

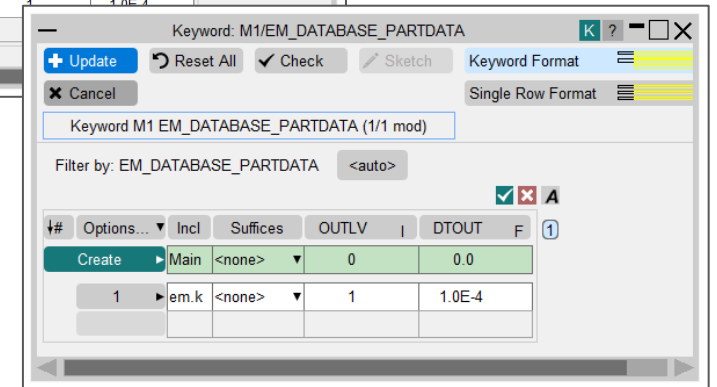
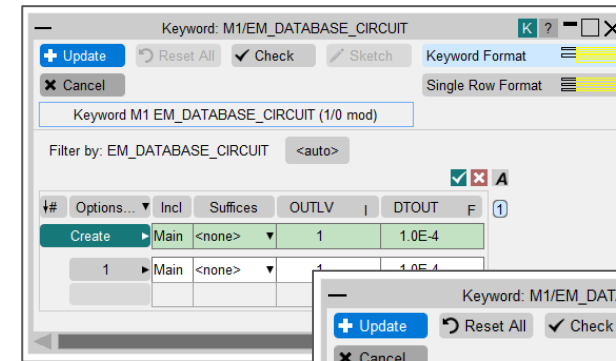
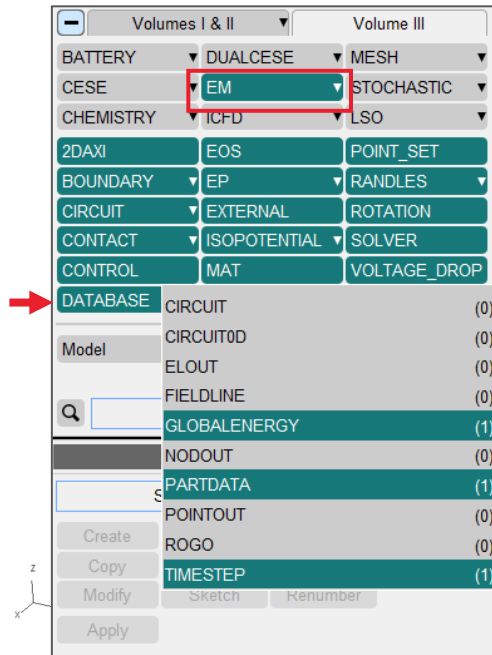
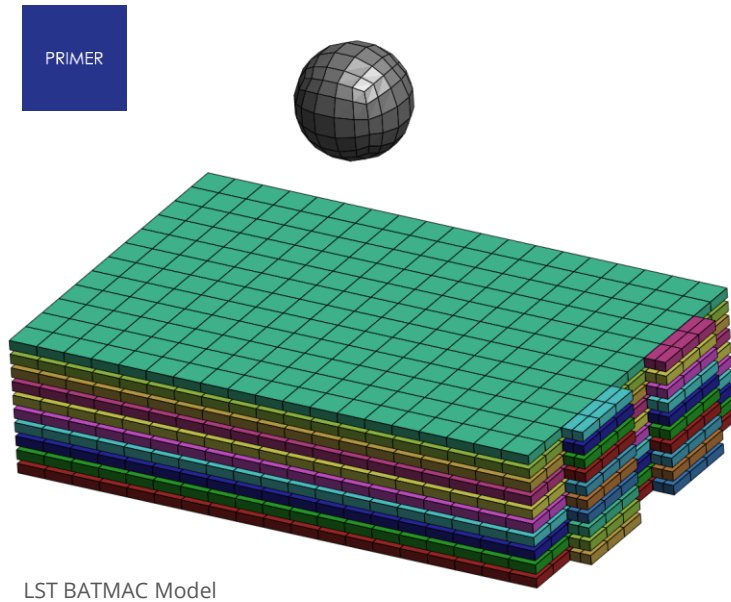
T/HIS 18.0

T/HIS 18.0 – Contents

- [EM Support](#)
- [New Data Components](#)
- [Reading ISO-MME Files](#)
- [Writing ISO-MME Files](#)
- [JavaScript API](#)
- [JavaScript GUI Builder](#)
- [JavaScript Engine Upgrade](#)
- [Checkpoint Files](#)
- [Miscellaneous](#)

EM Support

Additional EM ASCII Files supported



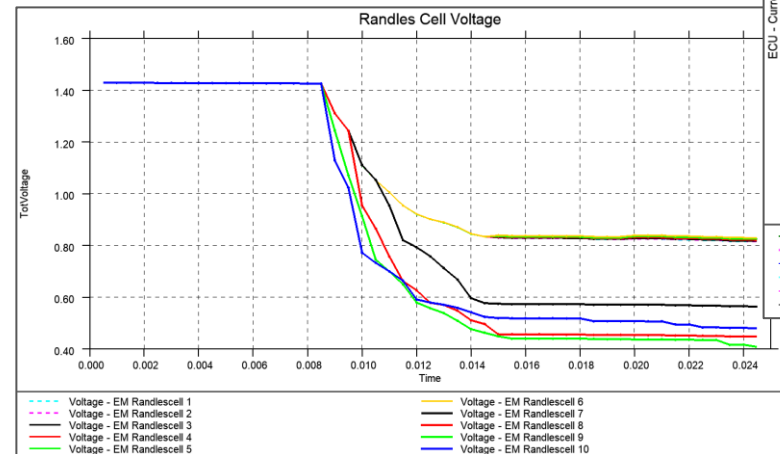
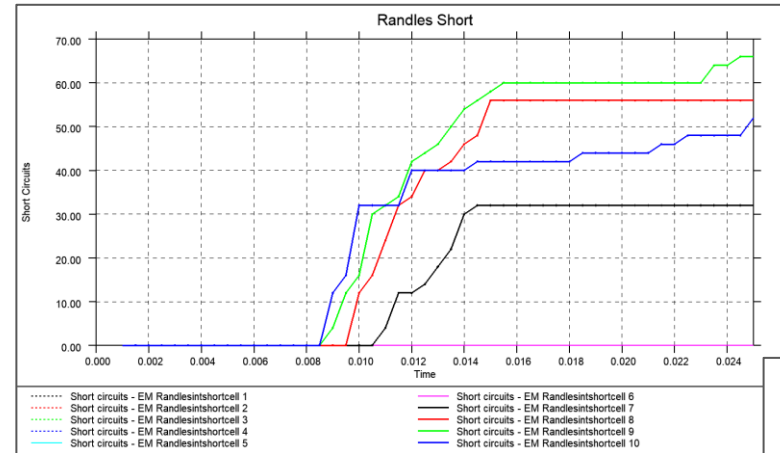
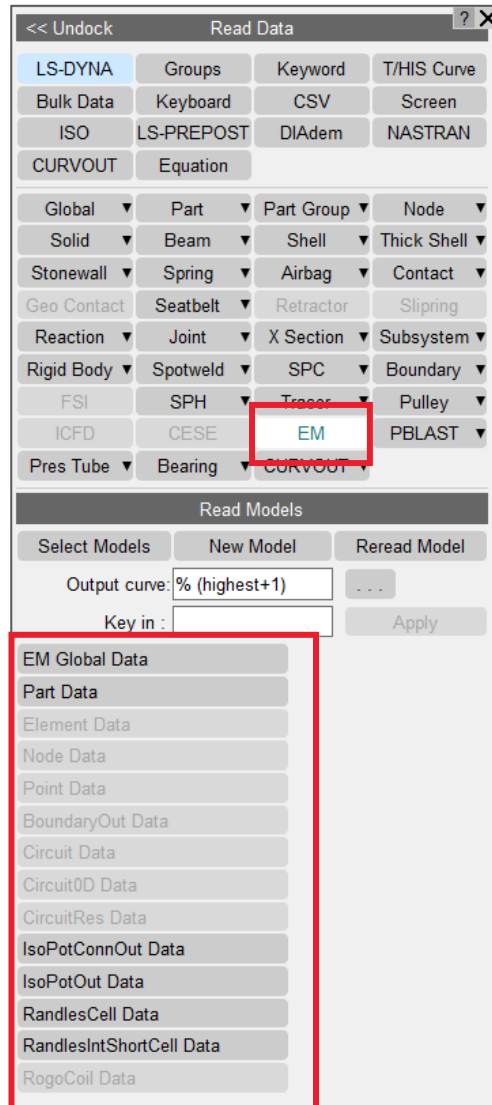
T/HIS now supports the following new fields:

- Circuit
- Circuit0D
- PartData
- IsoPotOut
- CircuitRes
- BoundaryOut
- IsoPotConnOut
- RandlesCell
- RandlesIntshortCell
- RogoCoil

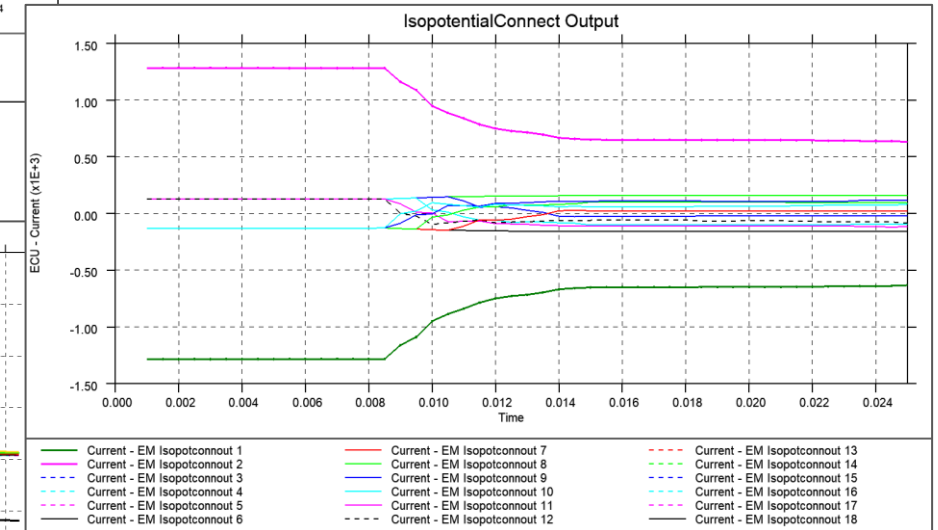
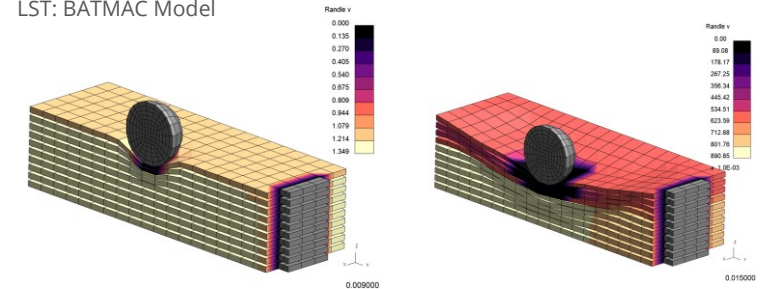
*These fields
can be found
in EM Global*

- Timestep
- RandlesCellTot
- RandlesCellTotEn
- GlobEnergy
- RandlesIntShort

Additional EM ASCII Files supported



LST: BATMAC Model



New Data Components

New Data Components

In addition to EM data components, support for the following new data components has been added:

Entity type	Component
Seatbelt Slipring	Warpage Angle
Seatbelt Slipring	Skew Angle
Seatbelt Slipring	Friction Coefficient
Seatbelt Slipring	Normal Force
Seatbelt Slipring	Side 1 Belt Force
Seatbelt Slipring	Side 2 Belt Force
Airbag	Mass in
Airbag	Mass out
Airbag	Mass flow rate out though fabric
Airbag	Mass flow rate out via vent
Airbag	Mass out through fabric
Airbag	Mass out via vent

Reading ISO-MME Files

Reading ISO-MME Files

The format of the channel index file has changed in v2.0 of the ISO-MME data exchange format ([ISO/TS 13499:2019](#)). T/HIS reads this to get the list of available channel data files.

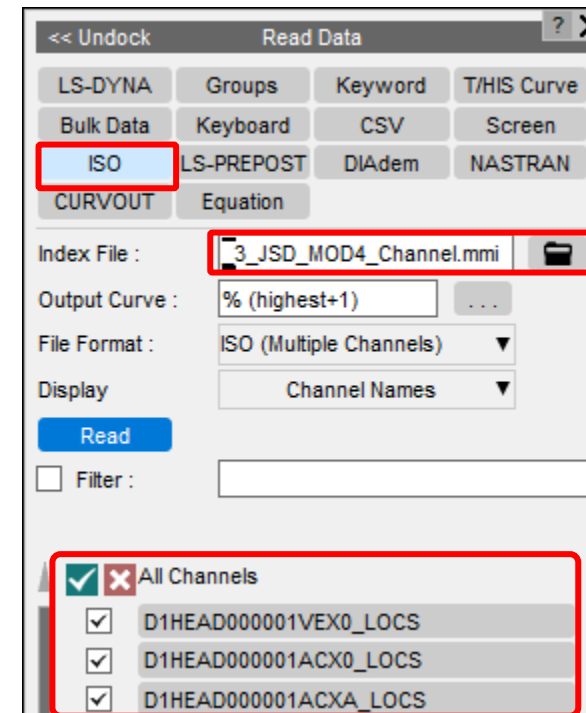
T/HIS 18.0 has been updated to read the new format (and is still able to read the earlier v1.6 format).

v1.6

```
Instrumentation standard :ISO 6487 (1987) / SAE J211 (MAR95)
Comments                 :Channels 4-10 have another reference system
Number of channels       :10
Name of channel 001      :DOHEAD00000PCACXA / Kopf SP1 / X
Name of channel 002      :DOHEAD00000PCACYA / Kopf SP1 / Y
Name of channel 003      :DOHEAD00000PCACZA / Kopf SP1 / Z
```

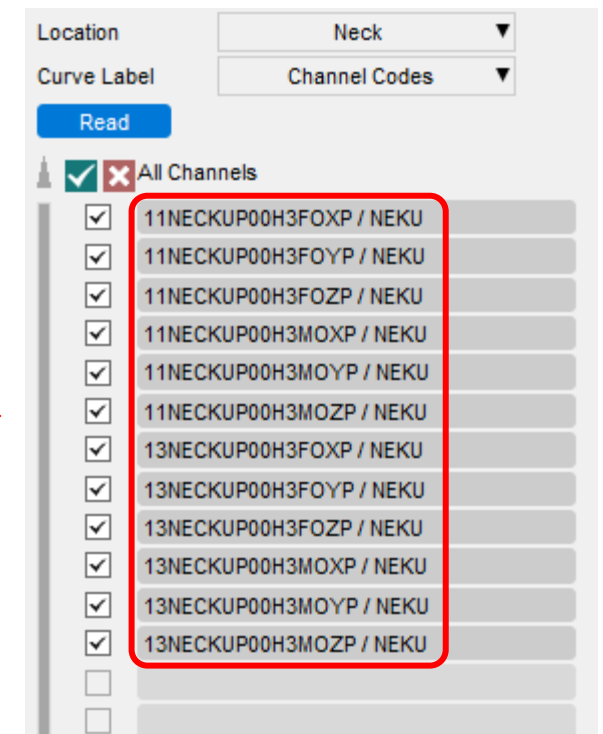
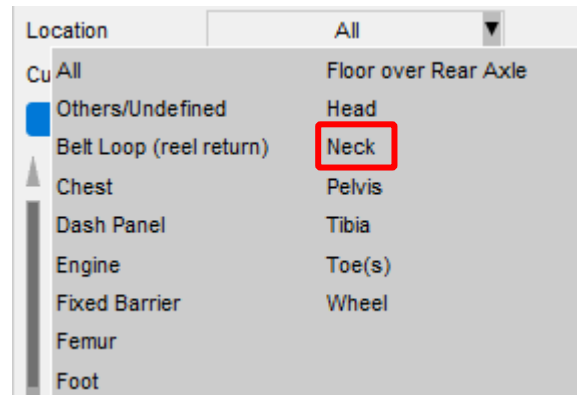
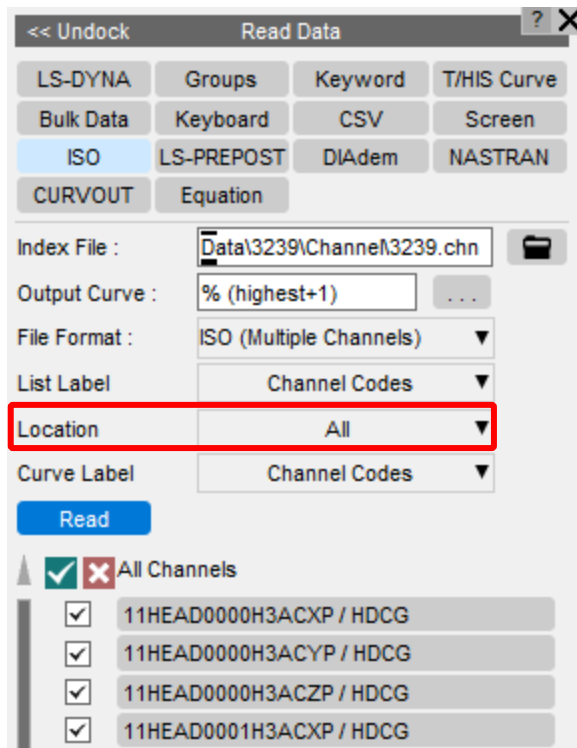
v2.0

```
Number of channels       :74
Reference system id      :LOC
Data origin              :S
Data source              :simulation
#Begin of channel
Extended channel code    :D1HEAD0000001VEX0_LOCS
#End of channel
#Begin of channel
Extended channel code    :D1HEAD0000001ACX0_LOCS
#End of channel
#Begin of channel
Extended channel code    :D1HEAD0000001ACXA_LOCS
#End of channel
```



Reading ISO-MME Files

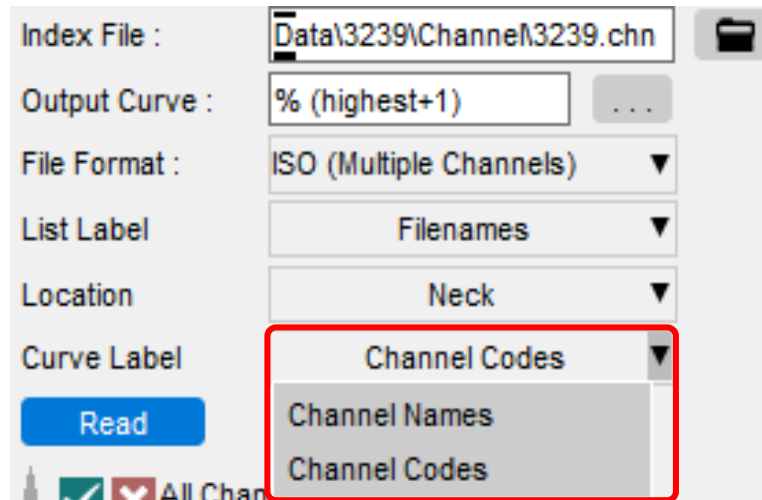
A 'Location' popup has been added to the **Read** → **ISO** menu to filter which channels are listed for selection. For example, to only list channels located in the neck:



Reading ISO-MME Files

A 'Curve Label' popup has been added to set the curve label format.

In earlier versions of T/HIS the channel name was used, but from T/HIS 18.0, they can now be labelled with the channel code instead. The channel code is now the default setting.



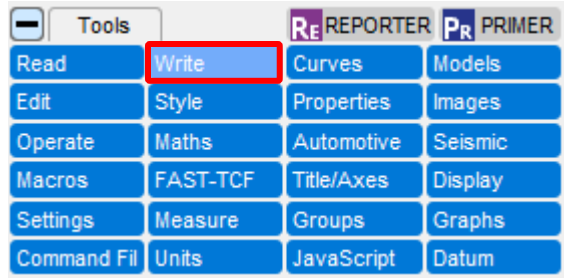
Channel name: — NEKU (N)

Channel code: — 11NECKUP00H3FOXP

Writing ISO-MME Files

Writing ISO-MME Files

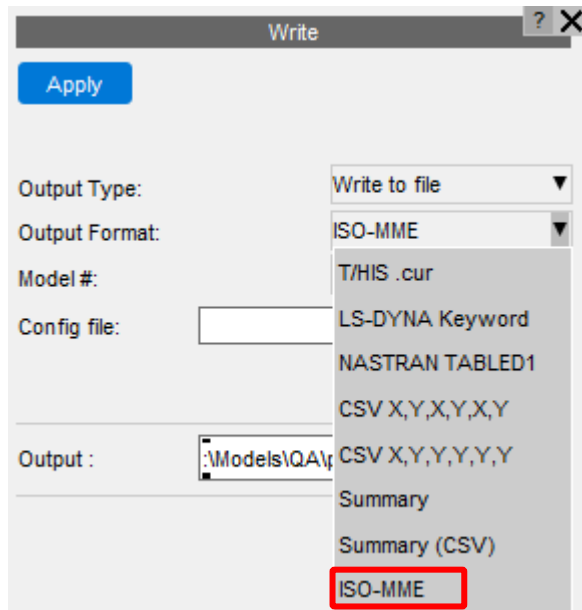
T/HIS 18.0 can now write curves in the ISO-MME format.



The ISO-MME option works slightly differently from the others in the Write menu. Rather than selecting output curves, you need to:

- Select a model from which T/HIS will extract the data
- Provide a configuration file to specify what data should be written

This is needed because the naming conventions of the output files (specified in the ISO standard [ISO/TS 13499:2019](#)) follow specific rules which require extra data that isn't present in the curves alone. ISO-MME files also contain header lines which describe the data in more detail. The contents of the configuration file are described in the T/HIS manual, and you can find a tutorial in T/HIS under **Help** → **Tutorials**.



JavaScript API

JavaScript API

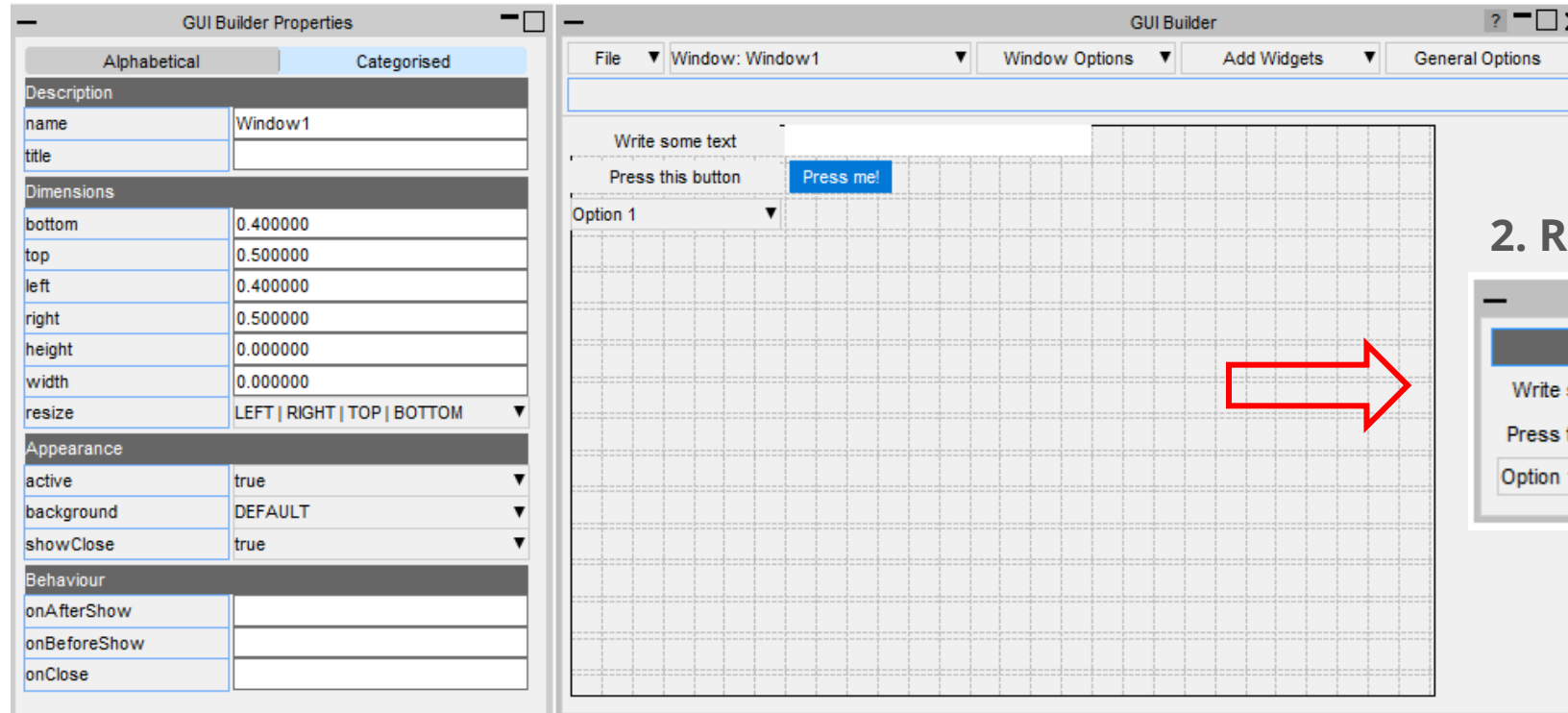
- From version 18.0, for speed, T/HIS does not update the curve manager menu while a JavaScript is running. If the JavaScript is interactive, e.g. uses curve picking, or is run from the debugger, the curve manager menu will be updated. This behaviour can be changed by setting the preference `this*javascript_update_curve_menu` to TRUE instead of FALSE (the default).
- Class constants have been added to the Read class, which can be used in the various Read methods.

JavaScript GUI Builder

JavaScript GUI Builder

An interactive GUI Builder has been added to PRIMER, D3PLOT and T/HIS to make it easier to build JavaScript GUIs, removing the need to write code to create windows and widgets.

1. Design and Save your GUI to a file

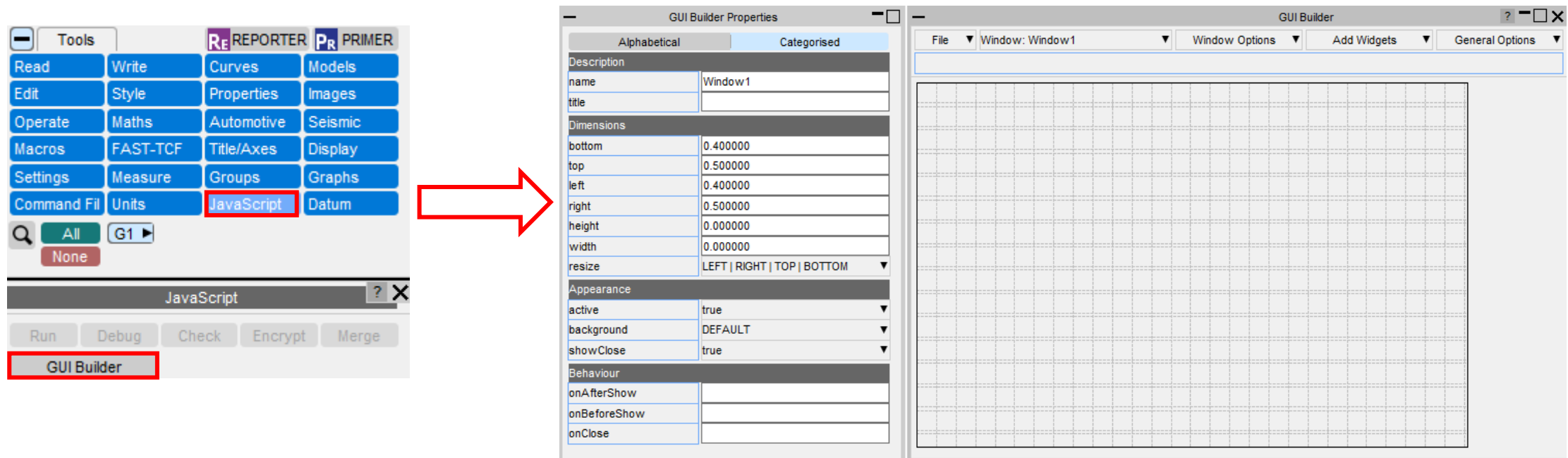


2. Read the file in your script



JavaScript GUI Builder

To open the GUI Builder in T/HIS go to JavaScript → *GUI Builder*



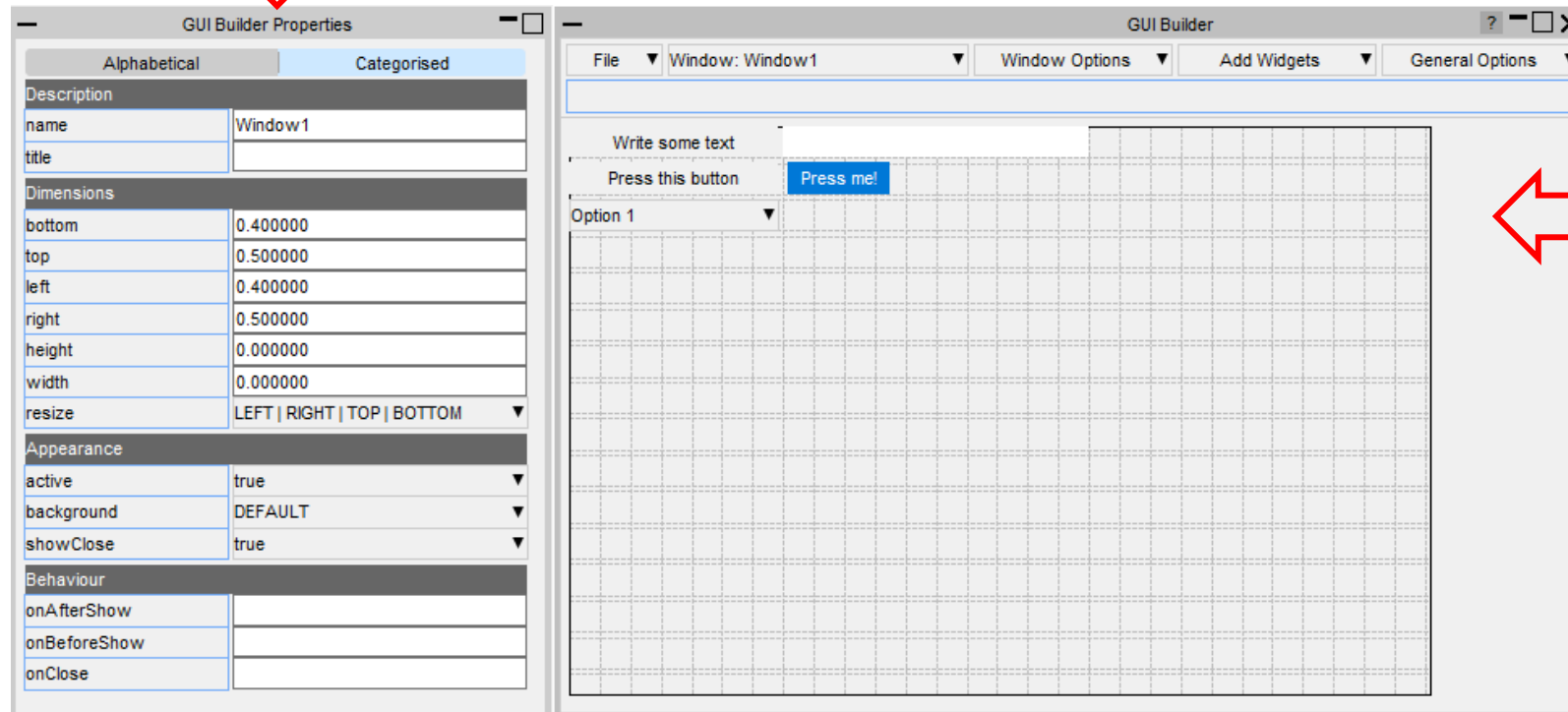
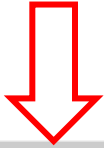
JavaScript GUI Builder

How to use the GUI Builder to build a GUI

JavaScript GUI Builder

Properties Window

The properties of widgets and windows are set here.

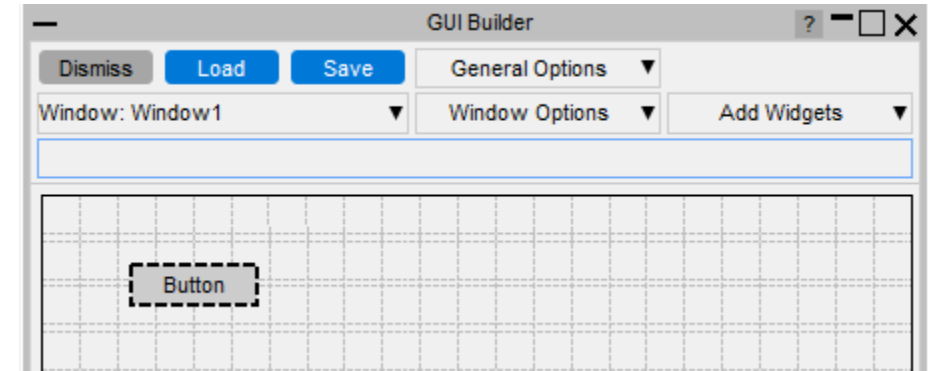
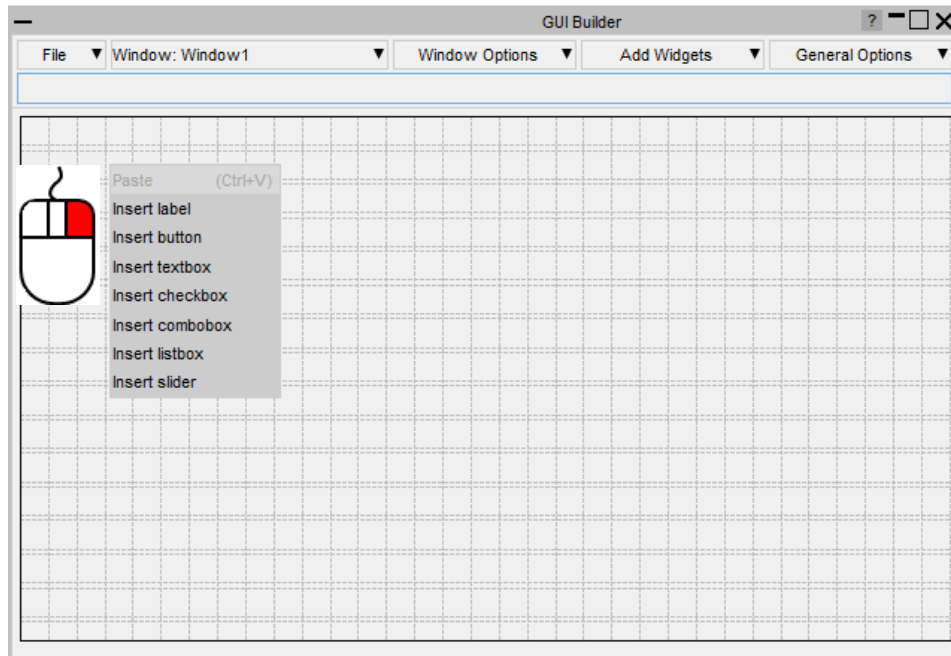


Design Window

Widgets are added, positioned and resized here.

JavaScript GUI Builder

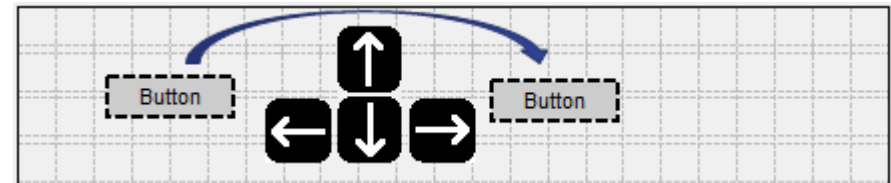
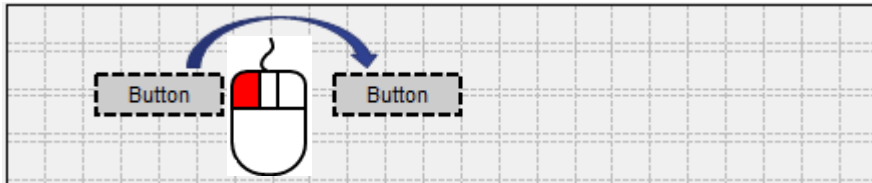
Widgets can be added by right-clicking on the design window and selecting the widget type to add.



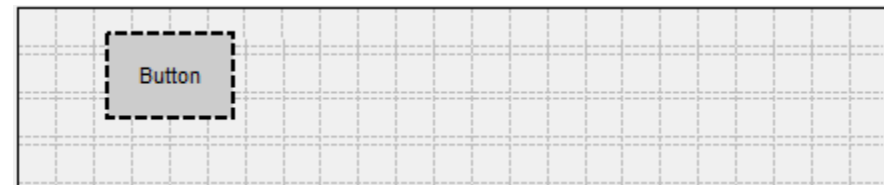
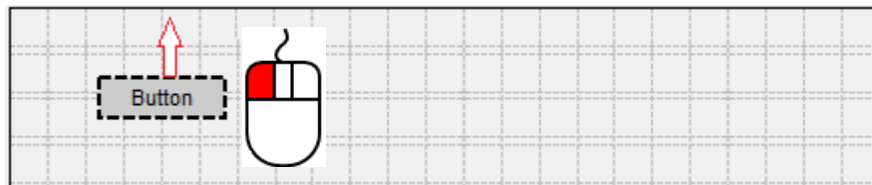
The widget will be added with default properties and highlighted with dashed lines to indicate that it's the current widget.

JavaScript GUI Builder

Widgets can be moved by left-clicking on them and dragging, or by using arrow keys

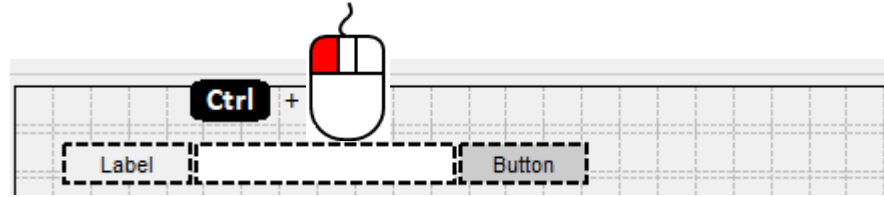


They can be resized by left-clicking on their border and dragging

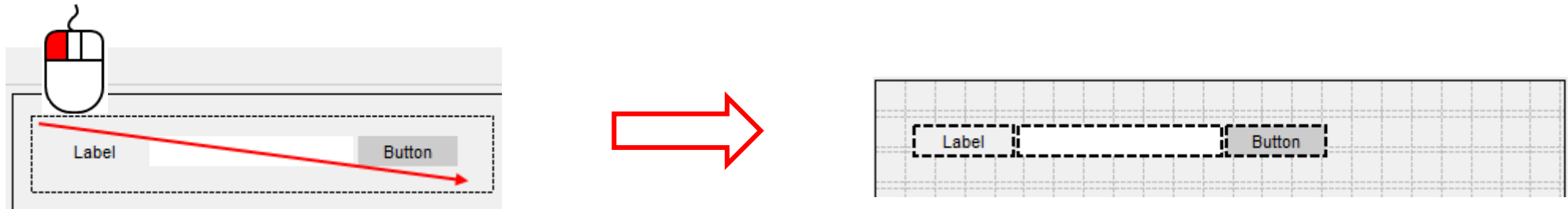


JavaScript GUI Builder

Multiple widgets can be selected by holding the Ctrl or Shift keys and left-clicking

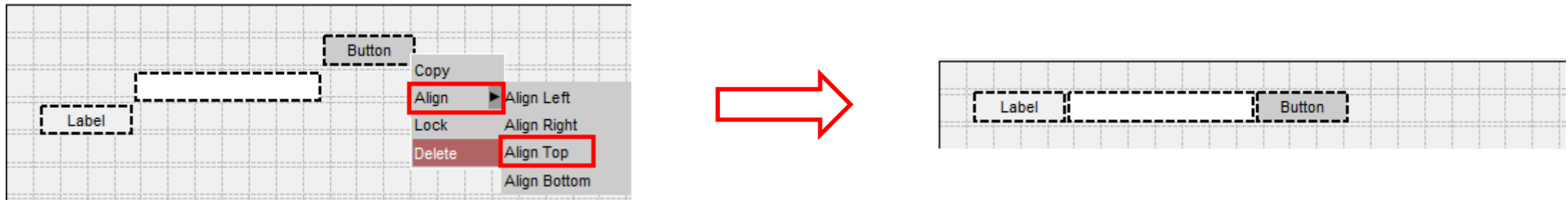


Alternatively a box can be dragged around the widgets you want to select



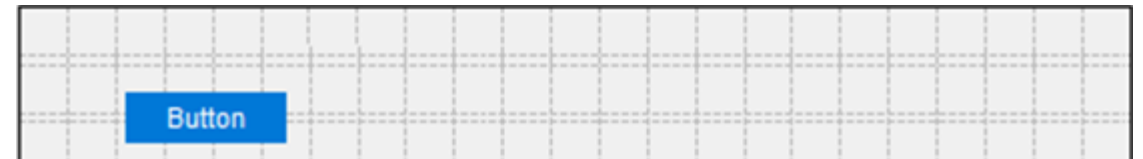
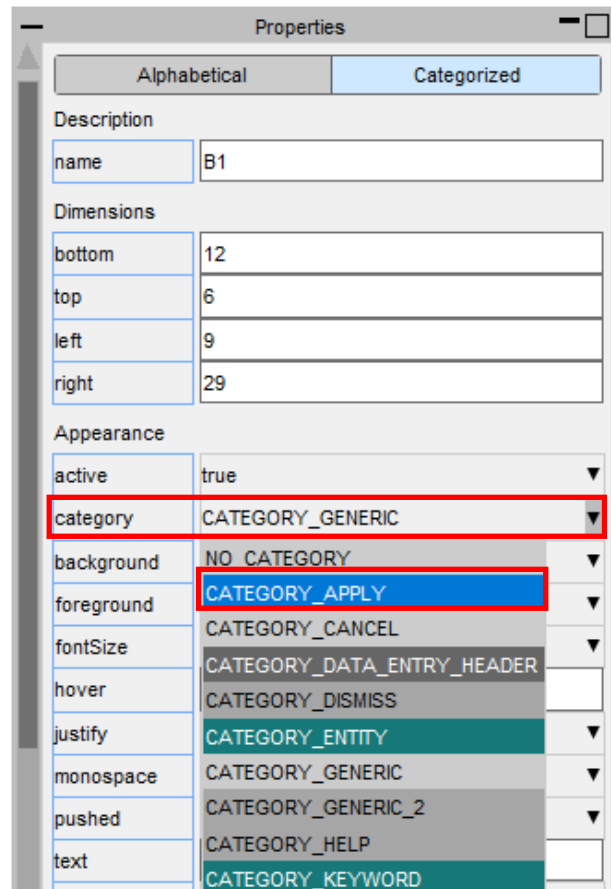
JavaScript GUI Builder

When multiple widgets are selected the borders can be aligned by right-clicking on the widget you want to align the other widgets to, and then selecting how you want them to be aligned:



JavaScript GUI Builder

The properties of a widget can be modified in the properties window, e.g. change the category to CATEGORY_APPLY:



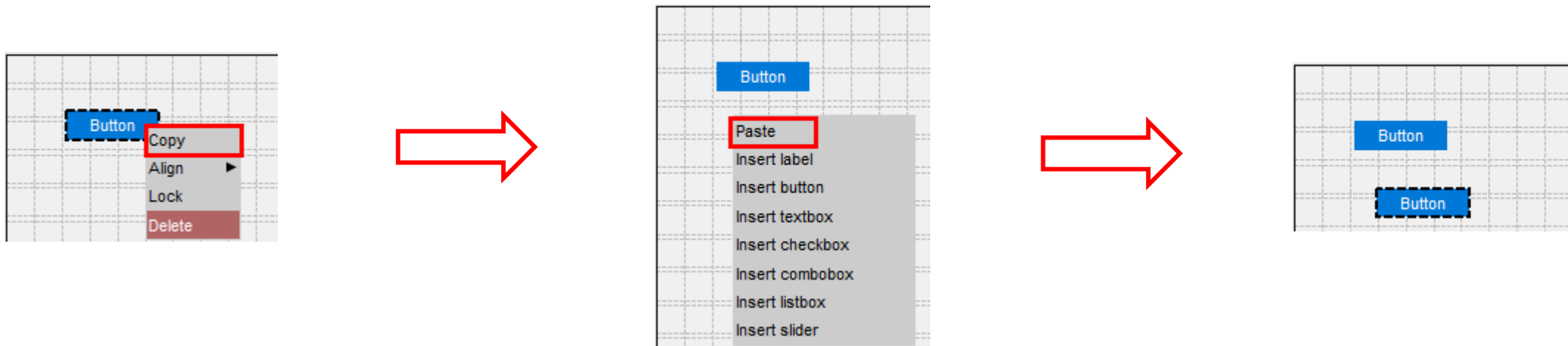
The appearance of the widget will update in the design window.

If multiple widgets are selected the property will be applied to all the selected widgets

JavaScript GUI Builder

You can copy and paste widgets by right-clicking on them and selecting 'Copy' and then right-clicking on the window and selecting 'Paste'. The new widget will have all the same properties as the copied widget.

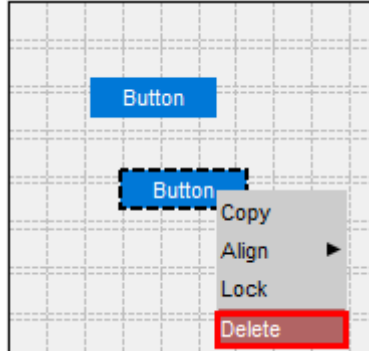
Alternatively you can use the shortcuts Ctrl-C and Ctrl-V.



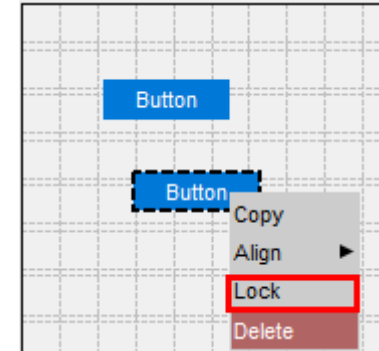
JavaScript GUI Builder

To delete a widget, right-click on it and select 'Delete'.

Alternatively you can press the Delete shortcut key.



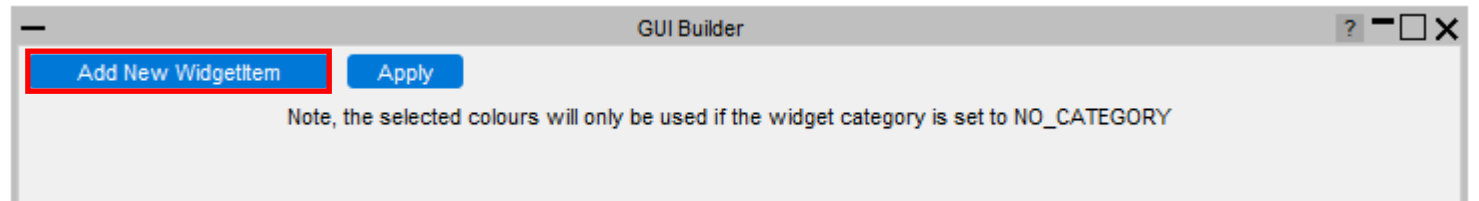
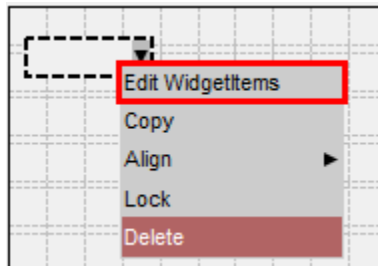
To lock the position of a widget so it can't be repositioned or resized, right-click on it and select 'Lock'. To unlock it again, right-click on it and select 'Unlock'.



JavaScript GUI Builder

To add WidgetItems to a Combobox or Listbox, right-click on it and select 'Edit WidgetItems'.

This will update the design window where you can add WidgetItems by pressing the 'Add New WidgetItem' button.

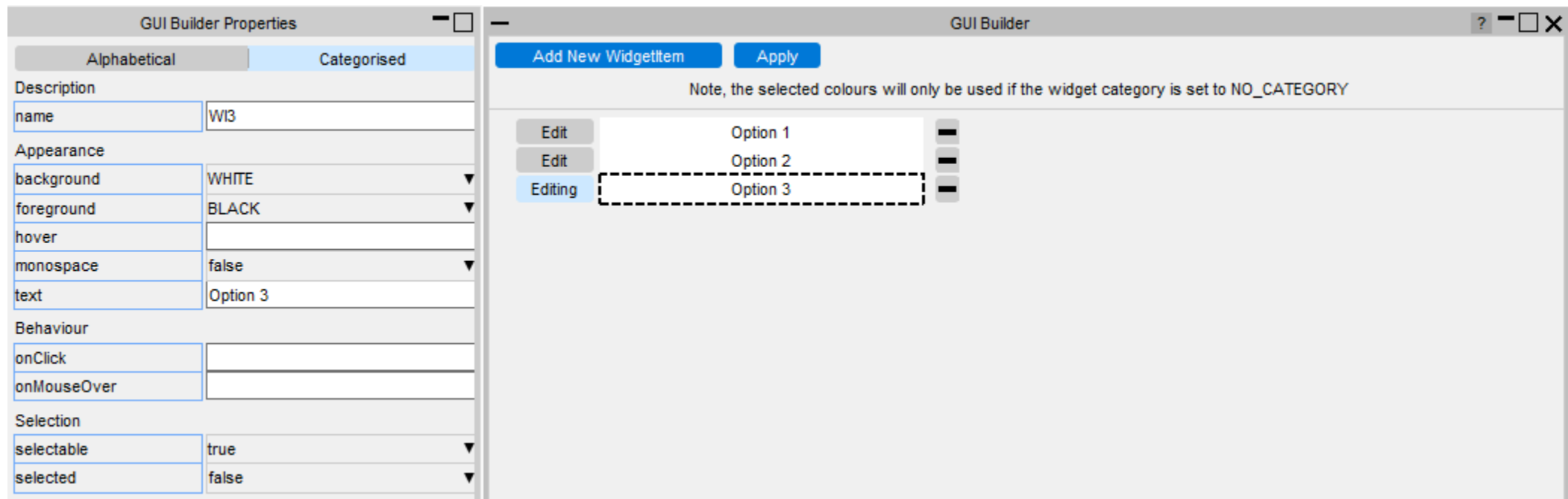


JavaScript GUI Builder

The appearance of the current WidgetItem can be modified in the same way as Widgets by clicking on the WidgetItem and updating its properties.

To delete a WidgetItem, click on the '-' on the right hand side.

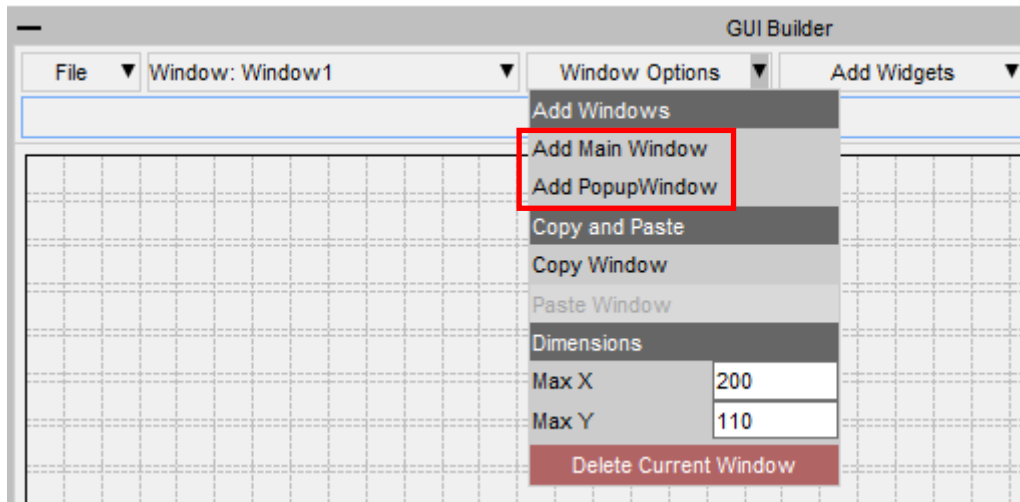
Once you have finished, press 'Apply' to return to the normal design window.



JavaScript GUI Builder

Additional windows can be created by clicking on the Window Options dropdown menu.

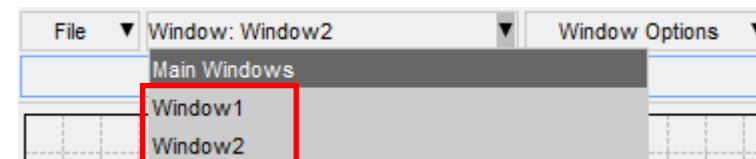
You can add either a Main Window or PopupWindow.



The name of the current window is displayed in the Window selection dropdown menu




To change to a different window, select it from the dropdown menu




JavaScript GUI Builder

PopupWindows can be linked to widgets by setting the popupWindow property.

Dimensions	
bottom	14
top	8
left	8
right	28
Popups	
popupDirection	BOTTOM ▼
popupSymbol	true ▼
popupWindow	<no popup> ▼
Timer	PopupWindow1
timerDelay	PopupWindow2
timerRepeat	<no popup>
Misc	
macroTag	



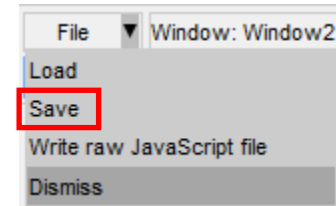
Dimensions	
bottom	14
top	8
left	8
right	28
Popups	
popupDirection	BOTTOM ▼
popupSymbol	true ▼
popupWindow	PopupWindow1 ▼
Timer	
timerDelay	1
timerRepeat	false ▼
Misc	
macroTag	



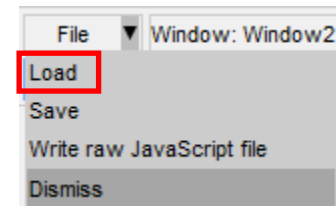
To remove a PopupWindow linked to a widget, set the popupWindow to <no popup>

JavaScript GUI Builder

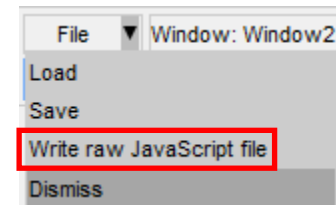
The GUI can be saved to file by pressing the 'Save' button and then selecting a file. The saved file is a JavaScript file containing the window and widget definitions in a JSON string, and a call to `Window.BuildGUIFromString()` which builds the GUI when the script is run. Further details are given in the next few slides.



It can be reloaded by pressing the 'Load' button and selecting the file to load.



The GUI can also be saved as a raw JavaScript file, with the calls to create and position the windows and widgets, explicitly defined, rather than using `Window.BuildGUIFromString()`. This cannot be loaded back into the GUI Builder, however it may be useful for creating GUI's to run in versions prior to v18 that don't have the `Window.BuildGUIFromString()` function.



JavaScript GUI Builder

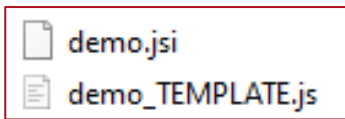
How to use the GUI in a script

JavaScript GUI Builder

The GUI is saved to a JavaScript file, containing the GUI definition in a JSON string and a call to `Window.BuildGUIFromString()`. It is saved with the extension `.jsi` to indicate that it should be included from another file. You should not need to edit this file.

A `*.js` file is also written to demonstrate how to include the `*.jsi` file and display the GUI. This can be used as a template to follow and modify.

It is written to the same folder as the `*.jsi` file and named `'<jsi_filename>_TEMPLATE.js'`, e.g. if the `*.jsi` file is called **`'demo.jsi'`**, the `*.js` file will be saved as **`'demo_TEMPLATE.js'`**



The following slides explain what is in the file and how you can reference the Windows, Widgets and WidgetItems in the script.

JavaScript GUI Builder

To read the GUI in a script you need to include the file using the Use() function.

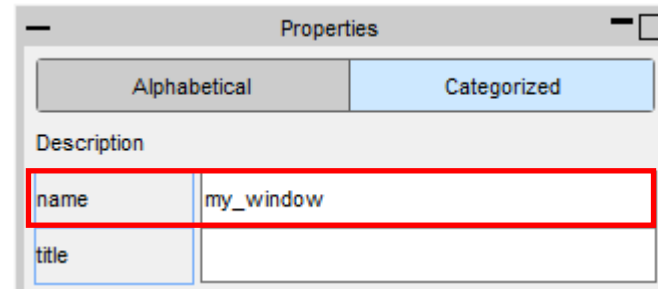
This will create a global variable ('gui' by default) containing all the GUI objects. The name of the variable can be changed in the GUI builder menu under General Options.

For example, to build the GUI saved in C:\my_gui.jsi:

```
Use("C:\\my_gui.jsi");
```

JavaScript GUI Builder

The GUI Window objects are stored as properties on the global object. The name of the property is whatever was defined in the properties window in the GUI builder:

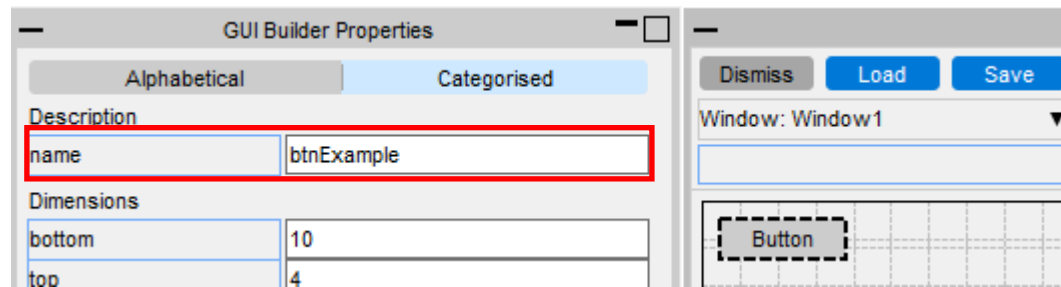


To display the Window called 'my_window' use the Show() method:

```
if (gui) gui.my_window.Show();
```

JavaScript GUI Builder

Similarly, each Widget object is a property of the Window object. The name of the Widget property is whatever was defined in the properties window in the GUI builder:



For example if the window is called 'my_window' and the widget is called 'btnExample', the Widget object can be accessed and modified with:

```
var btn = gui.my_window.btnExample;
```

```
btn.text = "Test";
```

JavaScript GUI Builder

WidgetItem objects are a property of the Widget.

For, example if the window is called 'my_window', the widget the widget item is on is called 'cbxExample' and the widget item is called 'wi1', it can be accessed and modified with:

```
var wi = gui.my_window.cbxExample.wi1;
```

JavaScript GUI Builder

Callback functions (onClick, onChange, etc.) can be assigned to the window and widgets in the properties window, by adding the name of a function to call.

For example to set the onClick property of a widget so it calls a function called 'pressed':

Functions	
onClick	pressed
onPopup	
onTimer	

This function then needs to be defined in your script:

```
Use("C:\\test.jsi");  
  
if (gui) gui.my_window.Show();  
  
function pressed()  
{  
    Message("You clicked me!");  
}
```

JavaScript Engine Upgrade

JavaScript engine upgrade

- For T/HIS 18.0 the JavaScript engine used in T/HIS has been significantly upgraded.
- In T/HIS 17.0 and earlier the engine only supported [ECMAScript 5](#) features.
- In T/HIS 18.0 the engine now supports [ECMAScript 6](#) (ES6) and many newer features.
 - The engine we use is [Spidermonkey](#) provided by Mozilla from the Firefox web browser.
 - For T/HIS 18.0 we are now using the current 'Extended Support Release' version (ESR78)
 - Future releases will continue to use the latest ESR version available.

JavaScript engine upgrade

- The primary reason for upgrading is to give access to newer JavaScript features
- In some cases newer JavaScript code people obtained/learned from books and/or the web and tried to use in T/HIS did not work in T/HIS 17.0 as we only supported ECMAScript 5.
- Upgrading the engine allows the latest ECMAScript 6 (ES6) language features to be used.
 - Which ES6 (and newer) features are supported by the engine can be viewed at <http://kangax.github.io/compat-table/es6/#firefox78>
- Additional benefits to upgrading as well as ES6 support are outlined on the following slides.

JavaScript engine upgrade – ES6 features

- Upgrading the JavaScript engine gives access to lots of significant new ES6 (and newer) language features such as

- [class](#) keyword
- Block scope with [let/const](#)
- [Promises](#)
- [Arrow functions](#)
- [Default parameters](#), [rest parameters](#) and [spread syntax](#)
- [Set](#) and [Map](#)
- [Iterators](#) and [generators](#)
- [Symbol](#)

And many more

- A few examples follow but read a good book (e.g. JavaScript: The Definitive Guide) or look online for more details

JavaScript engine upgrade – example ES6 features

- class keyword
 - ES6 makes it much easier to create classes using the new class keyword and syntax

ES 5

```
function Circle(radius)
{
    this.r = radius;
}

Circle.prototype.area = function()
{
    return Math.PI * this.r * this.r;
}

var c = new Circle(5);
Message("Area of circle with radius " +
        c.r + " is " + c.area() )
```

ES 6

```
class Circle
{
    constructor(radius)
    {
        this.r = radius;
    }

    area()
    {
        return Math.PI * this.r * this.r;
    }
}

var c = new Circle(5);
Message("Area of circle with radius " +
        c.r + " is " + c.area() )
```

JavaScript engine upgrade – example ES6 features

- let statement
 - The let statement in ES6 allows you to create variables with block scope (variables declared with var have scope for the containing function which can be a source of bugs)
 - Accessing variables defined with let before they are initialised is an error (helps trap bugs)

ES 5

```
function test()
{
    Message(x);      // undefined

    var x = 1;
    {
        var x = 2;    // same variable!

        Message(x);  // 2
    }

    Message(x);      // 2
}
```

ES 6

```
function test()
{
    Message(x);      // Error

    let x = 1;
    {
        let x = 2;    // different variable

        Message(x);  // 2
    }

    Message(x);      // 1
}
```

JavaScript engine upgrade – example ES6 features

- Spread operator
 - The spread operator expands an array into the list of values in the array. It can be useful when array values are needed in a function.

ES 5

```
// modify a curve point
var pt = curve.GetPoint(10);
pt[1] = Math.max(pt[1], 1.0);
curve.SetPoint(10, pt[0], pt[1]);

// draw a rectangle on a widget
var pt1 = [0, 0];
var pt2 = [100, 100];
widget.Rectangle(Widget.RED, true,
                 pt1[0], pt1[1],
                 pt2[0], pt2[1]);
```

ES 6

```
// modify a curve point
var pt = curve.GetPoint(10);
pt[1] = Math.max(pt[1], 1.0);
curve.SetPoint(10, ...pt);

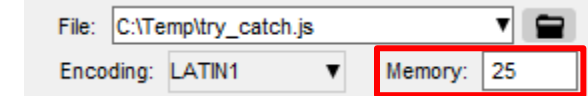
// draw a rectangle on a widget
var pt1 = [0, 0];
var pt2 = [100, 100];
widget.Rectangle(Widget.RED, true,
                 ...pt1,
                 ...pt2);
```

JavaScript engine upgrade – other benefits

- Memory consumption
 - JavaScript uses 'garbage collection' to manage any memory that needs to be used for a script.
 - Every object, array or string you use needs to store a small amount of data to be able to do this.
 - This storage in T/HIS 18.0 is approximately 2/3 of the size in T/HIS 17.0.

With the default memory size of 25Mb

- T/HIS 17.0 could create ~350,000 objects.
- T/HIS 18.0 can now create ~500,000 objects



JavaScript engine upgrade – other benefits

- Speed
 - Scripts which do a lot of mathematical operations will be faster (~ x3.5 speed increase in our tests).
 - String manipulation in scripts is faster (~ x3 speed increase in our tests).
 - Regular expressions in scripts are faster (~ x2.5 speed increase in our tests).
 - Several other features may see some speed increase from these and other improvements.

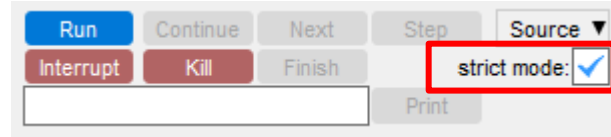
JavaScript engine upgrade – other benefits

- Debugger
 - The implementation of the debugger has also changed with the new engine.
 - Stepping through code using 'Step' and 'Next' is now significantly faster compared to T/HIS 17.0, especially for scripts with many lines and/or functions.
 - In T/HIS 17.0 try and catch did not work properly in the debugger. For example, the following script would always fail with an exception in the debugger instead of 'catching' it. This now works correctly in T/HIS 18.0.

```
var o = {};  
try  
{  
    o.UndefinedMethod();  
}  
catch(err)  
{  
    Message(err);  
}
```

JavaScript engine upgrade – other benefits

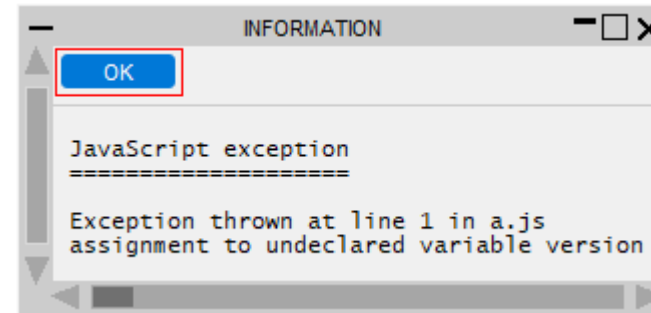
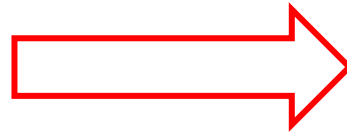
- Debugger



- Strict mode

- By default the debugger now works in 'strict mode' (see https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode for details) as this helps to find potential errors.
 - This has changed since T/HIS 17.0. In T/HIS 17.0 the 'strict mode' checkbox actually added some more checks to the debugger. It did not enforce 'strict mode'. This has been corrected for T/HIS 18.0.

```
version = 18.0;  
Message(version);
```



- This is equivalent to running the script

```
"use strict";  
  
version = 18.0;  
Message(version);
```

- This behaviour can be turned off if required using the checkbox.

JavaScript engine upgrade – other benefits

- Better checking
 - The new engine has better checking. For example, in the following code an error will be given when compiling that some code is unreachable (as there are { } missing so the return is not part of the if block and is always evaluated).

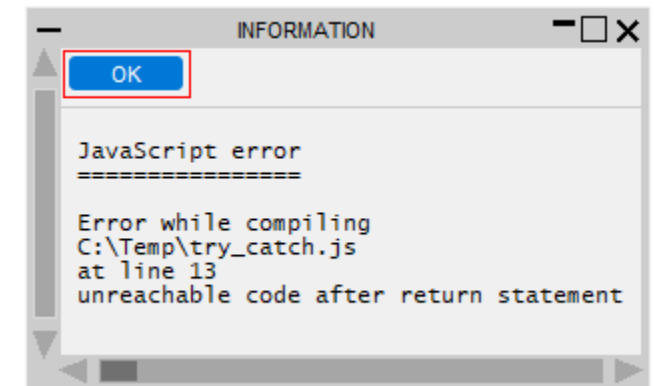
```
var vector = [ 1.0, 0.5, -0.2 ];
var length = vectorLength(vector);

function vectorLength(v)
{
    var l = 0;

    if ( !(v instanceof Array) )
        ErrorMessage("vectorLength not called with array");
        return null;

    for (var i=0; i<v.length; i++)
    {
        l += v[i]*v[i];
    }

    return Math.sqrt(l);
}
```



JavaScript engine upgrade – important changes

- ES6 Modules have not been implemented yet.
 - Upgrading the JavaScript engine has enabled ES6 (and newer) features to be used.
 - Modules are one ES6 feature that require significant changes in our software to implement and we are still resolving these.
 - For T/HIS 18.0 we want users to benefit from all the other ES6 features so have released the new engine without module support instead of waiting until we resolve this.
 - Support for ES6 modules will be added in a future release.

JavaScript engine upgrade – important changes

- `hasOwnProperty()` bug in T/HIS 17.0 and earlier
 - The JavaScript engine from T/HIS 17.0 (and earlier) contained a bug which meant that for the classes we define, object properties that were inherited from the object prototype appeared to be own properties of the object.
 - For example a Window object inherits properties title, left, right, top, bottom etc. from its prototype.
 - In T/HIS 17.0 this bug makes these properties appear to be an own property of the window as well as the prototype.
 - If you relied on this feature (unlikely) you will have to modify your code

```
var w = new Window("Test", 0.8, 1.0, 0.5, 0.6);  
w.dog = "Bark";
```

```
Message(w.hasOwnProperty('title'));           // false. w does not have own property title. true in 17.0 (bug)  
Message(w.hasOwnProperty('dog'));           // true. w does have own property dog
```

```
Message(w.__proto__.hasOwnProperty('title')); // true. title is inherited from prototype  
Message(w.__proto__.hasOwnProperty('dog'));   // false. dog is not inherited from prototype
```

JavaScript engine upgrade – important changes

- Script encoding differences

- In T/HIS 17.0 the default encoding used for scripts by the engine was 'Latin-1'.
- However on Windows this was actually implemented using the default encoding, which for many countries is Windows-1252.
- In T/HIS 18.0 the upgraded engine now compiles scripts as UTF-8 instead by default.
 - For scripts that just use ASCII (English) characters this will make no difference.
 - However the Windows-1252 encoding also contains special characters such as special quotes , apostrophes, en-dash and em-dash characters (‘ ’ “ ” – —) and these are incompatible with UTF-8.
 - If your script contains these characters it will no longer compile as 'Latin-1' (and it would also not have run on Linux in T/HIS 17.0 and earlier). Either remove the characters or save the script in a different encoding using your editor.
- Setting a specific encoding for a script such as Shift-JIS or UTF-8 is unaffected.



JavaScript engine upgrade – important changes

- Extra checking **may** occasionally mean old scripts that ran in T/HIS 17.0 no longer compile in T/HIS 18.0.
 - As the updated engine has better checking (such as the check for unreachable code mentioned earlier) in some rare cases it may mean that a script which worked in T/HIS 17.0 will fail to compile in T/HIS 18.0 until the error is fixed.

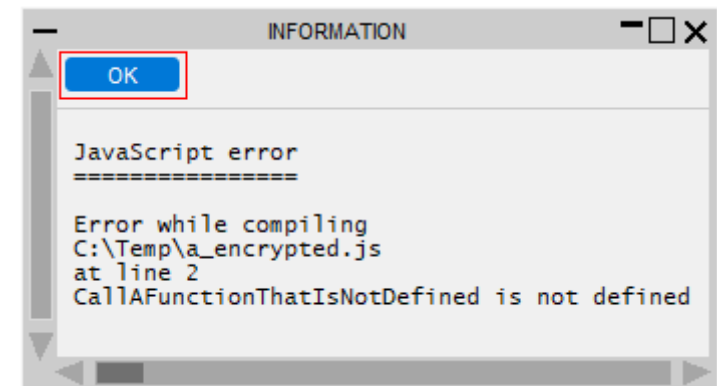
JavaScript engine upgrade – important changes

- Error messages have been enabled for encrypted scripts
 - In T/HIS 17.0 if a script was encrypted no error messages would be given when compiling/running.
 - For example if the following script was encrypted

```
Message("Starting...");  
CallAFunctionThatIsNotDefined();  
Message("Done.");
```

no error message would be given when the script tried to run the undefined function. This could make it very hard to determine the cause of a 'released' script failing.

- As the upgraded engine has better checking and there may be some rare cases when scripts don't run we have now changed this for T/HIS 18.0 so error messages will now be given for encrypted scripts.



Checkpoint Files

Regulate Read/Write of the Checkpoint Files

Checkpoint Files

The following preferences will regulate the checkpoint files read/write.

T/HIS also has the command-line options with the same names.

- **this*write_checkpoint_files**: Enable/disable the recording of the checkpoint files upon T/HIS startup. Valid values are TRUE/FALSE (default is FALSE)
- **this*show_checkpoint_files**: Enable/disable the reading of the checkpoint files upon T/HIS startup. Valid values are TRUE/FALSE (default is FALSE).
 - If the writing of the checkpoint files is disabled, the reading will also be automatically disabled.
- **this*checkpoint_dir**: Specify the folder path to write the checkpoint files to and also read the checkpoint files from.

Miscellaneous

Keyboard Shortcut Additions

- **Shift + Left Arrow** : Highlights from the current cursor position to the left by one character
- **Shift + Right Arrow** : Highlights from the current cursor position to the right by one character
- **Shift + Up Arrow** : Highlights from the current cursor position to the left-most character in the string (0 or prefix)
- **Shift + Down Arrow** : Highlights from the current cursor position to the right-most character (length of the string)
- **Ctrl + Left Arrow** : Jumps the cursor from the current cursor position to the left-most character of the word
- **Ctrl + Right Arrow** : Jumps the cursor from the current cursor position to the right-most character of the word
- **Ctrl + Shift + Left Arrow** : Highlight the rest of the word to the left of the cursor position up to the breaking character
- **Ctrl + Shift + Right Arrow** : Highlight the rest of the word to the right of the cursor position up to the breaking character
- **Double Click Left Mouse Button** : Highlights the whole word
- **Triple Click Left Mouse Button or Ctrl + A** : Highlights the whole line

Contact Information

ARUP

www.arup.com/dyna

For more information please contact us:

UK

T: +44 121 213 3399
dyna.support@arup.com

China

T: +86 21 3118 8875
china.support@arup.com

India

T: +91 40 44369797 / 98
india.support@arup.com

USA West

T: +1 415 940 0959
us.support@arup.com

or your local Oasys distributor