)
 # calculate effective section height
d=h-c

 # estimate rebar area for design moments
*AsTop=max(abs(Mpos)/(0.9*d*fydl),0.0013*b*d)*
*AsBot=max(abs(Mneg)/(0.9*d*fydl),0.0013*b*d)*

 # insert rebar in the model - find bar diameter for 3 bars at bottom and 3 at top
nBars=3
*TopD=**getBarDiam**(AsTop/nBars)*
*BotD=**getBarDiam**(AsBot/nBars)*
*TopArea=pi*TopD^2/4*

```
TopArea=unitconv(mm^2,model_L^2,TopArea)
BotArea=pi*BotD^2/4
BotArea=unitconv(mm^2,model_L^2,BotArea)

# stirrups D=8mm
stirrArea=unitconv(mm^2,model_L^2,50)
stirrSpac=unitconv(mm,model_L,150)

# cycle for bars
i=1
xdim=(b-2*c)/(nBars-1)
execwhile(i<=nBars,1.0)
$$1.0
   # design material ID for bars = 2
   # X coord., Y coord., bar area, design material ID, Linit [0,1), Lend (0,1]
   # top
   addRebarL(-b/2+c+xdim*(i-1),h/2-c,TopArea,2,0,1)
   # bottom
   addRebarL(-b/2+c+xdim*(i-1),-h/2+c,BotArea,2,0,1)
   # shear reinforcements
   # legs in Y, legs in Z, bar area, spacing, design material ID = 2, Linit [0,1), Lend (0,1]
   addStirrups(2, 2, stirrArea, stirrSpac, 2, 0,1)
   i=i+1
$!
```

The code above estimates the needed rebar area to resist the bending moment around the major section axis. The instructions to add rebar to beams are:

- **addRebar**: adds longitudinal rebar to the current element.
  Usage: addRebar(zCoord, yCoord, rebarArea, design material ID, startAt [0,1), endAt (0,1]).
  Example: addRebarL(-b/2+40,-h/2+40,201.0,2,0,1)
  adds a rebar of area 201.0 at position 40, 40 from the bottom left corner of the section, for the whole element.
  From version 1.8.3, the position of the rebar must be given with respect to the center of the section.
- **addStirrups**: adds stirrups to the current element.
  Usage: addStirrups(legs in Y, legs in Z, single bar area, spacing, design material ID, startAt [0,1), endAt (0,1]).
  Example: addStirrups(2, 2, 50.0, 200, 2, 0,1)
  adds 2-by-2 legs stirrups, each one of area 50.0, at 200 of spacing, for the whole element.
- **clearRebar:** clear all rebar, including stirrups, in the current element.
- **getBarDiam**: gets the minimum diameter of a longitudinal bar required to satisfy the given area. The output is in mm. Example: getBarDiam(2.01) # for a model in cm, returns 16.

- **getStirrupDiam**: gets the minimum diameter of a stirrup bar required to satisfy the given area. The output is in mm. Example: getStirrupDiam(0.5) # for a model in cm, returns 8.

TranslateMomentYY and TranslateMomentZZ

As required by the codes (EC2 6.2.2 (5) and 6.2.3 (7)), due to the fragile behavior of the concrete and the formation of diagonal cracks, it is necessary to carry out the resistance verification of a section using an increased value of the tensile force on stretched bars.

In the practice of a structural verification procedure, this is equivalent to carrying out a translation of the envelope diagram of the bending moment, choosing the most penalizing combination of stresses. This translation distance is a function of the inclination of the concrete struts and the useful height of the section.

To get the translated value of the solicitation moment to be used in a verification script for NextFEM Designer, you can use the *TranslateMomentYY* and *TranslateMomentZZ* functions:

Produces the variables Mzzmax, Mzzmin

TranslateMomentZZ (deltaZZ, position)

Produces the Myymax, Myymin variables

TranslateMomentYY (deltaYY, position)

The required input parameters are:

- *deltaZZ*: the translation distance along the axis line for the moment around Z

- *deltaYY*: the distance of translation along the axis line for the moment around Y

- position: the position at which the values of the moment translated diagram are to be obtained. Measured from Node I along the axis line

The two functions respectively create the variables *Mzzmax, Mzzmin, Myymax* and *Myymin* which contain the maximum and minimum values, for the given position, of the moment diagrams.

If the element under test is part of a member consisting of more than a single element, the *TranslateMomentYY* and *TranslateMomentZZ* functions carry out the calculation using the bending moment diagrams of the contiguous elements if *position + delta* and *position - delta* fell outside the current element.

Finally, to dimension RC beams and walls you can use the following script, that uses block to distinguish which code to execute on the base of the current element.

```
# dimensioning of rebar in a beam part of a rigid diaphragm
clearRebar
# long. rebar strength fydl
fydl=unitconv(MPa,model_S,391.3)
# stirrup strength
fyds=fydl
# translate moment for current position (null translation) - major axis
#TranslateMomentZZ(0,pos)
```

```
# if beam go to block 1, if wall go to block 2
execif(isWall==0,1.0)
execif(isWall==1,2.0)

$$1.0
# design moment - take maximum and minimum over beam
Mpos=MmaxZ
Mneg=MminZ

# set rebar cover to 40mm
c=unitconv(mm,model_L,40)
# calculate effective section height
d=h-c

# estimate rebar area for design moments
AsTop=max(abs(Mpos)/(0.9*d*fydl),0.0013*b*d)
AsBot=max(abs(Mneg)/(0.9*d*fydl),0.0013*b*d)

# insert rebar in the model - find bar diameter for 3 bars at bottom and 3 at top
nBars=3
TopD=getBarDiam(AsTop/nBars)
BotD=getBarDiam(AsBot/nBars)
TopArea=pi*TopD^2/4
TopArea=unitconv(mm^2,model_L^2,TopArea)
BotArea=pi*BotD^2/4
BotArea=unitconv(mm^2,model_L^2,BotArea)

# stirrups D=8mm
stirrArea=unitconv(mm^2,model_L^2,50)
stirrSpac=unitconv(mm,model_L,150)

# cycle for bars
i=1
xdim=(b-2*c)/(nBars-1)
execwhile(i<=nBars,1.1)
$!

$$1.1
   # design material ID for bars = 2
   # X coord., Y coord., bar area, design material ID, Linit [0,1), Lend (0,1]
   # top
   addRebarL(-b/2+c+xdim*(i-1),h/2-c,TopArea,2,0,1)
   # bottom
   addRebarL(-b/2+c+xdim*(i-1),-h/2+c,BotArea,2,0,1)
   # shear reinforcements
   # legs in Y, legs in Z, bar area, spacing, design material ID = 2, Linit [0,1), Lend (0,1]
   addStirrups(2, 2, stirrArea, stirrSpac, 2, 0,1)
```

```
   i=i+1
$!

# wall
$$2.0
# design moment - take maximum and minimum over beam
Mpos=MmaxZ
Mneg=MminZ

# set rebar cover to 40mm
c=unitconv(mm,model_L,40)
# calculate effective section height
d=h-c

# estimate rebar area for design moments
As=max(max(abs(Mpos),abs(Mneg))/(0.9*d*fydl),0.003*A)

# insert rebar in the model - lattice reinforcement 20x20cm D=8mm
lt=unitconv(cm,model_L,20)
nBars=ceil((h-2*c)/lt)+1
# rebar diameter in mm
bd=8
RebArea=pi*bd^2/4
RebArea=unitconv(mm^2,model_L^2,RebArea)

# stirrups D=8mm
stirrArea=RebArea
stirrSpac=lt

# cycle for bars
i=1
xdim=lt
execwhile(i<=nBars,2.1)
$!

$$2.1
   # design material ID for bars = 2
   # X coord., Y coord., bar area, design material ID, Linit [0,1), Lend (0,1]
   rpos=if(c+xdim*(i-1)>h,h/2-c,-h/2+c+xdim*(i-1))
   # left
   addRebarL(-b/2+c,rpos,RebArea,2,0,1)
   # right
   addRebarL(b/2-c,rpos,RebArea,2,0,1)
   # shear reinforcements
   # legs in Y, legs in Z, bar area, spacing, design material ID = 2, Linit [0,1), Lend (0,1]
   addStirrups(2, 2, stirrArea, stirrSpac, 2, 0,1)
   i=i+1
$!
```

8 License agreement for using NextFEM API

NextFEM APIs are free to use and maintained and updated by NextFEM. The present license applies to any customization implemented by the user (for instance: plugins, sections and materials libraries, NVV verification files, etc.), without differences if such customization is distributed to the public or not.

The user accepts the NextFEM License in its entirety, and all following additional conditions:

I. NextFEM APIs are made available by NextFEM. Their availability can vary through time and versions, without warnings and/or advices.

II. NextFEM APIs files (e.g. .dll files) are not redistributable. Every used must be registered to the NextFEM website in order to download and install NextFEM Designer before accessing APIs.

III. Developers (companies, individuals, etc.) must not reproduce any paid function in NextFEM products (e.g. verifications, sectional checkers, meshing tools, etc.). If this rule is violated, NextFEM can block the call to its APIs at any time, without warning, to the specific third-party software violating this rule. Depending on the case, developer could be requested to sign an agreement to allow distribution.

IV. For the same functionality, plugins developed for NextFEM Designer must use the libraries that Designer already uses. For example, for plugins developed in .NET languages, serialization in JSON, where not covered directly in the referenced framework, must be done using the NewtonSoft library already present in NextFEM products. The addition of libraries for the same functionality would result in a violation of the present license agreement and, consequently, the blocking of the plugin.

V. Commercial distribution of third-party products based on or simply referencing NextFEM APIs is permitted only after explicit approval of the NextFEM commercial office, which can be contacted via email at licensing@nextfem.it.

VI. If needed, adoption of NextFEM license platform is strongly advised. To use it, please contact NextFEM commercial office at licensing@nextfem.it.