**Xcalibur Workbench**

The Xcalibur Workbench is a flexible data browser which supports customization and automated data processing through the Lua scripting language. It is built with three primary tools.

1. LuaInterface: This library provides access to .NET functionality, allowing the construction of the GUI.
2. ZedGraph: This is a .NET plotting library that supports line, pie, and bar charts.
3. LuaRawFile.dll: This library provides native access to Xcalibur .raw files.

The GUI is constructed using a MDI interface, using a notebook motif for each raw file, where each notebook is allowed to have multiple pages. All page types are derived from either a plot page or a grid page.

The Xcalibur Workbench is designed around Lua 5.1. It is recommended to run from Zerobrane Studio when customizing or creating data processing routines so that debugger support is available. To start the Xcalibur Workbench, just run workbench.lua. It will load the necessary supporting Lua files in the project’s working directory, along with all files located in user directory.

The Xcalibur Workbench can be associated with raw files in Windows so that double clicking will open the file directly in the Workbench. See the included XcaliburWorkBench.bat batch file for details.

**User Programs**

Lua files located in the User directory will automatically be loaded during initialization. User files may alter the GUI, add notebook templates, and provide custom processing options. User programs can use functionality supplied by the Workbench using the normal require() syntax. This applies both to the base functionality of the Workbench along with LC and MS functionality located in the utility files.

*local mdiNotebook = require(“mdiNotebook”)*

*local newNotebook = mdiNotebook()*

*local ms = require(“ms”)*

**Setup**

There are two recommended interpreters for running the Workbench. The simplest is “Lua For Windows”, which is a complete build of Lua 5.1 with many useful utilities, including the required LuaInterface files (https://code.google.com/archive/p/luaforwindows/). Install Lua For Windows and then select the *LuaForWindows* interpreter in ZeroBrane Studio. The second option is to run with the ZeroBrane Studio default Lua interpreter, which is really LuaJIT.

To access Xcalibur raw files from within the Workbench, you must have *LuaRawFile.dll* available in the cpath directory. The recommended location is the *clibs* directory of ZeroBrane Studio. You must also have *LuaRawFile.lua* in the Lua path. The recommended location is the *lualibs* directory of ZeroBrane Studio. These files are available as part of the LuaRawFile GitHub project. The source code for this library is also available on GitHub (https://github.com/thermofisherlsms/lua-raw-file).

.NET components are accessed via LuaInterface. This is composed of two separate files. The first is *luanet.dll*. When running with LuaJIT, this should be located in the *clibs* directory of ZeroBrane Studio. When running with LFW, this is included with the installation, and is located in the *clibs* directory of LFW. The second file is LuaInterface.dll. When running with LuaJIT, this should be located in the project directory. For LFW, this is included with the installation and located in the LFW *clibs* directory.

All graphing components come from ZedGraph. This consists of two files, ZedGraph.dll and ZedGraph.XML. For LFW, these should be located in the LFW directory along with the Lua.exe executable. For LuaJIT, these should be located in the project directory.

**GUI Features**

Zooming: Clicking with the left mouse button, hold the button down and dragging across the region of interest. Optionally, the scroll wheel will zoom in/out, but only based on the center of the plot. To unzoom, right click and select either “Un-zoom” or “Undo All Zoom/Pan”.

Panning: Hold down the control key and left mouse key and then drag.

Setting the Active Pane: Either double click the pane, or use the Page Up/Page Down keys to cycle through all the panes on a page.

Browsing: A property dialog can be displayed from Edit🡪Properties… Altering the information on this dialog and clicking OK will redraw the current display to reflect the values in the dialog. The spectrum pane will respond to several key clicks from the user. The left and right arrow keys will move the spectrum up or down one scan. If the shift key is held with the arrow, the spectrum will move 10 scans. If the control key is held, the spectrum will move 100 scans. If both the control and shift keys are held, the spectrum will move 1000 scans. Home will take you to the first spectrum and End will take you to the last spectrum. When the active pane is a spectrum, clicking in a chromatogram pane will display the spectrum at the point of the click. When the active pane is a chromatogram, clicking in a spectrum pane will display a unit resolution XIC centered at the point of the click. Any change made in the active pane by browsing can be reversed by choosing Edit🡪Undo.

Trends: From header and status grids, right clicking on any row will bring up a context menu for plotting a trend for that parameter across the entire file. The trend plot will be displayed in a new trendPage that is added to the current notebook.

**Components**

**mdiNotebook**

*mdiNotebook([{fileName, addPages}])*: Returns a new mdiNotebook using the optional arguments. If *fileName* is supplied, then the specified file will be opened. If *addPages* is supplied, the function will be used to add pages to the notebook on creation. See the *templates* section for more details on how to construct the *addPages* function.

*mdiNotebook.form*: A Winforms Form which is the MDI child form for the notebook. The *form* has a *Tag* property which is the Lua *mdiNotebook* which can be used for callback functions.

*mdiNotebook.tabControl*: A Winforms TabControl on the right side of the notebook used to control the displayed page. The *tabControl* has a *Tag* which is the Lua *mdiNotebook*, which can be used for callback functions.

*mdiNotebook.noteBookList:* A Lua table that includes all notebooks currently available in the workbench. This can be called either directly from *mdiNotebook* or can be called from any instance of a *mdiNotebook*. The *noteBookList* also has a keyed “*active*” entry which indicates the notebook which is currently active in the Workbench.

*mdiNotebook.pageList{}*: A Lua table that includes all pages in the notebook. It also includes an *active* entry, which is the page currently being show.

*mdiNotebook.rawFile*: A Lua userdata created by the Thermo library that allows access to the specified raw file.

*mdiNotebook.fileName:* File name for raw file, not including path.

*mdiNotebook.fullFileName:* Complete file name for raw file, including path.

*mdiNotebook.pathName:* Path to the raw file.

*mdiNotebook.GetActiveNoteBook()*: A Lua function to get the active notebook. Returns false if no active notebook.

*mdiNotebook.GetActivePage()*: A Lua function to get the active page of the active notebook. Returns false for no active notebook, or an active notebook which has no pages.

*mdiNotebook.GetActivePane()*: A Lua function to get the active pane of the active page of the active notebook. Returns false for no active notebook, or an active notebook which has no pages, or an active page with no panes, or an active page that’s not derived from *multiPlotPage*.

*mdiNotebook:AddPage(page)*: A method for adding a *page* to the notebook.

*mdiNotebook:Close()*: A method to close the notebook along with any associated raw file.

*mdiNotebook:GetUniquePageName(baseName)*: A method to create a unique page name for that notebook using a supplied base name and an integer suffix.

*mdiNotebook:Open(fileName)*: A method to open a raw file. Can be used if the raw file was not specified when the notebook was originally created.

**menu**

*menu.itemList{}*: A Lua table that includes a list of Winform MenuItems.

*menu.AddMenu({name, [label, callback, parentName, beforeName})*: Creates and returns a new Winforms MenuItem with the *name* specified by the argument table. If *label* is specified, this will be what is shown in the GUI, otherwise *name* will be shown. If *callback* is specified, this Lua function will be called when the menu item is selected in the GUI. If *parentName* is specified, the new MenuItem will appear under the MenuItem specified by *parentName*. If *parentName* is not specified, the new MenuItem will appear at the top level. If *beforeName* is specified, then the new MenuItem will appear before the MenuItem with the name that matches *beforeName*, otherwise the new MenuItem will appear at the end of the parent menu.

**configure**

*configure.chromatogramColor*: A Winforms Color, used as the default color when drawing chromatograms.

*configure.spectrumColor*: A Winforms Color, used as the default color when drawing spectra.

*configure.userDirectory*: The directory where user Lua files are located.

*configure.utilityDirectory*: The directory where Lua utility files are located. These are functions for handling normal MS and LC data processing tasks.

**tabPage**

*tabPage({name})*: Returns a new tabPage. The *name* argument will appear in the tab.

*tabPage.pageControl*: A Winforms *TabPage*. The *TabPage* has a *Tag* that is the Lua *tabPage* which can be used for callbacks.

*tabPage:ParentNotebook():* Returns the Lua parent notebook for the page.

**multiPlotPage**

*multiPlotPage({name, [{panes}]})*: Returns a new multiPlotPage, which is derived from a tabPage. If *panes* is specified, all panes contained in the table will be added to the page.

*multiPlotPage.plotControl*: A Zedgraph *ZedGraphControl*. The *plotControl* has a *Tag* which is the Lua *multiPlotPage*, which can be used during callback functions.

*multiPlotPage.paneList{}*: A Lua table containing all panes on the page. An *active* entry specifies the currently active pane.

*multiPlotPage:AddCurve()*: Adds a new curve to the active pane. See *pane:AddCurve()* for further details.

*multiPlotPage:AddPane(pane)*: Add the specified pane to the page and set it as the active pane.

multiPlotPage:AddXYTable(): Adds an XY series to the active pane. See *pane:AddXYTable()* for further details.

*multiPlotPage:ChangeActivePane(direction)*: Changes the active pane either up or down, depending on the setting of direction. Positive directions move down the page, while negative directions move up the page.

*multiPlotPage:SetActivePane(pane)*: Sets the specified pane to the active pane.

**spectrumPage**

spectrumPage({rawFile}): Returns a new spectrumPage associated with the specified *rawFile*, derived from multiPlotPage. The new page has just one pane.

spectrumPage:PlotSpectrum(): Plot a spectrum on the spectrumPage. See details in *msPane:PlotSpectrum()*.

**chromatogramPage**

*chromatogramPage({rawFile})*: Returns a new chromatogramPage associated with the specified rawFile, derived from multiPlotPage. The new page has just one pane.

chromatogramPage:PlotChromatogram(): Plot a chromatogram on the chromatogramPage. See details in *msPane:PlotChromatogram()*.

**trendPage**

*trendPage({rawFile})*: Returns a new *trendPage* associated with the specified *rawFile*, derived from *multiPlotPage*. The new page has just one *trendPane*.

**gridPage**

*gridPage()*: Returns a new gridPage, derived from tabPage.

*gridPage.gridControl*: A Winforms DataGridView.

*gridPage:Fill(data)*: Fills the DataGridView control with values in the Lua table data. The table must be rectangular (ie. all rows must have the same number of columns).

**headerPage**

*headerPage({rawFile, [skipInit]})*: Returns a new headerPage, derived from gridPage, and associated with the specified *rawFile*. If *skipInit* is true, the headerPage will not display initial data.

*headerPage:ShowHeader({scanNumber})*: Displays scan header and trailer information in the grid for the scan specified by *scanNumber*.

**statusPage**

*statusPage({rawFile, [skipInit]})*: Returns a new statusPage, derived from gridPage, and associated with the specified *rawFile*. If *skipInit* is true, the statusPage will not display initial data.

*statusPage:ShowStatus({scanNumber})*: Displays status log in the grid for the scan specified by *scanNumber*.

**tunePage**

*tunePage({rawFile, [skipInit]})*: Returns a new tunePage, derived from gridPage, and associated with the specified *rawFile*. If *skipInit* is true, the tunePage will not display initial data.

*statusPage:ShowTune()*: Displays tune report in the grid.

**textPage**

*textPage({})*: Returns a new textPage, derived from tabPage.

*statusPage:Fill(text)*: Displays the specified text in the texPage’s text box.

**methodPage**

*methodPage({rawFile, [skipInit]})*: Returns a new methodPage, derived from textPage, and associated with the specified *rawFile*. If *skipInit* is true, the methodPage will not display initial data.

*statusPage:ShowMethod()*: Displays method in the text box.

**zPane**

*zPane()*: Returns a new zPane.

*zPane.paneControl*: A ZedGraph *GraphPane*. Use this *GraphPane* to gain access to the curves using the ZedGraph *CurveItem* list. Syntax is *zPane.paneControl.CurveItem[n]*, where n is a base 0 index.

*zPane:AddCurve({[name, color, symbol, symbolSize, noLine, seriesStyle]})*: Add a new curve to the *paneControl*. If name is specified, this will show up in the legend. If color, a Winforms *Color*, is specified, it will be used for the curve instead of a default color. If *symbol*, a ZedGraph *SymbolType*, is specified, it will be used, otherwise no symbol will be shown. If *symbolSize* is specified, it will be used, otherwise the default will be used. If *noLine* is specified, then only the symbol will be shown. If *seriesType* is specified, it will be used, otherwise a generic ZedGraph curve will be used. Options for *seriesType* are “curve”, “stick”, and “bar”.

*zPane:AddPieSlice({value, [color, displacement, name, skipRedraw]}):* Add a pie slice to a *zPane*, which makes it a pie chart. The *value* for the slice must be specified. The *color* will be used for the slice if specified. The *displacement* (0 to 1) will displace the slice from the center of the pie if specified. The *name* will be used to label the pie slice. If *skipRedraw* is true, the call will not redraw the graph, and it must be drawn either manually or by a subsequent call to *AddPieSlice()*.

*zPane:AddXYTable({data, xKey, yKey, [index, xMin, xMax, yMin, yMax, skipDraw]})*: The *data* parameter must be a Lua table formatted as a list of points. Each point can be indexed with *xKey* and *yKey*, which can be either numeric or strings. An optional *label* key will label each point with the specified string. The optional *index* specifies the curve to use for plotting. If not specified, the curve at index 1 is used. If any of the optional axes limits are set, they will override the ZedGraph automatic settings. If *skipDraw* is true, the graph will not draw after adding the table and will need to be manually redrawn or redrawn with a subsequent AddXYTable() call.

*zPane:Clear()*: Clears points from all curves.

*zPane:SetActive(setting)*: Sets the *zPane* to the active one for the page. If *setting* is not false, then a blue border will be drawn around the *zPane*.

**msPane**

*msPane({rawFile, [mode, skipDraw]})*: Returns a new *msPane*, derived from *zPane*, associated with the specified *rawFile*. The optional *mode* can be either “spectrum” or “chromatogram”, with “chromatogram” being the default. If *skipDraw* is true, then the *msPane* is not updated during the initialization.

*msPane.mode*: The current mode for the *msPane*.

*msPane:GetChromatogramTitle({[title, style]})*: Returns a string that will be the title of the chromatogram.

*msPane:GetMassRange({mass1, [mass2]})*: Returns a string specifying the mass range for the chromatogram. If mass2 is not specified, a unit resolution around mass1 is assumed.

*msPane:GetSpectrum({[spectrum, rawFile, scanNumber]})*: Returns a spectrum for plotting, with an additional entry with a *IsCentroid* key if the spectrum is centroid. If *spectrum* is specified, then it just returns it. If *rawFile* is not specified, then the current *rawFile* for the *msPane* is used. If scanNumber is not specified, then the first spectrum is used.

*msPane:GetSpectrumTitle({[title, rawFile]})*: Returns a title for the spectrum. If *title* is specified, it is just returned. If *rawFile* is specified, it is used instead of the current *rawFile* for the *msPane*.

*msPane:PlotCentroidSpectrum({spectrum})*: Plots the specified centroid spectrum in the *msPane*.

*msPane:PlotChromatogram({[chromatogram, rawFile, style]})*: Plots a chromatogram in the *msPane*. If *chromatogram* is specified, it will be plotted, otherwise a chromatogram will be retrieved. If *rawFile* is specified it will be used in place of the current *rawFile* of the *msPane*. If style is specified (“xic”, “bpc”, “tic”) it is used instead of the default of “tic”.

*msPane:PlotProfileSpectrum({spectrum})*: Plots the specified profile spectrum in the *msPane*.

*msPane:PlotSpectrum({[spectrum, scanNumber]})*: Plots a spectrum in the *msPane*. Will retrieve a spectrum if not specified with *spectrum*.

*msPane:SetRawFile(rawFile)*: Set the *rawFile* for the *msPane*. Can be used to change the current rawFile or specify it when not included during initialization.

*msPane:SetSpectrumMassRange(spectrum, filter)*: Adds entries *firstMass* and *lastMass* to the spectrum table based on the values in the *filter*.

**trendPane**

*trendPane({rawFile, [skipDraw]})*: Returns a new trend*Pane*, derived from *zPane*, associated with the specified *rawFile*. If *skipDraw* is true, then the trend*Pane* is not updated during the initialization.

*trendPane:Plot(label)*: Plots a trend line for the data specified by label. The label must exactly match the text characters used in the scan header, scan trailer, or tune report.

**templates**

*templates.templateList{}*: A Lua table list of available templates.

*templates.default*: The default template that will be used when a raw file is opened through the menu.

*templates.Register(template)*: Register the specified *template.*

*templates.SetDefault(template)*: Set the specified template to the default template.

**template**

*template.name*: The name of the template

*template.AddPages(notebook)*: A function that takes a specified *notebook* and adds desired pages to it. See templates.lua for example functions.