

Scripting with Mnova

MestReNova: NMR plugin

Pablo Monje, Ph D

www.mestrelab.com

Benefits and uses

The availability of scripting in Mnova provides opportunities and benefits both for skilled NMR and casual, non-expert users:

- Users are less dependent on the application developers for new features since scripting allows them to implement minor features themselves which can be deployed to less expert users.
- NMR administrators can write scripts to customize applications before deploying them to a group of users. For example, an NMR manager could create a script to automate some particular processing which he can distribute among his users in such a way that they will simply have to click one button (e.g. toolbar button) to apply that processing.
- Mnova developers can use scripting to prototype new features that can later be incorporated into the next version of the application. This is ideal when a client requires some processing/analysis feature not available in the application and cannot wait until a new version of Mnova is released.
- Mnova scripting language is based on an ECMAScript-based language making it easier for existing Web developers to customize and extend Mnova.

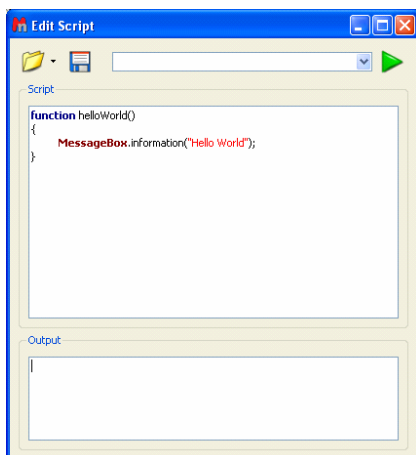
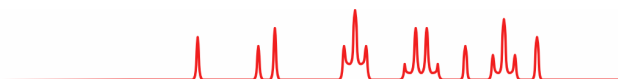
In essence, nearly everything that can be done with the Mnova graphical interface (UI) can be reproduced with Mnova's scripting module, making it very easy to automate repetitive tasks. For example, you may have to routinely process 1H NMR spectra using the same processing operations. You can create a script which wraps up all these processing functions in such a way that by simply clicking a toolbar button or pressing a keyboard shortcut, all of your 1H NMR spectra will be processed accordingly.

Furthermore, if some particular processing function is not available in Mnova, you can create a script which modifies the spectral data points according to your requirements.

Introducing Mnova Script – Hello World

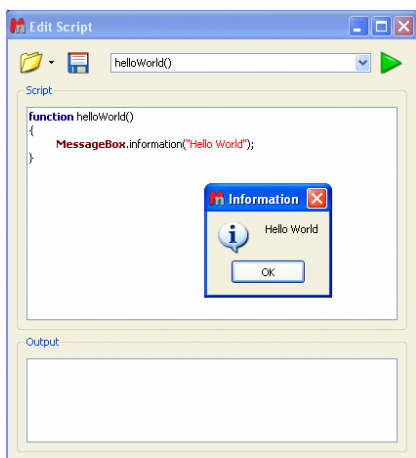
A "hello world" program has become the traditional first program that many books use to introduce a programming language. It will act as a cheatsheet for you until you familiarize yourself with the language. The point of this application is to use all the most commonly used elements of Mnova Script language and putting it in a single place so that you can refer to it later. We will start showing the basic elements of an Mnova script and from that point some more advanced methods will progressively be added.

Mnova scripts can be created with any text editor, although Mnova includes a specialized script editor with syntax highlighting and debugging output capabilities. This text editor is available from the Scripts/Edit Script menu command. Issue this command and type in the following text:

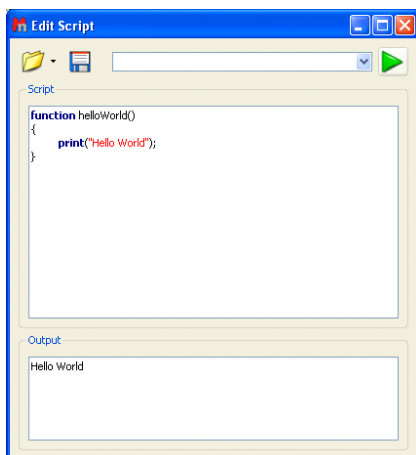
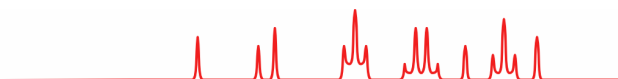


In this example we have defined a function called **helloWorld()** which will simply display a message box with the text 'Hello World'. Note that **MessageBox** is in bold, meaning that Mnova script editor recognizes it as a valid function.

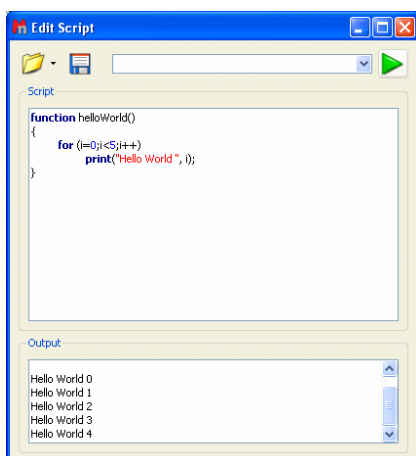
Next, go to the edit control at the top of the script editor and type **helloWorld()**. Do not forget the brackets as they are required by the scripting framework in order to properly interpret the function. Once you have typed it, click on the green arrow at the right side of the edit control. You should get the pop up message box.



It is also possible to write information directly on the **output** control in the script dialog by using the **print** command as showed below. This is very useful when debugging scripts.



As Mnova scripting is an ECMAScript-based language, you can use any of the many functions and statements supported by this standard such as loops, conditions, etc. Remember that JavaScript is ECMAScript based too so if you are familiar with it or with C/C++, writing Mnova scripts should be very easy for you. If you are not familiar with the ECMAScript language (<http://doc.trolltech.com/4.3/ecmascript.html>), there are several existing tutorials and books that cover this subject, such as [JavaScript: The Definitive Guide](#).

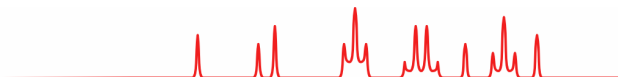


Where should the scripts be saved?

In principle you could save your scripts anywhere in your computer. However, it's strongly recommended to save them in the **scripts** folder which you will find in your application data folder. In Windows XP this is usually located in:

```
C:\Documents and Settings\user\program data\Mestrelab Research  
S.L\MestReNova\scripts
```

This is very important to allow Mnova to find these scripts when starting in such a way that they can be added to the GUI of Mnova (e.g. Toolbars, Menus, etc).



For example, type the following script:

```
// <GUI menuname="Hello" shortcut="Ctrl+1" tooltip="Hello World" />
function HelloWorld()
{
    MessageBox.information("Hello World");
}
```

And save it in the scripts directory (program data ...) . Now close Mnova and open it again.

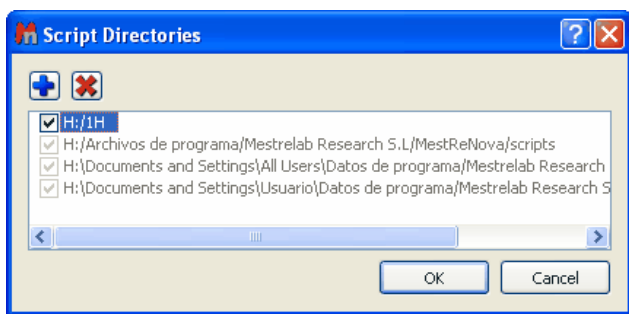


You will appreciate that the script is available in the menu bar. The key to get this working lies in the first line of the script (and of course, in the fact that Mnova can locate the script as it has been saved in the appropriate folder):

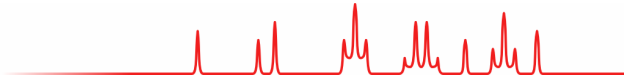
```
// <GUI menuname="Hello" shortcut="Ctrl+1" tooltip="Hello World" />
```

This line is used to instruct Mnova about the name to be displayed in the scripts menu along with the keyboard shortcut and the text used as a tooltip. For a full description on the usage of this line, see [connecting scripts](#) and [Mnova GUI](#) section below.

In any case, you can add any desired (or remove any undesired) folder as a 'Script Directory' by following the menu 'Scripts/Scripts Directories':



It is also possible to create dialog boxes with controls. The script below shows a simple example.



```

function myDialog()
{
    //Create the dialog
    var diag = new Dialog();
    diag.title="My Dialog";

    //Add some radio buttons
    var groupBox = new GroupBox();
    diag.add(groupBox);
    var rButtons = [];
    for ( var i = 0; i <3; i++)
    {
        rButtons[i] = new RadioButton();
        rButtons[i].text = "Radio " + i;
        if (i==0)
            rButtons[i].checked = true;
        groupBox.add(rButtons[i]);
    }
    //Add a check box
    var checkBox = new CheckBox();
    checkBox.text = "My Check Box";
    checkBox.checked = true;
    diag.add( checkBox );
    //Exec the dialog
    if( diag.exec() )
    {
        MessageBox.information("OK");
    }
    else
        MessageBox.information("Cancel");
}
    
```

Execution of this script results in the following dialog box:



Connecting scripts with Mnova objects

Simple access to an NMR spectrum

The examples we have shown so far are certainly not very useful. The good thing about scripting in Mnova is that it can be used to manipulate objects (e.g. spectra, molecules, pictures, etc) in



Mnova and automated repetitive tasks. This section will explain how to access an NMR spectra from within the scripting framework.

Consider the following code snippet:

```
function dumpSpectrum()
{
    //To get the active spectrum
    var spectrum = nmr.activeSpectrum();
    //The function isValid informs about if the spectrum obtained
    is correct
    print( spectrum.isValid() );
    //To print the spectrum information
    print( "Frequency: " + spectrum.frequency() );
    print( "Lowest Frequency: "+spectrum.hz() );
    print( "Real: "+spectrum.isReal );
    print( "Solvent: "+ spectrum.solvent );

    //To get information about each dimension
    var dCount = spectrum.dimCount;
    for( var i = 1; i <= dCount; i++ )
    {
        print( "Dimension "+i );
        print( "Nucleus: "+spectrum.nucleus(i) );
        //Spectral Size
        print( "Spectral Size: "+spectrum.count(i) );
    }
    //To print the value of one of the spectral points
    if( spectrum.isReal )
        print( spectrum.real(100));
    else
        print( spectrum.real(100)+" + "+spectrum.imag(100)+"j" );
}
```

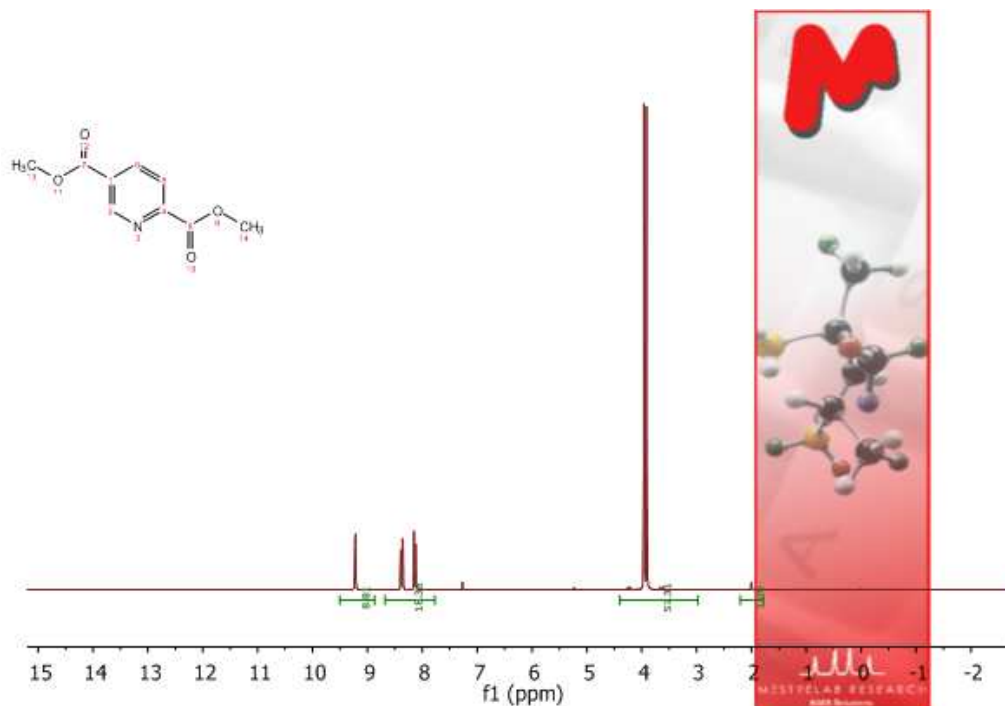
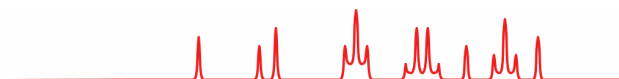
The first thing to note is the **nmr** object. This object gives access to the NMR plug-in framework and as such it allows you to get access to the NMR spectra in Mnova. For example, `nmr.activeSpectrum()` function makes it possible to select current active spectrum in Mnova. From that point, you can read relevant information of the NMR spectrum such as the number of points, spectral width, etc. For more information about the spectrum properties see the [reference](#).

Let's now move to a more comprehensive example which will illustrate, step by step, how to access all objects within an Mnova page.

Having access to multiple objects: the layout script

As an illustrative example, we will build a script which will make it possible to organize different objects contained within a page. This script will be called **layout** for obvious reasons.

To start off with, open a spectrum, a molecule and a picture (which could be, for example, your company logo). The following picture shows the initial conditions for this example:



The goal here will be to create a script which will arrange these 3 objects in such a way that the picture will be located at the left side of the page whilst the spectrum and the molecule will be moved next to the picture.

```
// <GUI menuname="Layout" shortcut="Ctrl+2" tooltip="Layout" />
function layout()
{
    var w = new DocumentWindow(Application.mainWindow.activeWindow());
    var p = new Page(w.curPage());
    var n = p.itemCount();

    for(i=0; i<n; i++)
    {
        var item = p.item(i);
        var width = item.width();
        var height = item.height();
        switch(item.name())
        {
            case "NMR Spectrum":
                item.left = p.left + 50;
                item.right = p.right - 10;
                item.top = p.top+20;
                item.bottom = p.bottom;
        }
    }
}
```



```
break;
case "Molecule":
    item.left = p.left + 50;
    item.right = item.left + width;
    item.top = p.top;
    item.bottom = p.top + height;
break;
case "Image":
    item.left = p.left;
    item.right = item.left + width;
    item.top = p.top;
    item.bottom = item.top + height;
break;
}
}
w.update();
}
```

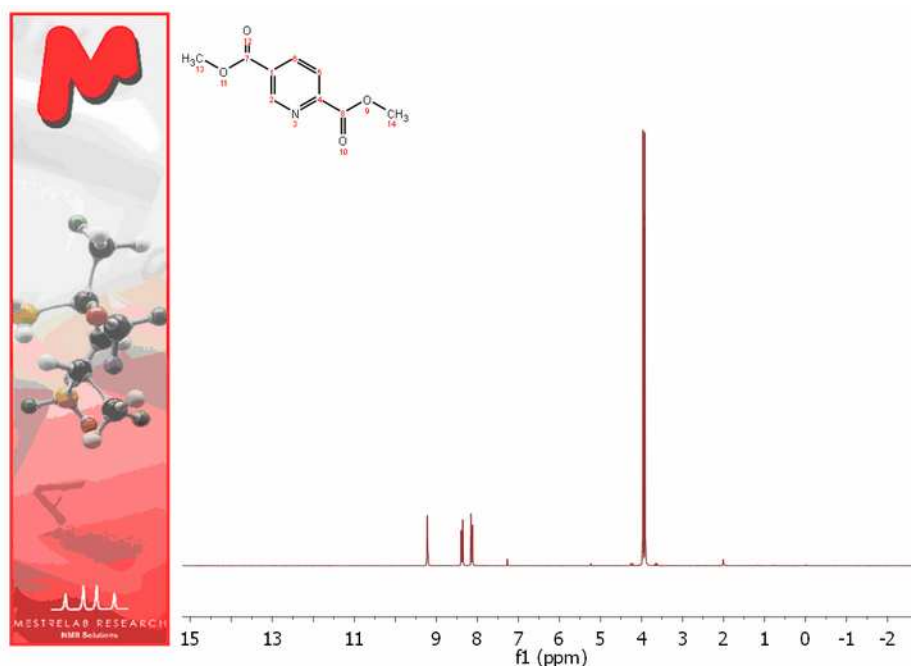
Every object in Mnova is contained in a page which in turn is contained in a document. So the first thing we should get access to is the document. This is done in this line:

```
var w = new DocumentWindow(Application.mainWindow.activeWindow());
```

Basically we access to the document window ([DocumentWindow](#)) via the **Application** object. Once we have the document, we can easily access to the currently active page through [Page](#) object like this:

```
var p = new Page(w.curPage());
```

Now we are ready to navigate through all the objects contained in this page. For every object, the exact type is checked within the switch command and then the objects are repositioned accordingly. This is the result for this example.



Tuning up the layout script

We will now modify the layout script in order to achieve the following two objectives:

1. Load the image from a file
2. Show the spectrum title automatically on the page when the script is executed

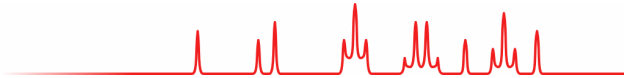
First off, we will show how to load an image from the disk. To do that, you have to use the [serialization](#) object. For example, assuming that the image is located in `c:\kk\logo.jpeg`, you can use this code:

```
serialization.open("c:\\kk\\logo.jpeg");
```

(Note the double back slash!).

Next thing is how to show the spectrum title on the page. In order to do that, it will be necessary to create a specialized NMR spectrum object rather than using the default properties available in every item object. See the following code snippet:

```
case "NMR Spectrum":
    var nmrSpc = new NMRspectrum(item);
    nmrSpc.left = p.left + 50;
    nmrSpc.right = p.right - 10;
    nmrSpc.top = p.top+20;
    nmrSpc.bottom = p.bottom;
    var tit = draw.text("<h3>"+nmrSpc.title+"</h3>", true);
    tit.left = p.left+50;
    tit.top = p.top;
break;
```



The first thing to note is the [NMR Spectrum](#) object. It will allow us to construct a specialized NMR spectrum object if a valid item is provided. From that point on, we will be able to change any spectrum properties or gain access to any of its elements such as the spectrum title. In this particular example we read the title using [nmrSpec.title](#) property which is then plotted with the `draw.text` command and moved to the top of the page next to the logo image.

It's important to note that in line

```
tit = draw.text( "<h3>" + nmrSpec.title + "</h3>", true );
```

we are creating a new object (or item). This means that the array of items will not be the same now and the internal for loop will not longer be valid. One could think of just incrementing the number of items by one unit, but it's not guaranteed that the internal order of the objects is the same after inserting a new one. The simplest solution is to make a temporary copy of the original objects in the page (before creating the title object) as this:

```
var c = p.itemCount();
var i = 0;
var itemArray = new Array(c);
```

And then we can iterate through all these items:

```
for( i = 0; i < itemArray.length; i++ )
{
    var item = itemArray[i];
    // ...
}
```

The complete new script looks like this:

```
// <GUI menuname="Layout2" shortcut="Ctrl+2" tooltip="Layout2" />
function layout2()
{
    var w = new
    DocumentWindow(Application.mainWindow.activeWindow());
    var p = new Page(w.curPage());
    serialization.open("c:\\kk\\logo.jpeg");
    var c = p.itemCount();
    var i = 0;
    var itemArray = new Array(c);
    for(i=0; i<c; i++)
    {
        itemArray[i] = p.item(i);
    }
    for( i = 0; i < itemArray.length; i++ )
    {
        var item = itemArray[i];
        var width = item.width();
        var height = item.height();

        switch(item.name())
        {
            case "NMR Spectrum":
```



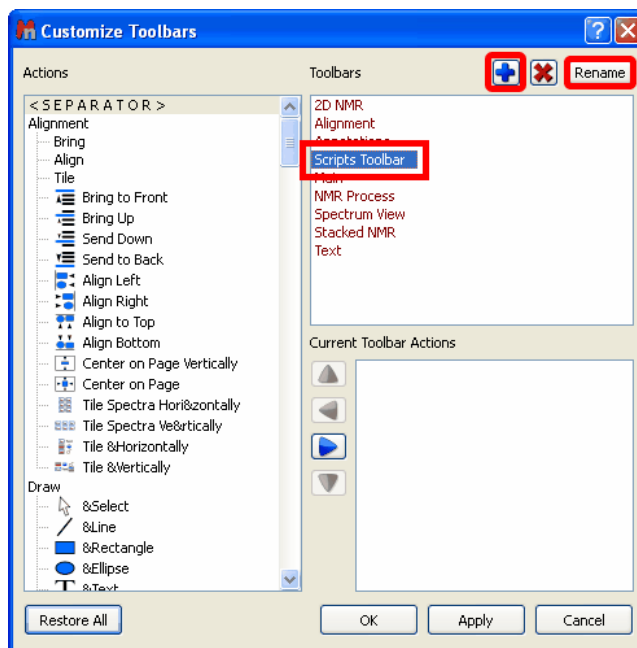
```

var nmrSpC = new NMRspectrum(item);
nmrSpC.left = p.left + 80;
nmrSpC.right = p.right - 10;
nmrSpC.top = p.top+20;
nmrSpC.bottom = p.bottom;
var tit = tit;
draw.text("<h3>" + nmrSpC.title + "</h3>", true);
tit.left = p.left+50;
tit.top = p.top;
print(item.name());

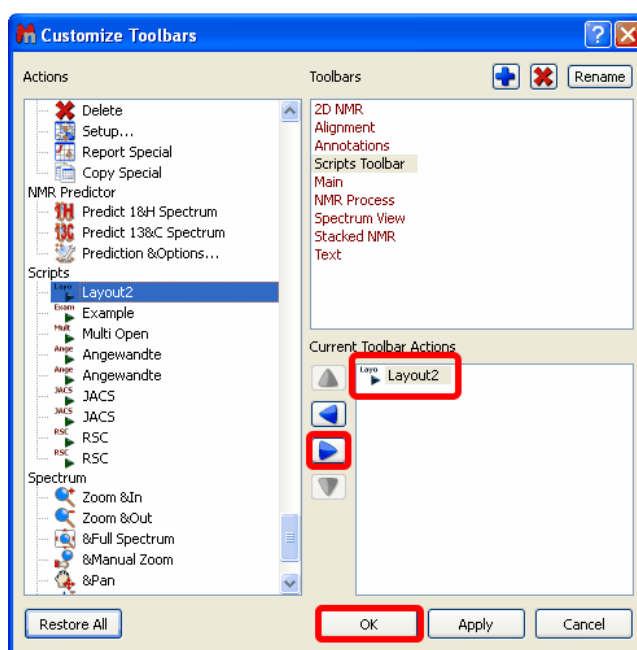
break;
case "Molecule":
    item.left = p.left + 90;
    item.right = item.left + width;
    item.top = p.top+50;
    item.bottom = item.top + height;
    print(item.name());
break;
case "Image":
    item.left = p.left;
    item.right = item.left + width;
    item.top = p.top;
    item.bottom = item.top + height;
    print(item.name());
break;
}
}
w.update();
}

```

Once you have the script saved (please bear in mind that the script file will need to have the same name as the function which you want to call, in this case the name of the script must be 'layout2.qs') in the Mnova scripts folder (and then have restarted the program), you will be able to add it to the toolbar just by following the menu: 'View/Toolbars/Customize Toolbars'. This will display the 'Customize Toolbars' dialog box. Then, click on the '+' button to create a new toolbar and type its name (Scripts Toolbar) by clicking on the 'Rename' button.



Next, navigate to the 'Script' icon (Layout2) in the 'Actions' menu (at the left-hand side of the window) and Click on the 'Right Arrow' button to add the 'Layout2' script to the 'New Toolbar'.



Finally, click on OK and you will be able to see the 'Layout2' icon in the 'New toolbar'. You will also be able to introduce the script in any existing contextual menu by following a similar procedure in the 'Customize Context menus' dialog box.



Custom NMR Processing

There are two ways to process NMR spectra via scripting:

1. By means of Processing Templates
2. By using the [Processing](#) interface

1 Processing templates

It is possible within Mnova to process a 1D or 2D NMR spectrum via scripting by using the processing templates (generated with the Full Processing feature). As you can see in the below example, you can process (with an easy script) any kind of spectrum (1H, 13C, 1D or 2D). You only need to create the processing template for each experiment. In the example below we have created 3 different processing templates, one for the processing of 1H-NMR spectra (1H.mnp), the second for the processing of 13C-NMR spectra (13C.mnp) and the last one for the processing of COSY spectra (COSY.mnp). As you can see in the scripts, the three the templates were saved at c:/kk/ (of course you can change this path and also the name of the template files).

```
function procTemplate()
{
    var spec = new NMRspectrum( nmr.activeSpectrum() );
    var processing = new String;
    if (!spec.isValid())
        return;
    if (spec.dimCount == 1)
    {
        if (spec.nucleus() == "1H")
            processing = "c:/kk/1H.mnp";
        else if (spec.nucleus() == "13C")
            processing = "c:/kk/13C.mnp";
    }
    else if (spec.dimCount == 2)
    {
        if (spec.nucleus(1) == "1H" && spec.nucleus(2) == "1H")
            processing = "c:/kk/COSY.mnp";
    }
    nmr.processSpectrum(spec, processing);
}
```

2 Processing interface

The [Processing](#) interface makes it possible to change or apply any processing operation available in Mnova.

For example, let's make a simple script which will set an exponential weighting function:

```
function myProcess()
{
```



```
var spec = new NMRspectrum( nmr.activeSpectrum() );
var p = new NMRProcessing(spec.proc);

p.setParameter( "Apodization[1].Apply", true );
p.setParameter( "Apodization[1].Exp.Apply", true );
p.setParameter( "Apodization[2].Exp.Value", 0.5 );
print( p.getParameter( "Apodization[1].Exp" ) );

spec.proc = p;
spec.process();
mainWindow.activeWindow().update();
}
```

After creating a spectrum object which represents current active spectrum, we create a processing variable (p) which represents the processing operations for that particular spectrum. Once this new object is created, we can read or change its values using `getParameter` and `setParameter` functions. For instance, in this example we first set to true the Apodization property (otherwise no apodization will be applied) and then, after setting to true the exponential function, we set the line broadening to 0.5 Hz.

Finally, when we are done, we assign this processing object to the spectrum object (`spec.proc = p;`) and we call `spec.process();` to actually apply all the processing operations.

Connecting scripts with Mnova GUI

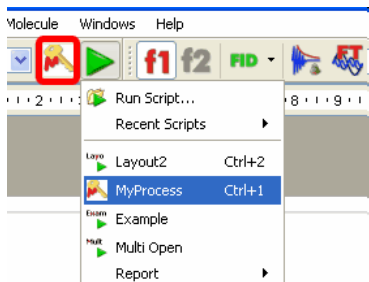
As you have seen in the examples above, it is possible to insert any script in the Mnova Scripts menu, contextual menus or toolbars.

The key to get this working lies in the first line of the script (and of course, in the fact that Mnova can locate the script as it has been saved in the appropriate folder):

```
// <GUI menuname= "MyProcess" shortcut="Ctrl+1" tooltip="Exponential
0.5" icon="C:/bin2.jpg" />
```

This line is used to instruct Mnova about the name to be displayed in the scripts menu along with the keyboard shortcut and the text used as a tooltip. Please bear in mind that the user is also able to use any created image file as icon. In this case, the name of the script is *MyProcess*, the shortcut is *Ctrl+1*, the tooltip is *Exponential 0.5* and the icon used was saved at *C:/bin2.jpg*.

You will be able to see in the picture below, that we have added the script 'MyProcess' to the 'Scripts' menu with the corresponding shortcut (Ctrl+1) and that we have introduced our own icon on the toolbar (red square in the picture below).



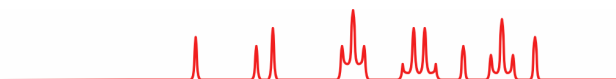
Please bear in mind that the script file will need to have the same name that the function which you want to call, in this case the name of the script must be 'MyProcess.qs', because the name of the function is: myProcess().

Mnova script reference

This reference contains a list of objects, functions and properties supported by Mnova script. For a complete ECMAScript Reference, consult this link: <http://doc.trolltech.com/4.3/ecmascript.html>

Example scripts are present in the examples/scripts installation directory

- **MainWindow:** Here it is a list of the objects related to the Mnova spectral window:
 - **MainWindow** (MainWindow aMainWnd)
 - MainWindow aMainWnd
 - function **newWindow**(): This function creates a new document window. Because no other statements in the document require the reference to the new window just opened, the statement does not assign its returned value to any variable. This is an acceptable practice in JavaScript if you don't need the returned value of a function or method.
 - function **activeWindow**(): Returns a DocumentWindow object that represents the active document window
 - function **toString**(): This function returns string values (it 'prints' the information on the screen). Every JavaScript core language object and every DOM document object has a toString() method associated with it. This method is designed to render the contents of the object in as meaningful a way as possible.
 - function **newDocument**(): returns a new document.
- **DocumentWindow:** Represents a document object. The DocumentWindow object is a member of the DocumentWindows collection. The DocumentWindows collection contains all the open document windows.
 - **DocumentWindow** (DocumentWindow aDocWin)
 - DocumentWindow aDocWin

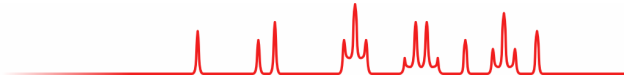


- function **close**(): Closes the current document
 - function **pageCount**(): You can use this function to determine the number of pages of the document
 - function **page**(Number aIndex): return Page
 - Number aIndex
Description: Returns the page number aIndex
 - function **curPage**(): This function returns the current page
 - function **update**(): This function refreshes the current page
 - function **toString**(): It returns a String which represent the current object
- **Page:** You will find below a list of the Page objects:
 - Number **left**
Description: Left margin of the page (mm)
 - Number **right**
Description: Right margin of the page (mm)
 - Number **top**
Description: Top margin of the page (mm)
 - Number **bottom**
Description: Bottom margin of the page (mm)
 - Number **width**
Description: Width of the page
 - Number **height**
Description: Height of the page
 - **Page** (Page aPage)
 - Page aPage
 - function **itemCount**(String aType): return Number
 - String aType: Element Type
Valid Values: { "Arrow" "Ellipse" "Rectangle" "Polygon" "Image" "Text" "NMR Spectrum" "Molecule" "OLE Object" "Mass Spectrum" }
 Description: This function returns the number of objects in the current page
 - function **itemCount**(): retrieves the number of elements currently displayed in the page.
 - function **item**(Number aIndex, String aType): return
 - Number aIndex: Number of elements
 - String aType: Element Type
 Description: Returns the item of the page
 - function **item**(Number aIndex): return



- Number aIndex: Number of the element
 - Description: Returns an item of the page
 - function **addItem**(PageItem aPageItem): return
 - PageItem aPageItem: Item to be added
 - Description: this function adds an item to the page
 - function **update**(): return
 - Description: The update function will be used to refresh the page
 - function **toString**(): return String

- **PageItem**: Below you will find the list of the objects related with the item of a page:
 - String **name**
Description: Type of the element (arrow, ellipse, rectangle, polygon, image, text, NMR spectrum, molecule, OLE Object or Mass spectrum")
 - String **uuid**
Description: The UUID structure defines Universally Unique Identifiers (UUIDs). UUIDs provide unique designations of objects such as interfaces, manager entry-point vectors, and client objects. A string UUID contains the character-array representation of a UUID. A string UUID is composed of multiple fields of hexadecimal characters. Each member has a fixed length, and fields are separated by the hyphen character.
 - Number **left**
Description: Returns the left margin of the item
 - Number **right**
Description: Returns the right margin of the item
 - Number **top**
Description: Returns the top margin of the item
 - Number **bottom**
Description: Returns the bottom margin of the item
 - Number **width**
Description: Width of the element
 - Number **height**
Description: Height of the item
 - Number **angle**
Description: Width of the item
 - **PageItem** (PageItem aPageItem)
 - PageItem aPageItem
 - function **toString**(): return String
 - function **setProperty**(String aParamName, Variant aValue): return
 - String aParamName:



- Variant aValue:

Description: The SetProperty sets the specified parameter with the corresponding value. In order to see the list of the aParamName, please click [here](#)

- function **getProperty**(String aParamName, String aUnit): return Variant

- String aParamName:

- String

aUnit:

Valid Values: { "mm" "cm" "inches" }

Description: Retrieves parameter in the selected unit (mm, cm, inches). In order to see the list of the aParamName, please click [here](#)

- function **update**(): return

Description: Redraw the item

- **Serialization:** is the process of converting an object into a form that can be readily transported. For example, open and save documents.

- **Serialization** (Serialization aSerialization)

- Serialization aSerialization

- function **open**(String aFileName): return Boolean

- String aFileName: File Name

Description: this function is used to open a file

- function **open**(Array aFileNameList): return

- Array aFileNameList

Description: Opens an array of files

- function **save**(String aFileName, String aFormat): return

- String aFileName

- String aFormatValid Values: { "mnova" "mnlit" "mncs" "pdf" "ps" "eps" "bmp" "jpg" "mrc" "txt" "jcamp" "png" "ppm" "svg" "tiff" "xpm" "xbm" }

Description: Saves a document (different formats available).

- function **toString**(): return String

- **Settings:** You will find below a list of the objects related with saving and reading settings:

- **Settings** (Settings aSettings)

- Settings aSettings

- function **value**(String aKey, Variant aDefaultValue): return Variant

- String aKey: name of the key

- Variant aDefaultValue: Default value if the key doesn't exist or can't be read



Description: Read the value of a settings key

- function **setValue**(String aKey, Variant aValue): return
 - String aKey: a Settings key
 - Variant aValue: value to be saved
- function **toString**(): return

- **DrawPlugin**: It is the object to add text objects to the document

- **DrawPlugin** (DrawPlugin aDrawPlugin)
 - DrawPlugin aDrawPlugin
- function **text**(String aText, Boolean alsHtml): return PageItem
 - String aText: Text
 - Boolean alsHtml: HTML text

Description: This function will add a text box to the document in order to insert the text.

- function **toString**(): return

- **NMRPlugin**: The NMRPlugin objects allow us to process our NMR spectra. The objects of type NMRPlugin have the following properties:

- **NMRPlugin** (NMRPlugin aNMRPlugin)
 - NMRPlugin aNMRPlugin
- function **process**(String aProcFile): return
 - String aProcFile: File Name

Description: this function reads a processing file and applies it to the active spectrum

- function **processSpectrum**(NMRspectrum aSpectrum, String aProcFile): return
 - NMRspectrum aSpectrum:
 - String aProcFile:

Description: Reads a processing file and applies it to a spectrum

- function **beginModification**(NMRspectrum aSpectrum): return
 - NMRspectrum aSpectrum:

Description: this function saves the current state of a spectrum

- function **endModification**(NMRspectrum aSpectrum): return
 - NMRspectrum aSpectrum:

Description: Used with beginModification creates a User Processing Command that allows Do/Undo

- function **multipletTable**(): return MultipletTable



Description: Returns the Multiplet Table of the active spectrum

- o function **peakTable**(): return PeakTable

Description: Returns the current Peak Table of the active spectrum

- o function **activeSpectrum**(): return NMRspectrum

Description: Returns the active spectrum

- o function **toString**(): return String

- **MultipletTable**: It allows the user to get the multiplet table.

- o **MultipletTable** (MultipletTable aMultipletTable)

- MultipletTable aMultipletTable

- o function **spectrum**(): return

Description: spectrum to which the table makes reference

- function **toString**(): return String

- **Multiplets**: The multiplets objects allow us to get the multiplet report. The objects of type Multiplets have the following properties:

- o Number

Description: Number of multiplets

count

- o **Multiplets** (Multiplets aMultipletList)

- Multiplets aMultipletList:

- o function **at**(Number aIndex): return Multiplet

- Number aIndex

Description: Returns multiplet at the position aIndex

- o function **sort**(Boolean bAscending): return

- Boolean bAscending:

Description: Sort Ascending/Descending

- o function **clear**(): return

Description: delete all the multiplets

- o function **removeAt**(Number aIndex): return

- Number aIndex:

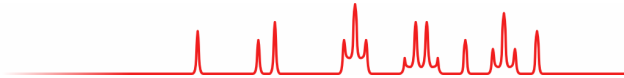
Description: delete the multiplet at aIndex chemical shift

- o **append** (Multiplet aMultiplet): return

- Multiplets aMultiplet:

Description: to add the multiplet to the list

- o function **toString**(): return String



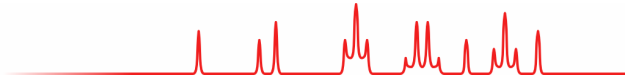
- **Multiplet**: The multiplet objects allow us to get the multiplet report. The objects of type Multiplet have the following properties:
 - String **category**
Description: Category; multiplet (m), singlet (s), doublet (d) , triplet (t), etc.)
 - Number **delta**
Description: chemical shift
 - String **name**
Description: name of the multiplet
 - Number **rangeMin**
Description: minimum value of the range of the multiplet
 - Number **rangeMax**
Description: maximum value of the range of the multiplet
 - Number **nH**
Description: number of Hydrogens
 - **Multiplet** (Multiplet aMultiplet)
 - Multiplet aMultiplet: to create a multiplet
 - **Multiplet** (SpectrumRegion aSpectrumRegion, String aCategory)
 - SpectrumRegion aSpectrumRegion:
 - String aCategory:
 Description: to create a multiplet indicating the region of the spectrum and the corresponding category (s, d, t, etc.)
 - **Multiplet** (SpectrumRegion aSpectrumRegion, NMRspectrum aSpectrum)
 - SpectrumRegion aSpectrumRegion:
 - NMRspectrum aSpectrum:
 Description: to report the result of an automatic multiplet analysis from a region of a spectrum
 - function **jList**(Boolean aReduced): return
 - Boolean aReduced:
 Description: returns the coupling constants
 - function **setjList**(JList aJList): return
 - JList aJList:
 Description: to assign a list of coupling constants to a multiplet
 - function **toString**(): return
- **JList**. The JList objects allow us to get the information about the coupling constants values in the multiplet report. The objects of type **JList** have the following properties:
 - Number **count**
Description: number of coupling constants values



- Number **length**
Description: number of coupling constants values
 - **JList** (JList aJList)
 - JList aJList:
 - function **at**(): return Number
Description: it returns the chemical shift value
 - function **sort**(Boolean bAscending): return
 - Boolean bAscending:
 Description: it sorts the elements in ascending/descending order
 - function **clear**(): return
Description: deletes all the coupling constants
 - function **removeAt**(Number aIndex): return
 - Number aIndex:
 Description: it removes the element
 - function **append**(Number aJ): return
 - Number Number aJ:
 Description: to add the J to the list
 - function **toString**(): return String
- **NMRSpectrum**: The NMRSpectrum objects allow us to get information about our spectra. The objects of type NMRSpectrum have the following properties:
 - Number **dimCount**
Description: it counts the number of dimensions (one or two)
 - Number **threshold**
Description: Threshold
 - Boolean **isReal**
Description: informs if the spectrum is correct
 - NMRProcessing **proc**
Description: processing of the NMR spectrum
 - String **title**
Description: it returns the title of the spectrum
 - String **solvent**
Description: it returns the solvent of the experiment
 - Number **temperature**
Description: it shows the temperature value of the experiment
 - String **originalFormat**
Description: it returns the original format
 - Number **experimentType**
Description: it shows the type of the experiment



- **NMRSpectrum** (NMRSpectrum aSpectrum)
 - NMRSpectrum aSpectrum:
- function **isValid**(): return Boolean
Description: The function isValid informs about if the spectrum is correct or not.
- function **isValid**(Number aDim): return Boolean
 - Number aDim:
 Description: isValid returns true if aDim is less or the same as the number of the dimensions of the spectrum (dimCount)
- function **isFID**(Number aDim): return Boolean
 - Number aDim:
 Description: isFID returns true if the dimension aDim is a FID
- function **nucleus**(Boolean aHtml): return String
 - Boolean aHtml:
- function **nucleus**(Number aDim, Boolean aHtml): return String
 - Number aDim:
 - Boolean aHtml:
 Description: Returns the nucleus of the dimension, if aHtml is true uses html
- function **count**(Number aDim): return Number
 - Number aDim:
 Description: Number of points in the dimension aDim
- function **count**(): return Number
Description: Number of points in the last dimension
- function **real**(Number aIndex): return Number
 - Number aIndex:
 Description: Returns the real value of the aIndex point in the spectrum data
- function **setReal**(Number aIndex, Number aValue): return Boolean
 - Number aIndex:
 - Number aValue:
 Description: Sets the real value of the aIndex point in the spectrum data
- function **imag**(Number aIndex): return Number
 - Number aIndex:
 Description: Returns the imag value of the aIndex point in the spectrum data
- function **setImag**(Number aIndex, Number aValue): return Boolean
 - Number aIndex:
 - Number aValue:



Description: Sets the imag value of the aIndex point in the spectrum data

- function **fidReal**(Number aIndex): return Number
 - Number aIndex:

Description: Returns the real value of the aIndex point in the spectrum fid

- function **fidsetReal**(Number aIndex, Number aValue): return Boolean
 - Number aIndex:
 - Number aValue:

Description: Sets the real value of the aIndex point in the spectrum fid

- function **fidImage**(Number aIndex): return Number
 - Number aIndex:

Description: Returns the imag value of the aIndex point in the spectrum fid

- function **fidsetImage**(Number aIndex, Number aValue): return Boolean
 - Number aIndex:
 - Number aValue:

Description: Sets the imag value of the aIndex point in the spectrum fid

- function **multiplets**(): return Multiplets

Description: Returns the list of the multiplets of the spectrum

- function **setMultiplets**(Multiplets aMultipletList): return
 - Multiplets aMultipletList:

Description: sets the list of the multiplets of the spectrum

- function **peaks**(): return Peaks

Description: Returns the list of the peaks of the spectrum

- function **setPeaks**(Peaks aPeakList): return
 - Peaks aPeakList:

Description: sets the list of the peaks of the spectrum

- function **frequency**(): return

Description: returns the frequency of the spectrum

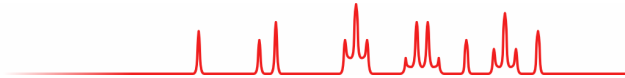
- function **frequency**(Number aDim): return
 - Number aDim:

Description: returns the frequency of the dimension aDim of the spectrum

- function **width**(Number aDim): return Number
 - Number aDim:

Description: returns the spectral width

- function **width**(): return Number



Description: returns the spectral width

- function **hz**(): return Number

Description: returns the frequency of the spectrometer

- function **hz**(Number aDim): return Number
 - Number aDim:

Description: returns the frequency of the spectrometer

- function **process**(NMRProcessing aProc): return Boolean
 - NMRProcessing aProc:

Description: Process the spectrum using aProc. Without parameter re-process the spectrum

- function **corrPearson**(NMRspectrum aSpectrum): return Number
 - NMRspectrum aSpectrum:

Description: this function allows the user to calculate the correlation coefficient between two spectra. That is to say, how much does one spectrum look like the other?

- function **getParam**(String aParam): return Variant
 - String aParam:

Description: to get a parameter

- function **setParam**(String aParam, Variant aValue): return
 - String aParam:
 - Variant aValue:

Description: Set parameter. Only for custom parameters

- function **isReadOnlyParam**(String aParam): return Boolean
 - String aParam:

Description: to get a parameter

- function **zoom**(Number aHFrom, Number aHTo, Number aVFrom, Number aVTo)
 - Number aHFrom
 - Number aHTo
 - Number aVFrom
 - Number aVTo

Description: Manual Zoom. Without parameters makes a full zoom

- function **horZoom**(Number aFrom, Number aTo)



- **Number** aFrom
 - **Number** aTo

Description: Horizontal Zoom. Without parameters makes a full zoom

 - function **vertZoom**(**Number** aFrom, **Number** aTo)
 - **Number** aFrom
 - **Number** aTo

Description: Vertical Zoom. Without parameters makes a full zoom

 - function **toString**(): return String
- **PeakTable**: The PeakTable objects allow us to get the table peaks.
 - **PeakTable** (PeakTable aPeakTable)
 - PeakTable aPeakTable:
 - function **spectrum**(): return

Description: returns the spectrum to which the peak table makes reference

 - function **toString**(): return
- **Peak**: The Peaks objects allow us to get information about the chemical shift of the signals. The objects of type Peak have the following properties:
 - Number **intensity**

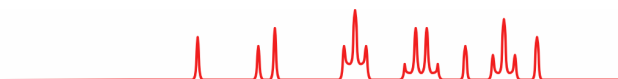
Description: returns the intensity of the peak
 - Number **type**

Description: returns the type of the peak. Valid Values: { "Peak.Compound" "Peak.Artifact" "Peak.Impurity" "Peak.Solvent" "Peak.CS_Ref" "Peak.Quantification_Ref" }
 - Number **kind**

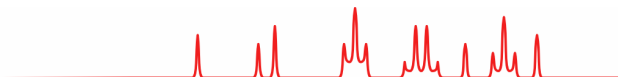
Description: returns the kind of the peak. Valid Values: { "Peak.Maximum" "Peak.Minimum" }
 - **Peak** (Peak aPeak)
 - Peak aPeak:

Description: Copy the peak
 - **Peak** (Number aPos, Number aIntensity)
 - Number aPos:
 - Number aIntensity:

Description: to create a peak with a determined chemical shift (Number aPos) and intensity (Number aIntensity)
 - **Peak** (Number aPos, NMRspectrum aSpectrum)
 - Number aPos:



- Number NMRspectrum aSpectrum:
 - Description: too add a peak at a chemical shift 'aPos' in a 1D-NMR spectrum.
The intensity of the peak is refilled with the spectral data
 - **Peak** (Number aPos1, Number aPos2, Number aIntensity)
 - Number aPos1:
 - Number aPos2:
 - Number aIntensity:
 - Description: to create a peak in a 2D-NMR spectrum with a determined chemical shift (Number aPos) and intensity (Number aIntensity)
 - **Peak** (Number aPos1, Number aPos2, NMRspectrum aSpectrum)
 - Number aPos1:
 - Number aPos2:
 - NMRspectrum aSpectrum:
 - Description: too add a peak at a chemical shift 'aPos' in a 2D-NMR spectrum.
The intensity of the peak is refilled with the spectral data
 - function **delta**(Number aDim): return
 - Number aDim:
 - Description: returns the chemical shift value of the peak
 - function **delta**(): return
 - Description: returns the chemical shift value of the peak
 - function **toString**(): return
- **Peaks:** The Peaks objects allow us to get information about the Peak Picking.
The objects of type Peak have the following properties:
 - Number **count**
Description:
 - **Peaks** ()
Description: Empty peak list
 - **Peaks** (Peaks aPeakList)
 - Peaks aPeakList:
 - Description:
 - **Peaks** (NMRspectrum aSpectrum, SpectrumRegion aRegion)
 - NMRspectrum aSpectrum:
 - SpectrumRegion aRegion:
 - Description:
 - function **at**(Number aDim): return Peak



- Number aDim:

Description: Returns the peaks at the position aDim

 - function **sort**(Boolean bAscending): return
 - Boolean bAscending:

Description: it sorts the peaks in ascending/descending order
 - function **sort**(Boolean bHAscending, Boolean bVAscending): return
 - Boolean bHAscending:
 - Boolean bVAscending:

Description: it sorts the peaks in ascending/descending order. In 2D, the user can select a different sort in each dimension.
 - function **clear**(Number aIndex): return
 - Number aIndex:

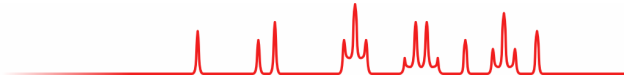
Description: clear the peaks
 - function **removeAt**(Number aIndex): return
 - Number aIndex:

Description: remove the peaks at a determined chemical shift
 - function **append**(Peak aPeak): return
 - Peak aPeak:

Description: to add the peak to the list
 - function **toString**(): return
- **Integrals:** The Integral objects allow us to get information about the integration analysis. The objects of type Integrals have the following properties:
 - Number **count**
Description: number of integrals
 - **Integrals** ()
Description: Constructs an empty integral list
 - **Integrals** (Integrals aIntegralList)
 - Integrals aIntegralList:

Description: constructs the integral list
 - function **at**(Number aIndex): return Integral
 - Number aIndex:

Description: Returns the integral at the position aIndex
 - function **clear**(): return



Description: to clear the integrals

- function **removeAt**(Number aIndex): return
 - Number aIndex:

Description: to remove the integral at a determined chemical shift

- function **append**(Integral aIntegral): return
 - Integral aIntegral:

Description: to add the integral to the list

- function **toString**(): return

- **Integral:**

- Number **IntegralValue**

Description: returns the value of the integral

- Number **rangeMin**

Description: the minimum value of the integral range

- Number **rangeMax**

Description: the maximum value of the integral range

- function **Integral**(Integral aIntegral):
 - Integral aIntegral:

Description: to create an integral

- function **Integral**(SpectrumRegion aSpectrumRegion, NMRspectrum aSpectrum):
 - SpectrumRegion aSpectrumRegion:
 - NMRspectrum aSpectrum

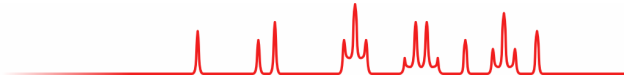
Description: to report the result of an automatic integral analysis from a region of a spectrum

- function **toString**(): return

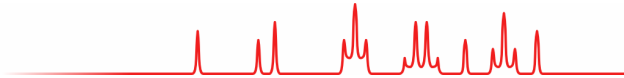
- **Dir:** The Dir object provides access to directory structures and their contents. All the scripts are present in the examples/scripts installation directory.

Example scripts are present in the examples/scripts installation directory

- property **name: String**
Returns the name of the directory
- property **absPath: String**
Returns the absolute path



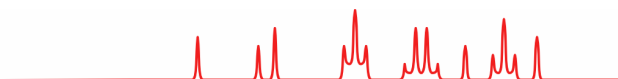
- property **exists**: **Boolean**
Returns true if the directory exists
- Constructor **Dir(String aDirPath)**
 - **String aDirPath**Description: Constructs a Dir pointing to the given directory path
- function **entryList(String aWildcard, Number aKind, Number aSortFlags): String**
 - **String aWildcard**: Filter that understands * and ? wildcards
 - **Number aKind**: Filter option. Combination of the following flags
Valid Values: {Dir.Dirs, Dir.Files, Dir.Drives, Dir.NoSymLinks, Dir.All, Dir.TypeMask, Dir.Readable, Dir.Writable, Dir.Executable, Dir.RWEMask, Dir.Modified, Dir.Hidden, Dir.System, Dir.AccessMask}
 - **Number aSortFlags**: Sort option. Combination of the following flags
Valid Values: {Dir.Name, Dir.Type, Dir.Size, Dir.Unsorted, Dir.SortByMask, Dir.DirsFirst, Dir.Reversed, Dir.IgnoreCase}Returns a list of the names of all the files and directories in the directory, ordered according to the name and attribute filters
- function **filePath(String aFileName): String**
 - **String aFileName**Description: Returns the path name of a file in the directory. Does not check if the file actually exists in the directory
- function **cdUp(): Boolean**
Changes directory by moving one directory up from the Dir's current directory
- function **cd(String aDirName):: Boolean**
 - **String aDirName**Changes the Dir's directory to dirName
- function **mkdir(String aDirName):: Boolean**
 - **String aDirName**Creates a sub-directory specified by aDirName
- function **mkpath(String aDirPath):: Boolean**
 - **String aDirPath**Creates the directory path aDirPath. The function will create all parent directories necessary to create the directory
- function **rmdir(String aDirName):: Boolean**
 - **String aDirName**Removes the directory specified by aDirName
- function **rmpath(String aDirPath):: Boolean**



- **String aDirPath**
Removes the directory path aDirPath. The function will remove all parent directories in aDirPath, provided that they are empty
 - function **Dir.home():String**
Returns the absolute path of the user's home directory
 - function **Dir.root():String**
Returns the absolute path of the root directory
 - function **Dir.temp():String**
Returns the absolute path of the system's temporary directory
 - function **Dir.application():String**
Returns the absolute path of the application executable
 - function **Dir.current():String**
Returns the absolute path of the application's current directory
 - function **Dir.setCurrent(String aDirPath): Boolean**
 - **String aDirPath**
Sets the application's current working directory to aDirPath
 - function **Dir.drives():Array**
Returns a list of the root directories on this system
 - function **Dir.cleanDirPath(String aDirPath):String**
 - **String aDirPath**
Removes all multiple directory separators "/" and resolves any "."s or ".."s found in the path, aDirPath
 - function **Dir.convertSeparators(String aDirPath):String**
 - **String aDirPath**
Returns aDirPath with the '/' separators converted to separators that are appropriate for the underlying operating system
 - function **fileExists(String aFileName):Boolean**
 - **String aFileName**
Returns true if the file called aFileName exists
 - function **toString(): String**
- **File:**
Example scripts are present in the examples/scripts installation directory
 - property **name: String**
Returns the file name
 - property **exists: Boolean**
Returns true if the directory exists



- property **size: Number**
Returns the file size
- Constructor **File(String aFileName)**
 - **String aFileName**Description: Constructs a new file object to represent the file with the given aFileName
- Constructor **File(File aFile)**
 - **File aFile**Description: Constructs a copy of the aFile
- function **open(Number aMode): bool**
 - **Number aMode**: This enum is used to describe the mode in which a file is opened
Valid Values: {File.ReadOnly, File.WriteOnly, File.ReadWrite, File.NotOpen}Opens the file using aMode mode, returning true if successful
- function **close(): Boolean**
Closes the file and sets its open mode to File. NotOpen
- function **remove(): Boolean**
Removes the file. Returns true if successful. The file has to be closed before it is removed
- function **File.remove(String aFileName): Boolean**
 - **String aFileName**: Removes the file specified by the given aFileName. Returns true if successful
- function **File.create(String aFileName): Boolean**
 - **String aFileName**: Creates the file specified by the given aFileName. Returns true if successful
- function **File.exists(String aFileName): Boolean**
 - **String aFileName**: Returns true if the aFileName exists
- function **File.rename(String aOldFileName, String aNewFileName): Boolean**
 - **String aOldFileName**
 - **String aNewFileName**Renames the file aOldFileName to aNewFileName. Returns true if successful
- function **File.copy(String aFileName, String aNewFileName): Boolean**
 - **String aFileName**
 - **String aNewFileName**Copies the file aFileName to aNewFileName. Returns true if successful
- function **toString(): String**



- ***TextStream***: The TextStream object provides an interface for reading and writing text to a file. Example scripts are present in the examples/scripts installation directory.

- property ***pos***: **Number**

Gets/sets the file position corresponding to the current position of the stream.

Returns -1 if an error occurs

- property ***precision***: **Number**

Gets/sets the current real number precision, or the number of fraction digits

TextStream will write when generating real numbers

- property ***notation***: **Number**

Gets/sets the real number notation

Valid Values: {TextStream.nScientific, TextStream.nFixed, TextStream.nSmart}

- Constructor ***TextStream***(File aFile)

- **File** aFile

Description: Constructs a TextStream that operates on aFile

- Constructor ***TextStream***(TextStream aStream)

- **TextStream** aStream

Description: Constructs a TextStream that is a copy of aStream

- function ***atEnd***(): **Boolean**

Returns true if there is no more data to be read from the TextStream

- function ***write***(Variant aParam1, ..., Variant aParamN): **Boolean**

- **Variant** aParam1

▪ ...

- **Variant** aParamN

Writes the arguments to the stream

- function ***readLine***(): **String**

Reads one line of text from the stream and returns it

- function ***readString***(): **String**

Reads a word from the stream and returns it. Words are separated by whitespace

- function ***readNumber***(): **Number**

Reads a number from the stream and returns it

- function ***read***(Number aMaxLen): **String**

- **Number** aMaxLen

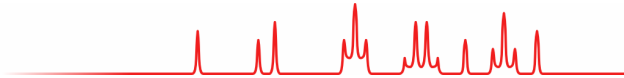
Reads at most aMaxLen characters from the stream, and returns the data read as a String

- function ***readAll***(): **String**

Reads the entire content of the stream, and returns it as a String

- function ***skipWhiteSpace***()

Reads and discards whitespace from the stream until either a non-space character is detected, or until atEnd() returns true



- function **skip**(Number aSize)
 - **Number** aSize

Skips the aSize bytes in the stream

- function **toString**(): **String**

- **BinaryStream**: The BinaryStream object provides serialization of binary data to a file. Example scripts are present in the examples/scripts installation directory

- property **pos**: **Number**

Gets/sets the file position corresponding to the current position of the stream.

Returns -1 if an error occurs

- property **endianness**: **Number**

Gets/sets the serialization endianness

Valid Values: {BinaryStream.eBig, BinaryStream.eLittle}

- Constructor **BinaryStream**(File aFile)
 - **File** aFile

Description: Constructs a BinaryStream that operates on aFile

- Constructor **BinaryStream**(BinaryStream aStream)
 - **BinaryStream** aStream

Constructs a BinaryStream that is a copy of aStream

- function **atEnd**(): **Boolean**

Returns true if there is no more data to be read from the BinaryStream

- Constructor **skip**(Number aSize)
 - **Number** aSize

Description: Skips the aSize bytes in the stream

- function **writeln8**(Number aValue)
 - **Number** aValue

Description: Writes a signed byte to the stream

- function **writeln16**(Number aValue)
 - **Number** aValue

Description: Writes a signed 16-bit integer to the stream

- function **writeln32**(Number aValue)
 - **Number** aValue

Description: Writes a signed 32-bit integer to the stream

- function **writeln64**(Number aValue)
 - **Number** aValue

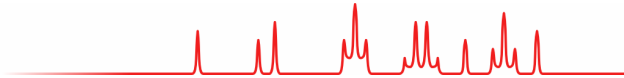
Description: Writes a signed 64-bit integer to the stream

- function **writeln8**(Number aValue)
 - **Number** aValue

Description: Writes a signed byte to the stream



- function **writeBool**(Boolean aValue)
 - Boolean aValueDescription:Writes a boolean to the stream
- function **writeReal32**(Number aValue)
 - Number aValueDescription:Writes a 32-bit real (float) to the stream
- function **writeReal64**(Number aValue)
 - Number aValueDescription:Writes a 64-bit real (float) to the stream
- function **writeString**(String aValue)
 - String aValueDescription:Writes a String to the stream
- function **writeCString**(String aValue)
 - String aValueDescription:Writes the '\0'-terminated (C-like) string to the stream
- function **writeBytes**(String aValue, Number aSize)
 - String aValue
 - Number aSizeDescription:Writes aSize bytes from aValue to the stream
- function **readInt8**()
Description:Reads a signed byte from the stream
- function **readInt16**()
Description:Reads a signed 16-bit integer from the stream
- function **readInt32**()
Description:Reads a signed 32-bit integer from the stream
- function **readInt64**()
Description:Reads a signed 64-bit integer from the stream
- function **readBool**()
Description:Reads a boolean from the stream
- function **readReal32**()
Description:Reads a 32-bit real (float) from the stream
- function **readReal64**()
Description:Reads a 64-bit real (float) from the stream
- function **readString**()
Description:Reads a String from the stream



- function **readCString()**
Description: Reads the '\0'-terminated (C-like) string from the stream
- function **readBytes(Number aSize)**
 - **Number aSize**
Description: Reads aSize bytes from the stream
- function **toString(): String**
- **Process:** The Process object is used to start external programs
 - function **Process.execute(String aProgram, Array aArguments): Number**
 - **String aProgram**
 - **Array aArguments**
Description: Starts the program aProgram with the arguments aArguments in a new process, waits for it to finish, and then returns the exit code of the process
 - function **Process.startDetached(String aProgram, Array aArguments): Boolean**
 - **String aProgram**
 - **Array aArguments**
Description: Starts the program aProgram with the arguments aArguments in a new process, and detaches from it. Returns true on success
- **SpectrumRegion:** The SpectrumRegion objects allow us to define a region in the spectrum. The objects of type SpectrumRegion have the following properties
 - **SpectrumRegion (SpectrumRegion aSpectrumRegion)**
 - SpectrumRegion aSpectrumRegion:
 - **SpectrumRegion (Number aFrom, Number aTo)**
 - Number aFrom:
 - Number aTo:
Description: limits of the spectrum in 1D-NMR
 - **SpectrumRegion (Number aFrom1, Number aTo1, Number aFrom2, Number aTo2)**
 - Number aFrom1:
 - Number aTo1:
 - Number aFrom2:
 - Number aTo2:
Description: limits of the spectrum in 2D-NMR
 - function **toString()**: return String
 - function **from(Number aDim)**: return
 - Number aDim:
Description: Returns the 'from' of the region in the aDim dimension



o function **to**(Number aDim): return

- Number aDim:

Description: Returns the 'to' of the region in the aDim dimension

o function **dim**(): return

Description: Dimension of the region

- **NMRProcessing**: The NMRProcessing objects allow us to process the NMR spectra. The objects of type NMRProcessing have the following properties

o Boolean

isValid

Description: The function isValid informs if the spectrum obtained is correct

o **NMRProcessing** ()

o function **toString**(): return

o function **getParameter**(String aArgumentString): return Variant

- String aArgumentString: This string has the form function1.function2...functionN.parameter.

Description: Returns the value of the aArgumentString processing parameter. For example: "Apodization[1].Exp.Value" to return the value of the exponential apodization function. In order to see the list of the aArgumentStrings, please click [here](#)

o function **setParameter**(String aArgumentString, Variant aValue): return Variant

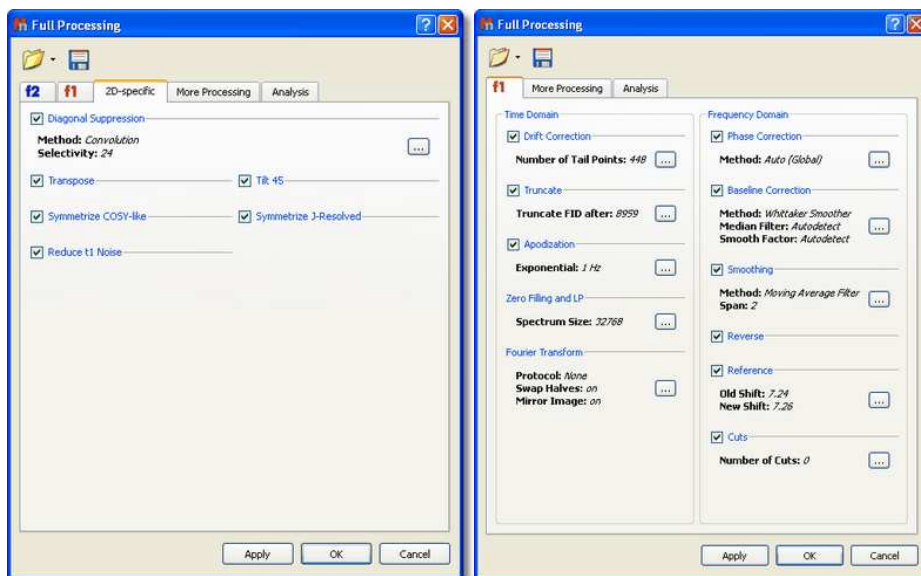
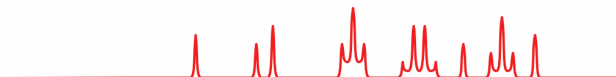
- String aArgumentString: This string has the form function1.function2...functionN.parameter

- Variant aValue: Value

Description: Sets the value of the aArgumentString processing parameter.

NMRProcessing. Describing Processing parameters (Argument Strings).

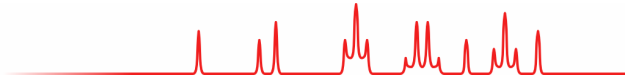
Below you will find a list of the argument strings (which have the form **function1.function2...functionN.parameter**) that you will be able to use in your processing scripts. These parameters appear in the graphical interface of the software (in the 'Processing / Full Processing' menu) and are described in depth in the manual of Mnova (hyperlink a processing menu).



- Boolean *Rev[aDimension]*: Reverse
- Boolean *Transpose*:
- Boolean *Tilt45*:
- Boolean *CosySym*: Symmetrize COSY-like
- Boolean *JResolv*: Symmetrize JResolved
- Boolean *Invert*:
- Boolean *Reduct1*: Reduce t1 Noise
- Boolean *Inadequate*:
- function *DC*: Drift Correction
 - Boolean *Apply*:
 - Number *Value*: Number of tail points
- function *Apodization[aDimension]*: Apodization
 - function *Exp*: Exponential
 - Boolean *Apply*:
 - Number *Value*:
 - function *Gauss*: Gaussian
 - Boolean *Apply*:
 - Number *Value*:
 - function *Sine*: Sine Bell
 - Boolean *Apply*:
 - Number *Value*:



- function **Sine2**: Sine Square
 - Boolean *Apply*:
 - Number *Value*:
- function **Traf**: TRAF
 - Boolean *Apply*:
 - Number *Value*:
- function **Trap**: Trapezoidal
 - Boolean *Apply*:
 - Number *Value*:
- function **FP**: First Point
 - Boolean *Apply*:
 - Number *Value*:
- function **Han**: Hanning
 - Boolean *Apply*:
 - Number *Value*:
- function **ConDif**: Convolution Difference
 - Boolean *Apply*:
 - Number *Value1*:
 - Number *Value2*:
 - Number *Value3*:
- function **Lr**: Linear Ramp
 - Boolean *Apply*:
- function **Ft**[aDimension]: Fourier Transform
 - Boolean *Apply*:
 - Boolean *Quadrature*:
 - Boolean *Invert*:
 - Boolean *RealFT*:
 - Boolean *Hyper*: Hyperphase
 - String *Protocol*: Valid Values: { "None" "Hyper" "Echo-Antiecho" }
- function **Zf**[aDimension]: Zero Filling
 - Number *size*:
- function **Lp**[aDimension]: Linear Prediction
 - function **Forward**:
 - Boolean *Apply*:
 - Number *First*:



- Number *Last*:
- function **Backward**:
 - Boolean *Apply*:
 - Number *First*:
 - Number *Last*:
- Number *size*:
- String *method*: Valid Values: { "Toeplitz" "ZhuBax" "Burg" "Debug" }
- Number *BasisPoints*:
- Number *Coefficients*:
- function **Pc**[aDimension]: Phase Correction
 - String *Method*: Valid Values: { "Global" "Selective" "Metabonomics" "Magnitude" "Power" "Manual" "NoPC" }
 - Number *Ph0*:
 - Number *Ph1*:
- function **Bc**[aDimension]: Baseline Correction
 - Boolean *Apply*:
 - Number *PolyOrder*:
 - Number *Filter*:
 - Number *Smooth*:
 - String *algorithm*: Valid Values: { "Whittaker" "PolyFit" "Bernstein" }
- function **Ds**: Diagonal Suppression
 - Boolean *Apply*:
 - String *Method*: Valid Values: { "Convolution" "ShiftedConv" "Wavelet" }
 - Number *Selectivity*:
 - String *Diagonal*: Valid Values: { "Std" "Double" "Quad" "Inverted" }
- function **Ss**: Signal Suppression
 - Boolean *Apply*:
 - String *Method*: Valid Values: { "Convolution" "ShiftedConv" "Wavelet" }
 - Number *Selectivity*:
 - NumberArray *Signals*:
- function **Smooth**[aDimension]: Smooth
 - function **Whittaker**:
 - Boolean *AutoLambda*:
 - Number *Lambda*:
 - function **Wavelet**:



- Boolean *Universal*:
- Boolean *Soft*:
- Number *Scales*:
- Number *Fraction*:
- function **SG**: Savitzky-Golay
 - Number *Width*:
 - Number *Order*:
- function **Average**:
 - Number *Span*:
- Boolean *Apply*:
- String *Method*: Valid Values: { "Whittaker" "SG" "Wavelet" "Average" }
- function **Pp**: Peak Picking
 - Boolean *Apply*:
 - Number *Sensitivity*:
 - Boolean *Parabolic*:
 - String *PeakType*: Valid Values: { "All" "Positive" "Negative" }
 - Number *MaxPeaks*:
 - Boolean *MergeMult*:
 - Number *f2Max*:
 - Number *f1Max*:
- function **Binning**[aDimension]: Binning
 - Boolean *Apply*:
 - Boolean *Full*:
 - Number *From*:
 - Number *To*:
 - Number *Width*:
- function **Ref**[aDimension]: Reference
 - Boolean *Apply*:
 - Number *OldShift*:
 - Number *NewShift*:
 - Boolean *AutoTune*:
 - Number *Tolerance*:
- function **Norm**: Normalize
 - Boolean *Apply*:

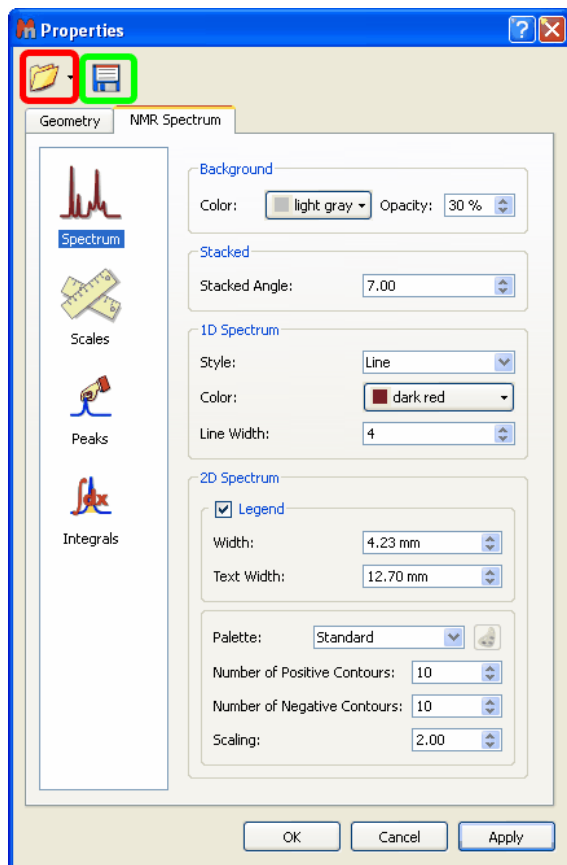


- String *Method*: Valid Values: { "TotalArea" "LargestPeak" "PeakPos" "PeakRange" "Manual" }
- Number *Value*:
- Number *PeakPos*:
- NumberArray *Range*:
- function **Trunc**[aDimension]: Truncate
 - Boolean *Apply*:
 - Number *After*:
- function **Comp**: Compression
 - Number *Ratio*:
- function **Mult**: Multiplet Analysis
 - Boolean *Apply*:
- function **Integration**: Integration
 - function **Auto**: Autodetect Regions
 - Number *Sensitivity*:
 - Boolean *AutoSensitivity*:
 - Number *MergeDistance*:
 - Number *MinArea*:
 - Boolean *Apply*:
 - String *Method*: Valid Values: { "Auto" "Predefined" }
 - SpectrumRegionArray *regions*
- function **Cuts**[aDimension]:
 - Boolean *Apply*:
 - SpectrumRegionArray *List*:



Mnova Properties

Below you will find the spectrum properties (*aParamName*) that you can manage via script, by using the functions `'SetProperty'` and `'GetProperty'`. You will find an example script in the examples/scripts installation directory and also in the `'Script Samples'` chapter. All of these properties can also be selected by using the GUI:



- String *background.color*: You will be able to set the background colour. For further information about the valid values, please take a look at: <http://www.w3.org/TR/SVG/types.html#ColorKeywords>
- Number *background.opacity*: to select the transparency of the background.
- String *curve.linestyle*: to select the linestyle between line, crosses or circles
Valid Values: { "line" "crosses" "circles" }
- String *curve.color*: To select the colour of the spectrum
Valid Values: { "#RRGGBB" "<http://www.w3.org/TR/SVG/types.html#ColorKeywords>" }
- Number *curve.linewidth*: to select the linewidth of the spectrum



- String *legend.width*: To select the width of the legend graphic. The get function needs an unit argument. The argument of the set function is a string like "1 mm"
- Boolean *legend.show*: to display the legend or not.
- String *legend.textwidth*: the width of the text legend. The get function needs an unit argument. the argument of the set function is a string like "1 mm" .
- Number *contours.positivenumber*: to select the number of positive contours.
- Number *contours.negativenumber*: to select the number of negative contours.
- Number *contours.scaling*: to select the scale of the contours
- String *palette*: to customize the palette

Let's take a look at the '**Scales**' properties:

- Boolean *grid.showhorizontal*: to display the horizontal grid.
- Boolean *grid.showvertical*: to display the vertical grid.
- Boolean *grid.showover*: to show the spectrum over the grid
- Boolean *grid.showframe*: to display the frame of the spectrum
- Boolean *grid.showtracesframe*: to display the frame of the traces
- String *grid.color*: to change the colour of the grid.
Valid Values: { "#RRGGBB"
"<http://www.w3.org/TR/SVG/types.html#ColorKeywords>" }
- String *axes.color*: to change the colour of the axes
Valid Values: { "#RRGGBB"
"<http://www.w3.org/TR/SVG/types.html#ColorKeywords>" }
- String *axes.font*: to select the font of the scales
- String *axes.margin*: to select the margin of the axes. The get function needs an unit argument. The argument of the set function is a string like "1 mm".
- Boolean *axes.horizontal.showlabel*: to display the label of the horizontal axis.
- Boolean *axes.vertical.showlabel*: to display the label of the vertical axis.
- Number *axes.horizontal.maxdigits*: to select the maximum number of displayed digits in the horizontal axis.
- Number *axes.vertical.maxdigits*: to select the maximum number of displayed digits in the horizontal axis.
- String *axes.horizontal.label*: to show/hide the horizontal label.
- String *axes.vertical.label*: to show/hide the vertical label.
- String *axes.vertical.units*: to select the units of the vertical scale (ppm, Hz, pt, seconds).
Valid Values: { "ppm" "hz" "pt" "sec" }



- String *axes.horizontal.units*: to select the units of the horizontal scale (ppm, Hz, pt, seconds).
Valid Values: { "ppm" "hz" "pt" "sec" }

Below you will find the objects related to the '**Peaks**' properties:

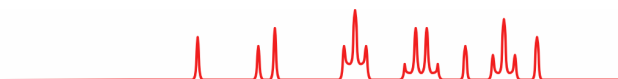
- Boolean *peaks.show*: to show/hide the peaks.
- String *peaks.color*: to change the colour of the peaks labels.
- String *peaks.font*: to change the font of the peaks labels.
- Boolean *peaks.showtick*: to show/hide the tick of the 1D-NMR peaks.
- String *peaks.pointer*: to select the peaks pointer in 2D-NMR (between: none, arrow or line).
- String *peaks.units*: to select the units of the peaks
Valid Values: { "ppm" "hz" }
- Number *peaks.decimals*: to select the number of the decimals in the peaks label.

Below you will find the objects related to the '**Integrals**' properties:

- Boolean *integrals.show*: to show/hide the integrals.
- Boolean *integrals.label.show*: to show/hide the integral labels.
- String *integrals.label.color*: to change the colour of the integral labels.
Valid Values: { "#RRGGBB" "<http://www.w3.org/TR/SVG/types.html#ColorKeywords>" }
- String *integrals.label.font*: to select the font of the integral label.
- Number *integrals.label.decimals*: to select the number of decimals of the label.
- String *integrals.label.position*: to select the position of the label.
Valid Values: { "Segment" "Curve" }
- Number *integrals.label.margin*: to select the margin of the label.
- Boolean *integrals.curve.show*: to show/hide the integral curve.
- String *integrals.curve.color*: to select the colour of the integral curve.
Valid Values: { "#RRGGBB" "<http://www.w3.org/TR/SVG/types.html#ColorKeywords>" }
- Number *integrals.curve.margin*: to select the margin of the integral curve.
- Number *integrals.curve.maxheight*: to select the maximum height of the integral curve.
- String *integrals.curve.shape*: to select the shape of the integral curve in 2D-NMR.
Valid Values: { "Ellipse" "Rectangle" }

Let's take a look at the '**Multiplets**' properties:

- Boolean *multiplets.show*: to show/hide the multiplet boxes.
- String *multiplets.font*: to change to font of the multiplets.



- String *multiplets.fontcolor*: to change the color of the font.
Valid Values: { "#RRGGBB"
"http://www.w3.org/TR/SVG/types.html#ColorKeywords" }
- Number *multiplets.decimals*: to select the number of decimals of the multiplets.
- Number *multiplets.margin*: to select the margin of the multiplet boxes.
- String *multiplets.labelcolor*: to select the colour of the multiplet boxes.
Valid Values: { "#RRGGBB"
"http://www.w3.org/TR/SVG/types.html#ColorKeywords" }

Scripts Samples

Other Scripts samples:

All the scripts are present in the examples/scripts installation directory. Some of them will be described in this chapter. Please bear in mind that it is also possible to run scripts from the command line, just by typing the path where the Mnova .exe file and the script are located and -sf "name of the script function": **"mestrenovaPathname" "scriptPathname" -sf "scriptFunctionToRun"**

For example:

```
"C:\Program Files\Mestrelab Research S.L\MestReNova\MestReNova.exe" "C:\Program Files\Mestrelab Research S.L\MestReNova\scripts\myScript.qs" -sf "myFunction"
```

It is also possible to specify arguments:

```
"C:\Program Files\Mestrelab Research S.L\MestReNova\MestReNova.exe" "C:\Program Files\Mestrelab Research S.L\MestReNova\scripts\myScript.qs" -sf "myFunction",0.1,10,true,off
```

In this case, MNova will run myFunction(0.1, 10, true, "off") from myScript.qs with the arguments specified. No spaces after commas!

Open and Save Document Script: Let's take a look at an easy script which you can use to open a .mnova file and export it as a PDF file. In this case, we will open a .mnova file named 'Dimethoxy':

```
function openSaveDocument()
{
    //If the open function gives a 'false', it will mean that Mnova does not recognize the file
    if( serialization.open("/dimethoxy.mnova") )
    {
        print("File Opened");
        //Let's see what the document contains
        //To get the active document
        var dW = new
        DocumentWindow(Application.mainWindow.activeWindow());
```



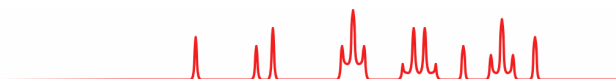
```

//To get information about the number of the pages of the document
print(dW.pageCount());
//To print the number of items and the description of them for each page
for( var i = 0; i < dW.pageCount(); i++ )
{
    var pag = new Page(dW.page(i));
    print ( "Page number "+i+" has "+pag.itemCount()+" items" );
    for( var j = 0; j < pag.itemCount(); j++ )
    {
        var item = new PageItem(pag.item(j));
        print( "\t"+item.name );
        //If the item is an NMR spectrum, it will print "This is an NMR
spectrum", if not, the function isValid will give a "False"
        var spectrum = new NMRspectrum(item);
        if(spectrum.isValid() )
            print( "This is an NMR Spectrum" );
    }
}
//To save as a PDF file
serialization.save("/dimethoxy.pdf", "pdf");
}

```

To run this script, just copy it to the 'Edit Script' dialog box and type "openSaveDocument()" in the combobox. Please bear in mind that, you will need to write the corresponding path of the saved Mnova file in the line 4 of the script (in this case the **dimethoxy.mnova** file was saved in C:\). As you can see in the script, if Mnova does not recognize the spectrum, you will obtain a 'False' message in the output due to the statement:
if(serialization.open("/dimethoxy.mnova"))

The line **print("File Opened");** will give the corresponding message in the output, as you can see in the picture below:



```

function openSaveDocument()
{
    //If the open function gives a 'false', it will mean that Mnova does not recognize the file
    if( serialization.open("/dimethoxy.mnova") )
    {
        print("File Opened");
        //Let's see what the document contains
        //To get the active document
        var dW = new DocumentWindow(Application.mainWindow.activeWindow());
        //To get information about the number of the pages of the document
        print(dW.pageCount());
        //To print the number of items and the description of them for each page
        for( var i = 0; i < dW.pageCount(); i++ )
        {
            var pag = new Page(dW.page(i));
            print ( "Page number "+i+" has "+pag.itemCount()+" items" );
            for( var j = 0; j < pag.itemCount(); j++ )
            {
                var item = new PageItem(pag.item(j));
                print( "\t"+item.name );
                //If the item is an NMR spectrum, it will print "This is an NMR spectrum", if not, the function isValid will give a "False"
                if(spectrum.isValid() )
                    print( "This is an NMR Spectrum" );
            }
        }
        //To save as a PDF file
        serialization.save("/dimethoxy.pdf", "pdf");
    }
}

```

Output

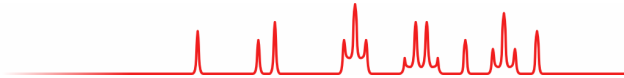
```

File Opened
2
Page number 0 has 1 items
    NMR Spectrum
This is an NMR Spectrum
Page number 1 has 5 items
    NMR Spectrum
This is an NMR Spectrum
    Molecule
    Arrow
    Rectangle
    Ellipse

```

The object 'Application' of the variable `var dW = new DocumentWindow(Application.mainWindow.activeWindow());` gives us the 'Active Window' of the 'Main Window', that is to say, the document which we can see in the screen. The function `print(dW.pageCount());` is used to know the number of the pages of the document.

The following lines of the script are used to obtain information about the items included on the document. For example the function `pageCount` is used to know the number of pages of the document. The variable `var pag = new Page(dW.page(i));` will analyze page by page to find the information. The line `print ("Page number "+i+" has "+pag.itemCount()+" items");` will print the corresponding phrase in the output, following the number of the page (+i+) and the number of the items included in the page `for(var j = 0; j < pag.itemCount(); j++)`. The variable `var item = new PageItem(pag.item(j));` will inform about the type of the item, and the function `print("\t"+item.name);` will print the name of the item. In this example (as we can see in the output of the picture below), the document has 2 pages; page 0 has 1 item, which is an NMR spectrum and page 1, has 5 items (NMR Spectrum, molecule, arrow, rectangle and ellipse).



The variable `var spectrum = new NMRspectrum(item);` will rebuild the active spectrum, and if the item is an NMR spectrum, it will print "This is an NMR spectrum", if not, the function `isValid` will give a "False" due to the lines: `if(spectrum.isValid()) // print("This is an NMR Spectrum");`

Finally, the plugin `serialization.save("/dimethoxy.pdf", "pdf");` will save the document as a PDF file in the specified location of the hard-disk (in this case, the PDF will be saved in C:\).

Import_Process_Save_Loop Script: This script can be used to apply a previously existing 'full processing' template (in this case, it is a template saved at C:/temp/proc.mnp) to several spectra (saved at C:/spectra/ and inside sub-folders with the extension ".fid", for example: "C:/spectra/1H.fid") and to save the result in different file formats (.mnova, .pdf and .txt). This script is only valid for these spectra containing a FID file.

```

/*****
Copyright (C) 2007 Mestrelab Research S.L. All rights reserved.

This file is part of the MNova scripting toolkit.

Authorized users of MNova Software may use this file freely, but this file is provided AS IS
with NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN,
MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE.
*****/
// This example function demonstrates how to apply some processing to a set of spectral files and
to save the result in the different file formats.
function import_process_save_loop()
{
    const dirName = "c:/spectra"; // Source directory with spectra to be processed.
    const resultDirName = dirName; // Destination directory where processed spectra should
be put.
    const procName = "c:/temp/proc.mnp"; // Processing file. It must be created in the Full
Processing dialog of MestReNova.
    const fileMask = "fid"; // This mask specifies files to be processed.
    const dirMask = "*.fid"; // This mask specifies directories to be scanned for spectra.

    var dir = new Dir(dirName);
    if (!dir.exists)
    {
        MessageBox.critical("Directory " + dirName + " does not
exist.",MessageBox.Ok);
        return;
    }
    dir.cdUp();

    // The below function getMaskFiles is located in the files.js script.
    // If you run import_process_save_loop from QSA Workbench then it is necessary to
introduce getMaskFiles to the interpreter by importing files.js.
    var files = getMaskFiles(dirName, dir.absPath, fileMask, dirMask, true);
    for(var i = 0; i < files.length; i++)

```



```

    {
        var dw = new DocumentWindow(Application.mainWindow.newWindow()); //
Create new document window
        if (serialization.open(files[i]))
        {
            nmr.process(procName);
            const toReplace = new RegExp("[/:]", "g");
            fileName = resultDirName + "/" + files[i].replace(toReplace, "_"); //
Generate a filename for processed spectrum.
            serialization.save(fileName+".mnova", "mnova"); // MestReNova file
format
            serialization.save(fileName+".pdf", "pdf"); // Adobe PDF file format
            serialization.save(fileName+".txt", "ascii"); // ASCII
            print(fileName); // Prints file name in the debug window
        }
        dw.close(); // Close document window of processed spectrum
    }
}

```

Parameters Script: This script can be used to obtain the desired parameters on your spectrum. You will be able to add or remove any parameter and to change the font by just using HTML code.

```

function spectrumParams()
{
    var spec = new NMRspectrum( nmr.activeSpectrum() );
    var htmlText = "<b>Nucleus</b>: %1 <br/> <b>Frequency: </b> %2 <br/> <b>Pulse
Sequence: </b> %3 <br/> <b>psLabel: </b> %4 <br/> <b>Solvent: </b> %5 <br/> <b>Title: </b>
%6";
    var nuc = spec.getParam("Nucleus[1]");
    var freq = spec.getParam("Spectrometer Frequency");
    var seqfill = spec.getParam("Pulse Sequence");
    var psLabel = spec.getParam("psLabel");
    var Solvent = spec.getParam("Solvent");
    var Title = spec.getParam("Title");
    print(spec.getParam("Nucleus[1]"));
    print(spec.getParam("Spectrometer Frequency"));
    print(spec.getParam("Pulse Sequence"));
    print(spec.getParam("psLabel"));
    print(spec.getParam("Solvent"));
    print(spec.getParam("Title"));
    htmlText=htmlText.arg(nuc).arg(freq).arg(seqfill).arg(psLabel).arg(Solvent).arg(Title);
    draw.text(htmlText, true);
}

```

In this case, the script will print the parameters Nucleus, Spectrum Frequency, Pulse Sequence, psLabel, Solvent and Title. Please bear in mind that getParam function returns only parameters which exist in the table of parameters (including invisible ones). Please make sure that you have these parameters included on the table of parameters (via customization dialog).



Fid Test Script: you will get information about the real and the imaginary part of the FID by running this script:

```
function fidTest()
{
    var spec = new NMRpectrum( nmr.activeSpectrum() );
    print(spec.fidImag(0));
    if( spec.fidSetImag(0,0) )
        print(spec.fidImag(0));
        print(spec.fidReal(0));
    if(spec.fidSetReal( 0, 0 ) )
        print(spec.fidReal(0));
    spec.process();
    mainWindow.activeWindow().update();
}
```

The command `print(spec.fidImag(0))` will return the value of the imaginary part of the FID at the *aIndex* point (in this case; 0. You can type any number from zero to the number of points of your spectrum), whilst the line `print(spec.fidReal(0));` will give you the value of the real component. The object *fidsetImag*; sets the imaginary value of the *aIndex* point of the spectrum fid.

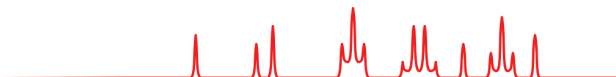
Integral Test Script: This script can be used to get information about the integrals of the spectrum:

```
function integralTest()
{
    var spec = new NMRpectrum( nmr.activeSpectrum() );
    var intList = new Integrals(spec.integrals());
    print(intList);
    print(intList.count);
    print(intList.normValue);

    var myInt = new Integral(intList.at(0));
    print(myInt);
    print(myInt.rangeMax(1));
    print(myInt.rangeMin(1));
    if( spec.dimCount > 1 )
    {
        print(myInt.rangeMax(2));
        print(myInt.rangeMin(2));
    }

    print("Integral Value "+myInt.integralValue());
    print("Normalized value "+myInt.integralValue(intList.normValue));

    var sReg = new SpectrumRegion(7.2, 7.5);
    var newInt = new Integral( spec, sReg, false );
    print(newInt);
    print(newInt.integralValue());
    spec.integrals().append(newInt);
    spec.process();
}
```



The first part of the script is used to count the integrals and to report the normalized value applied to the integrals (in the list 'intList'). With the second part of the script, the user is able to select any integral of the list (in this case the integral which appears in the first position, if we wanted to select the second integral, we should have to type `var myInt = new Integral(intList.at(1));`). Then the script returns the absolute value and the limits of the selected integral. The third part of the script returns the absolute and the normalized value of the integral.

The last part of the script can be used to add a new integral to the list (in this case an integral with the spectral region from 7.2 to 7.5 ppm). the script reports the absolute value of this integral.

Multiplet Test Script: This script is valid to obtain information about the multiplets of the spectrum and also to apply a 'manual multiplet analysis':

```
function multipletTest()
{
    var spec = new NMRspectrum( nmr.activeSpectrum() );
    print(spec.multiplets());
    var sMult = new Multiplets(spec.multiplets());
    print(sMult.count);
    print(sMult.at(2));

    var sReg = new SpectrumRegion(7.0, 7.5);
    var mult = new Multiplet( spec, sReg );
    var myMultiplets = new Multiplets();
    myMultiplets.append( mult );

    var mult2 = new Multiplet( spec, new SpectrumRegion(3.80,3.90) );
    myMultiplets.append( mult2 );

    var nM = new Multiplet(new SpectrumRegion(3.65,3.75), "t");
    nM.name="My multiplet";
    myMultiplets.append(nM);

    print(myMultiplets);
    spec.setMultiplets(myMultiplets);
    print(spec.multiplets());
    spec.process();
}
```

The first part of the script creates a list of the multiplets which appear on the spectrum and prints some information about the selected multiplet (in this case, the multiplet number 2, but of course you can select any other multiplet). The second part of the script creates a multiplet in the selected region 'sReg' (in this case the region is 7.0-7.5 ppm. It works as a manual multiplet analysis). Then, it creates a new empty list of multiplets which is added.

The third part of the script consists in a new region (from 3.80 to 3.90 ppm) to apply the multiplet analysis. The following part of the script is used to add a multiplet manually to the multiplet list (the user will be able to remove this part of this script without any problem). In this case, we wanted to add a triplet, which appears in the region from 3.65 and 3.75 ppm, with the name 'My multiplet'.



The end of the script is used to replace the original multiplet list for the list which we have just created.

Peak Test Script: A similar script used to obtain information about the chemical shift and the intensity of the peaks of a 1D-NMR spectrum:

```
function peakTest()
{
    var spec = new NMRspectrum( nmr.activeSpectrum() );
    var sR = new SpectrumRegion( 0, 9.5 );
    var ps = new Peaks(spec, sR); //fill the list with the peaks in sR
    print(ps);
    var p = new Peak(1.265, spec);
    print(p);
    ps.append(p);
    spec.setPeaks(ps); //Set the peaks to ps
    print( spec.peaks());
    spec.process();
}
```

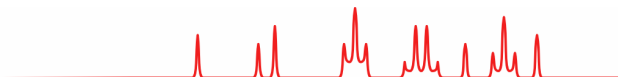
After running this script, we will get the peak picking analysis of a 1D-NMR spectrum from 0 to 9.5 ppm (of course, we will be able to change this range). Then, a new peak at 1.265 ppm will be added to the peak list (again, you can add any other peak, if you want), reporting the intensity value of this new peak. Finally, the spectrum will be re-analyzed to obtain the new number of peaks.

A similar script can be used for a 2D-NMR spectrum, but in this case we have to type the corresponding spectrum region in 2D mode:

```
function peakTest()
{
    var spec = new NMRspectrum( nmr.activeSpectrum() );
    var sR = new SpectrumRegion(4.0, 3.5, 75, 65 );
    var ps = new Peaks(spec, sR); //fill the list with the peaks in sR
    print(ps);
    var p = new Peak(72.73, 3.17, spec);
    print(p);
    ps.append(p);
    spec.setPeaks(ps); //Set the peaks to ps
    print( spec.peaks());
    spec.process();
}
```

Multiplet Reporter script: Let's take a look now at a very important part of the 'Multiplet Reporter' script:

```
// This function defines JACS-like multiplet report.
// To customize report, edit this class or implement another one.
function JACSMultipletReporter()
{
    MultipletReporter.call(this);
}
```



```

this.onlyElementName=false;
// Define font size and font family
this.font = "<font style=\"font-size: 10pt; font-family: Times New Roman\">";

this.nucleusTemplate="%1";
// Report header. %1 will be replaced with nucleusString, %2 with frequency, %3 with
solvent
this.header = "%1 NMR (%2 MHz, %3) &delta; ";

// Multiplet templates. %1 - delta, %2 - category, %3 - nH
this.reportRange = true; // set to true to get multiplet range instead of delta.
this.withoutJsTemplate = " %1 (%2, %3H)"; // multiplet template without J's
this.withJsTemplate = " %1 (%2, %4, %3H)"; // multiplet template with J's
this.rangeTemplate = "%1 &ndash; %2";

// J's list template. %1 - list of J's
this.jListTemplate = "<i>J</i> = %1";

this.jPrecision = 1; // J's precision
this.deltaPrecision = 2; // delta precision
this.mSeparator = ", "; // multiplet separator
this.jSeparator = ", "; // J's separator

this.start = this.font;
this.end = "</font>";
}

JACSMultipletReporter.prototype = new MultipletReporter();
JACSMultipletReporter.prototype.toString = function() { return "JACSMultipletReporter()";
}

(...)

```

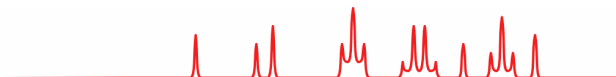
The user will be able to change the multiplet report template to obtain the desired multiplet report; for example:

`this.onlyElementName=false;` changing false with true, we will obtain only the element name without the atomic mass (For example: H, C instead of ^1H , ^{13}C).

The function: `this.font = "";` will define the font size and the font family of the multiplet report.

The line: `this.header = "%1 NMR (%2 MHz, %3) δ ";` is used to print the header of the report, where %1 will be the nucleus (H or C), %2 the frequency of the spectrometer (in MHz), and %3 the solvent, followed by a delta symbol (δ). For example: ^1H NMR (500 MHz, CDCl₃) δ .

The sentence: `this.reportRange = true` is used to obtain the multiplet range instead of the chemical shift.



The functions: `this.withoutJsTemplate = " %1 (%2, %3H)"` and `this.withJsTemplate = " %1 (%2, %4, %3H)"` are used to customize the appearance of the multiplet report by changing the positions of %1, %2, %3H, or %4 (where, %1 means: chemical shift; %2 means: type of multiplet (s, d, t, etc); %3H means: number of hydrogens and %4 means the coupling constant value). As you can see, the first line shows a multiplet without coupling constants, (while the last line shows a multiplet with coupling constants).

So, if you need to obtain something like this (Japanese format):

1H NMR (300 MHz, Solvent) δ ppm 6.43-6.22 (1 H, m), 3.17 (1 H, q, J = 7.15 Hz) etc...

You should modify both lines, as you can see below:

```
this.withoutJsTemplate = %1 (%3H, %2);
this.withJsTemplate = " %1 (%3H, %2, %4)";
```

If you prefer to obtain something like this:

1H NMR (300 MHz, Solvent) δ ppm 1.23 (d, J = 1.2 Hz, 3 H), etc...

Just replace the original lines with:

```
this.withoutJsTemplate = %1 (%2, %3H);
this.withJsTemplate = " %1 (%2, %4, %3H)";
```

To obtain the coupling constant symbol in normal instead of *'italic'*, just modify the script by removing the italic format (`<i>J</i>`). If you prefer to obtain it in "bold" just type:

```
this.jListTemplate = "<b>J</b> = %1";
```

The following paragraph will be used to customize the appearance of the coupling constants list:

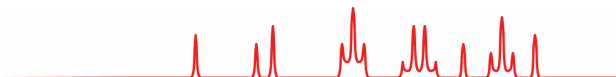
```
this.jPrecision = 1; // J's precision
this.deltaPrecision = 2; // delta precision
this.mSeparator = ", "; // multiplet separator
this.jSeparator = ", "; // J's separator
```

The first line is used for the precision of the coupling constants values, the second will be used for the precision of the chemical shift and the remaining two lines will print the separation between the multiplets and the coupling constants values.

If you need to obtain the coupling constants in descending order, replace `'true'` with `'false'` in the following line:

```
var jList = new JList(multiplet.jList());
jList.sort(true);
```

If you want to obtain the multiplet chemical shifts in ascending order, just replace the `'false'` with `'true'` in the script:



```
var multiplets = new Multiplets(spectrum.multiplets()); // get multiplets from
spectrum
jList.sort(true);
```

To obtain the multiplet range in ascending order, just replace the rangeMin with rangeMax (and vice versa) in the line below of the script:

```
shiftStr = this.rangeTemplate.argDec(multiplet.rangeMax, 0, 'f',
this.deltaPrecision).argDec(multiplet.rangeMin, 0, 'f', this.deltaPrecision);
```

Title and Solvent: running this script, you will get the title and the solvent on your spectrum

```
// <GUI menuname="title1" shortcut="Ctrl+2"
tooltip="Title_and_Solvent" />
function title1()
{
//To get the active spectrum
var spectrum = nmr.activeSpectrum();
//The function isValid informs about if the spectrum obtained is
correct
print( spectrum.isValid() );
var tit = draw.text("<h3>" + spectrum.title +
"+spectrum.solvent+"</h3>", true);
}
```

Zoom: This script can be used to apply a zoom into a selected range of a spectrum. In order to select the horizontal range, just type the desired values between the brackets on the line `spc.horZoom(1.0, 5.0);`

If you want to set the vertical zoom, just change the values of the line `spc.vertZoom(-100, 7000)`

```
function spectrumZoom()
{
var spc = nmr.activeSpectrum();
if( !spc.isValid() )
return;
spc.horZoom(1.0, 5.0);
spc.vertZoom(-100, 7000);
spc.update();
mainWindow.activeWindow().update();
}
```

Properties Script: You will find below a script example to change some properties of your spectra. In this case, the colour of the spectrum will be changed to red, the peaks font to Arial and the integral curves will be showed. You will find all the available properties at the ['Mnova Properties'](#) chapter:

```
function properties()
```



```
{
    var spec = new NMRspectrum(nmr.activeSpectrum());

    spec.setProperty("curve.color", "red");
    spec.setProperty("peaks.font", "Arial");
    spec.setProperty("integrals.curve.show", true);
    spec.update();
    mainWindow.activeWindow().update();
}
```

Cutting: This script can be used to apply cuts to the spectra. You will obtain two cuts in your spectrum, between 1.0-3.5 and 7.0-8.0 ppm respectively.

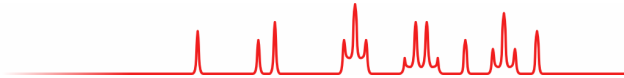
```
function myProcess()
{
    var spec = new NMRspectrum( nmr.activeSpectrum() );
    var p = new NMRProcessing(spec.proc);
    var regs = new Array(2);
    regs[0] = new SpectrumRegion(1.0, 3.5);
    regs[1] = new SpectrumRegion(7.0, 8.0);
    print(p.getParameter("cuts"));
    p.setParameter("cuts.apply", true);
    p.setParameter("cuts.list", regs);
    spec.proc = p;
    spec.process();
    mainWindow.activeWindow().update();
}
```

Magnitude Active Spectrum script: Let's see another script; in this case this script is used to obtain the 1D-NMR spectra in magnitude. Just open a 1D-NMR spectrum in Mnova, load this script by following the menu 'Script/Edit Script', type "magnitudeActiveSpectrum()" in the edit box and click on the green triangle to run the Script. You will obtain automatically your 1D-NMR spectrum in magnitude.

Please bear in mind that it is also possible to run scripts from the command line, just by typing the path where the Mnova .exe file and the script are located and -sf "name of the script function":
"mestrenovaPathname" "scriptPathname" -sf "scriptFunctionToRun"

For example:

```
"C:\Program Files\Mestrelab Research S.L\MestReNova\MestReNova.exe" "C:\Program Files\Mestrelab Research S.L\MestReNova\scripts\magnitudeActiveSpectrum.qs" -sf "magnitudeActiveSpectrum"
```



```
function magnitudeActiveSpectrum()
{
    //The magnitude of the active spectrum is calculated (only for 1D)
    //To get the active spectrum
    var spectrum = nmr.activeSpectrum();
    //To make sure that the spectrum is valid
    if( spectrum.isValid() )
    {
        //Only interested in 1D-NMR spectra
        if( spectrum.dimCount == 1 )
        {
            //The functions beginModification and endModification will allow us to save the initial state of the
            //spectrum to be able to apply Undo/Redo
            nmr.beginModification(spectrum);
            var ptsCount = spectrum.count();
            for(var k = 0; k < ptsCount; k++)
            {
                //To calculate the module of each point and to assign it
                re = spectrum.real(k);
                re *= re;
                im = spectrum.imag(k);
                im *= im;
                spectrum.setReal(k, Math.sqrt(re + im));
            }
            nmr.endModification(spectrum);
            //To get the active window and to refresh it so that the changes in the spectrum become visible
            Application.mainWindow.activeWindow().update();
        }
    }
}
```

Output

Threshold Script: This script will allow the user to adjust the threshold of the color scheme in 2D-NMR spectra:

```
// <GUI menuname="threshold2D" shortcut="Ctrl+T" tooltip="threshold2D" />
function threshold2D()
{
    var spectrum = nmr.activeSpectrum();
    if(!spectrum.isValid() )
        return

    var th = spectrum.threshold;

    var dlg = new Dialog();
    dlg.title = "Threshold";

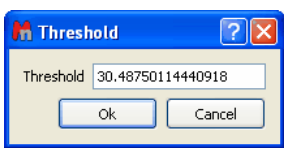
    var edit = new LineEdit();
    edit.label = "Threshold";
    edit.text = th;
    dlg.add(edit);
    if (dlg.exec())
    {
```



```
spectrum.threshold = edit.text;
}
spectrum.update();

mainWindow.activeWindow().update();
}
```

Running this script will show the 'Threshold' dialog box. From this window, the user will be able to adjust the threshold of the colour scheme by just typing the desired value and clicking on the 'OK' button.

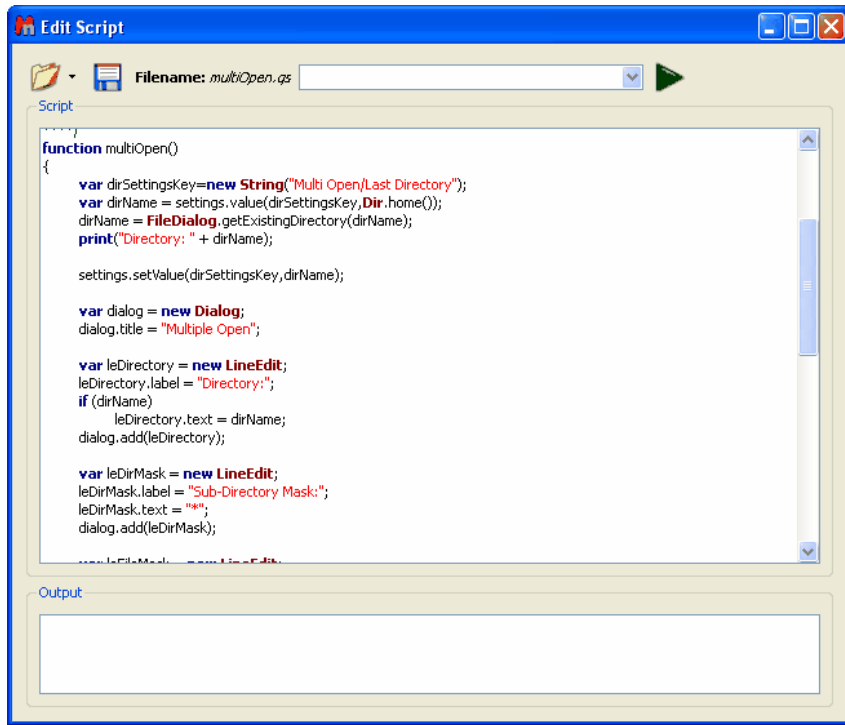
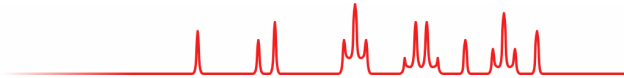


ProcessTest Script: This script can be used to get some data from Mnova, save it as a file, next run some external program in order to make calculations (or whatever you need) and once the external program finishes, Mnova will get the result from another file. In this case the external program is the 'Notepad'.


```
function processTest()
{
    fileName = Dir.temp() + "MNova_Process_Text.txt";
    var f = new File(fileName);
    f.open(File.WriteOnly);
    var s = TextStream(f);
    s.write("Hello Notepad!");
    f.close();
    print(Process.execute("notepad.exe", fileName));

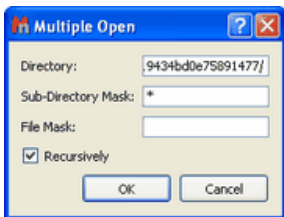
    f.open(File.ReadWrite);
    s.pos = f.size;
    print(s);
    s.write("\r\nHi again!");
    f.close();
    print(Process.startDetached("notepad.exe", fileName));
}
```

MultiOpen script: Mnova incorporates a script named 'MultiOpen'. This tool is very useful if, for example, you always process your spectra in the same way, or when you have to routinely handle a large number of datasets.



A practical example:

'*multiopen script*' allows to open simultaneously a collection of several spectra. Make sure that all spectra are located in the same folder. After that, select 'Run Script...' on the 'Script Menu' or click on the '**Run Script**' icon  on the toolbar and then click 'OK' after finding the corresponding script (in this case, the script is called 'multiOpen'). Then, select the directory where the spectra are located and a window like the one below will open:



Finally, write the 'File Mask' name (you can use wildcards, so this could be, for example, '*', '???', 'fid or ser), tick 'Recursively' and click 'OK'. All selected spectra will open simultaneously.

The user can also use the 'Full Processing' command to automate the processing of both 1D & 2D NMR data sets. (You can find a full description of this tool further on in the [Automated Processing](#) chapter.)