The MotionMonitor xGen Software Guide: Creating and Using Scripts

This document reviews the process of generating and using scripts in The MotionMonitor xGen. Scripting adds various ways in which The MotionMonitor xGen software can be customized, including controlling the workflow to ensure an accurate experimental setup and that data are collected in a consistent manner by all operators, to control the display of data and visualizations, and to add user interfaces.

Scripts perform a sequence of predetermined operations without additional manual intervention. The script syntax within The MotionMonitor xGen conforms to C syntax. Although advanced script routines can be generated by those with more programming experience, someone adept at creating formulas within Excel can also easily find success. Scripts can consist of a single line to reproduce a simple menu function or may contain various conditional statements and loops to perform more complicated and intricate operations. The following is meant to help provide an understanding of the capabilities for scripting within The MotionMonitor xGen software and provide guidance for best practices and getting started with using scripts.

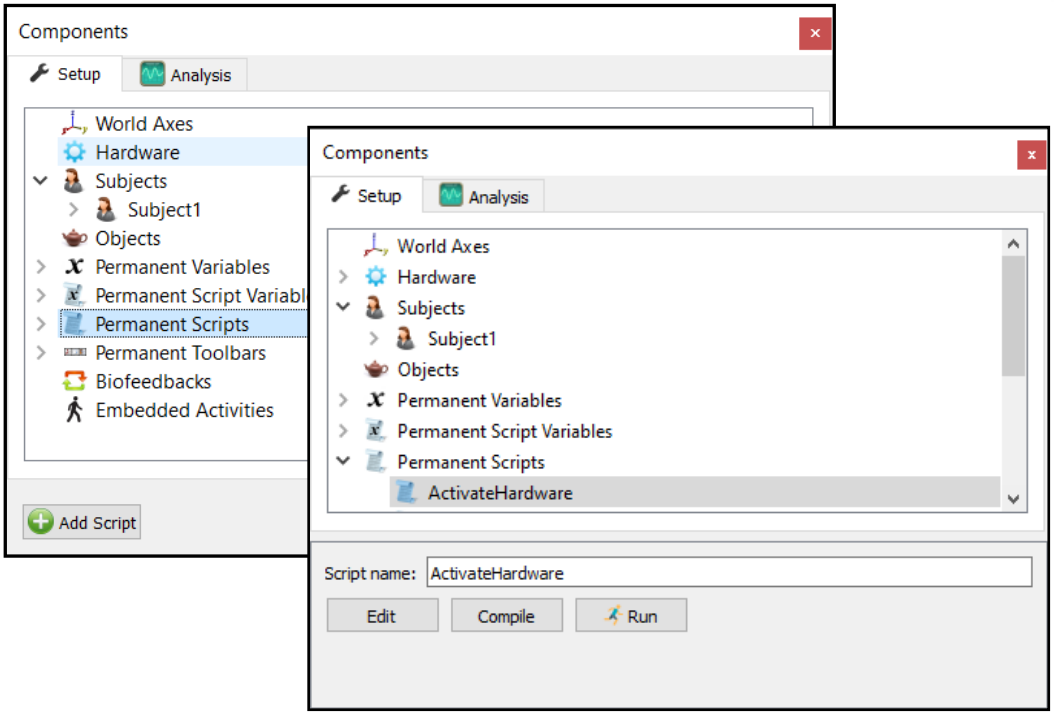
An example for creating a toolbar and associating a script with an icon will be described in a subsequent knowledge base article. Several scripting examples will also be provided in subsequent knowledge base articles. A complete list of variable and script operators and script functions, along with descriptions, can be found in Appendix A of The MotionMonitor xGen Elements User Guide.

Setup vs Analysis Scripts

The first thing you may notice when navigating through the Components window is that the Setup and Analysis tabs share variables, script variables, scripts and toolbars. So which tab is the best to use? Well, it depends. The setup side basically captures information at the time of a collection and is unchangeable thereafter. The analysis side is looking at data in the present and can be modified in post processing. Components on the Analysis side will be overwritten when loading an Analysis. The same functionalities can be performed from a script on the setup side as from a script on the Analysis side. However, a script can only utilize script variables from the same side of the Components window (i.e. a Setup script cannot reference an Analysis script variable). A script variable is a variable that gets their value from a script. Their values can be modified only through the execution of a script. For instance, a script could be used to obtain the body weight of an individual where the reading from the force plates is obtained and assigned to a script variable when the script is executed.

Scripts

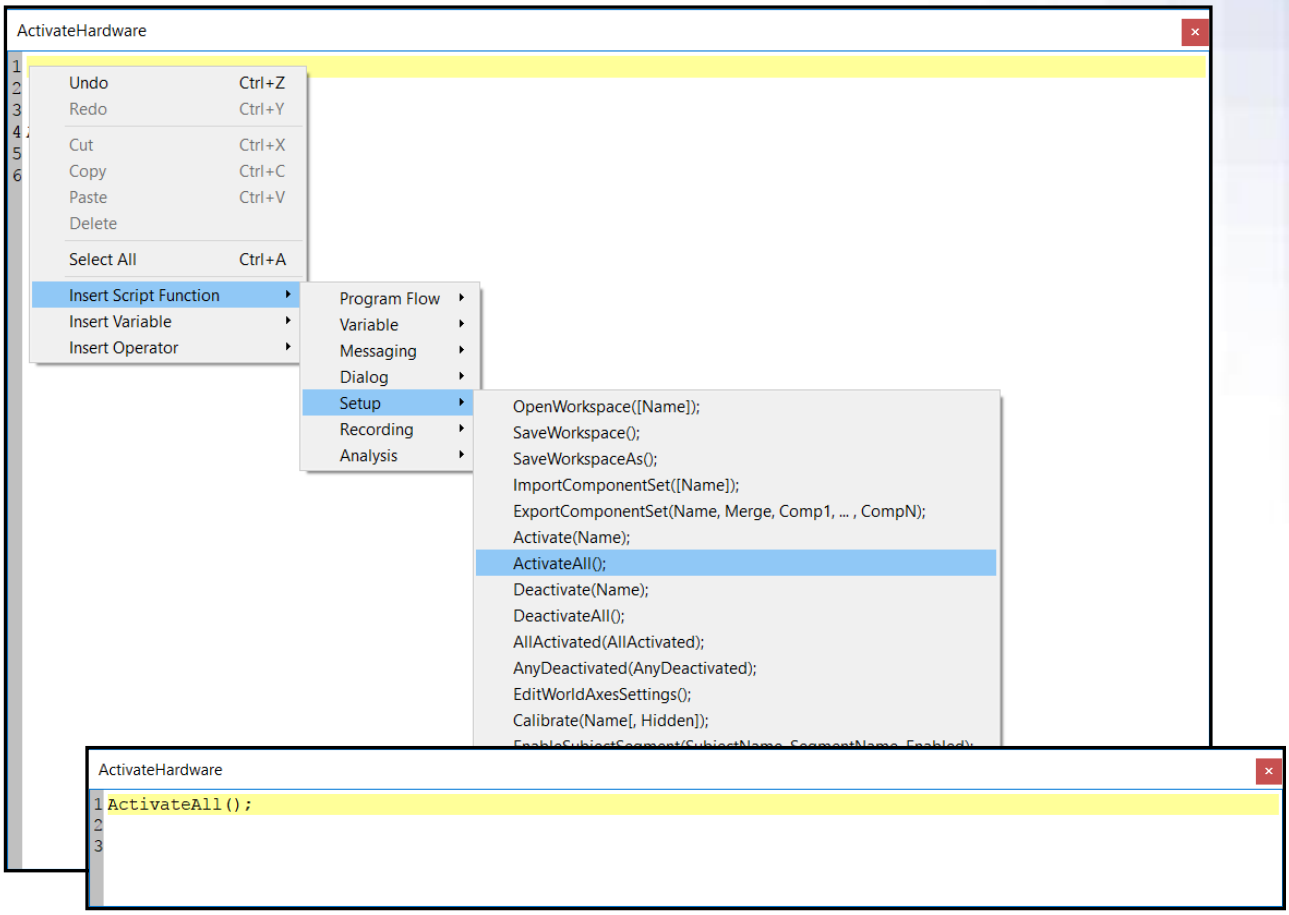
Adding a script and selecting the script node displays an edit field for naming the script and an “Edit” and “Run” button. The Run button will execute the script. Typically, a script is executed through the use of a Toolbar button or via another script, however the Run button is a handy way to test the script’s function as well as to highlight any errors. Errors can be corrected immediately, and the script run again for a fast, interactive debugging of the script. When creating and testing scripts, it is advisable to save the Workspace often.



Clicking the “Edit” button will open the script window for editing. Commands can be entered directly into the script window, however, right clicking in the window is the easiest way to call up code. This eliminates the need to be knowledgeable of syntax. Right clicking provides the choice of Script Functions, Variables or Operators. Script Functions include Program Flow, Variable, Messaging, Dialog, File, Setup, Recording and Analysis. Clicking the “Compile” button will compile the script, however the script will also be compiled the first time the script is executed. Most scripts will compile very quickly, but the compile button allows this process to be performed in advance, in case the script takes longer to compile or to identify potential errors. The “Run” button will execute the script.

Note that some script functions may be blocking while others are non-blocking, meaning that the script will wait for the function to execute before proceeding to the next line of the script or it will proceed immediately to the next line before the previous operation is completed, respectively. It is possible to use a non-blocking function in a manner where it may get called hundreds or thousands of times, thus overwhelming the system. If necessary, Messages or Delays can be added after a non-blocking function to avoid such issues. Please refer to Appendix A of The MotionMonitor xGen Elements User Guide for a list and description of all script operators.

Scripts can be a simple, single purpose command such as “ActivateAll()” hardware shown below.



Scripts can also be more complex consisting of //comments, Program Flow statements like if{} else{} and Messaging to control execution of the script. The “Insert Script Function” can be used together with “Insert Variable” and “Insert Operator” to completely eliminate the need to know code syntax.

ActivateHardware

```
1  // This script activates all hardware. If hardware fails to activate within 10 seconds the script will
   announce failure and return control. The script can also be used to deactivate the hardware if the the
   toolbar activate button is setup with two icons and the icon expression: if(IsAllActivated,1,0)
2
3  // The script requires the following variables in the Components|Setup tab
4  //   (boolean)    LOOPING
5  //   (boolean)    IsAllActivated
6  //   (time)       BeginTime
7
8  AllActivated(IsAllActivated);
9  if (IsAllActivated)
10 {
11         DeactivateAll();
12         IsAllActivated = FALSE;
13         OKMessage("Hardware has been deactivated.");
14         return;
15 }
16
```

When working with scripts, it may be useful to use the search function to find particular elements of interest. ctrl+f – allows for searching within a script for a particular string “”. The F3 key can then be used to skip to the next instance of the string in the script.

One important thing to point out regarding variable names and hardware names is that although changes to these names within the Components window will propagate throughout the software wherever that name is used in variables, graphs or reports, these changes will not propagate within scripts. So, for instance, if the name of a hardware device or variable is changed in the Components window, these updates will need to be manually made within all scripts.

For more information on setting up scripts for your applications, contact your client support engineer

Programming Style Recommendations

Below are some recommendations that we suggest following when creating variables and scripts in The MotionMonitor xGen, to help avoid logic errors and to keep your scripts organized and easy to troubleshoot.

Data within The MotionMonitor xGen can be represented in various forms and to potentially complicate things further, the same data from a hardware device can be represented in several ways (i.e. axes, vector, and scalar). So, it's important to enact a consistent nomenclature to avoid confusion. For instance, you could enact the following recommendations:

Boolean: Name should form a question when preceded by "Is" (for example, "ButtonEnabled" or "HardwareActivated").

Integers: If serving as an index into an array, name should end in "Index" (e.g., "SubjectIndex").

Scalars: Should contain enough information to uniquely identify it (e.g., "Sensor1X" or "Subject1Mass"). The last part should indicate the unit of measure, but for distances, masses, forces, and moments, this is could implicitly be meters, kilograms, Newtons, or Newton-meters.

Vectors: Should usually end in one of the following: Pos, Dir, Vel, Acc, Force, Moment.

Rotations: These are usually orientations and should end in "Ori".

Axes: Should end in "Axes", "Sensor", or "RigidBody". For relative axis systems, it could be of the form "Sensor1RelSensor2".

Time: Should end in "Time".

Strings: If serving as a name, should end in "Name" (e.g., "SubjectName").

Colors: Should end in "Color".

Abbreviations should be used sparingly, apart from agreed-upon conventions like those above.

Variable names get the first letter of every word capitalized. Constants are all-caps.

Ultimately, variable names should accurately reflect the data. Misreading a variable can cause logic errors and make debugging difficult. For example, *IsActivateHardwareRunning,* can be confusing…does it mean the script is running or that the hardware is activated? Better would be: *IsActivateHardwareScriptRunning* or *IsHardwareActivated*.

Make your code easy to read by following a few simple recommendations.

Comment frequently. Best practice is to comment the processing before coding. This can make coding easier and can catch logic errors early.
Example:

```
//Create a control to display instructions
//Create a control grid for the pushbuttons
//Create the Main control grid
//Create and Run the dialog
```

Don't pack your code. It makes reading hard and syntax errors difficult to find. Indent consistently and line up brackets to simplify reading and debugging.

Comments can be added by entering 2 forward slashes at the start of a line ("//"). While troubleshooting, lines can also be commented out in this manner to prevent them from being executed when the script is run.

Don't Do:

```
if (IsCancelButtonPressed){CloseDialog("Dialog");DialogOpen = FALSE;}
```

Better:

```
if (IsCancelButtonPressed)
{
      CloseDialog("Dialog");
      DialogOpen = FALSE;
}
```

Keep a list of variables used in a script, commented out at the beginning.