Why Aren't You Using App Builder Already?

App Builder (AB) was introduced in JMP 10. In the beginning, it had its bumps and warts and was, arguably, a bit of a challenge to use. As a long time JSL scripter, I, too was reticent to use it and preferred to code everything from scratch, even dialog boxes. I got tired of the work needed to make small visual tweaks to my dialogs and custom report windows, so I gave AB a relook. Since then, I have not looked back. If it's worth an interface, I'm using AB.

In this document I go into greater detail about AB than was possible in the recorded session, focusing on details affecting objects and scripting in AB. Whether you're new to AB or have been using it for some time, you should find this information useful.

Note: The names of all objects in the Source panel, except Data Filter(Local), Column Switcher, and Filter Col Selector end with Box. This word will be omitted except for List Box to identify and distinguish a List Box Box from a List Box (H List Box and V List Box), Check Box, which is technically a Check Box Box, and where it may cause confusion (e.g., to distinguish a Data Table from the box that contains it).

Working with the Interface

The AB interface consists of four parts. The Sources panel on the left side contains the visual elements you'll be adding to your application. I consider these elements as serving one of three purposes: display, organization, and interactivity. While there is some overlap with certain items, each can be considered to have a primary purpose. A list of all objects and their roles is given in *Table 1*. Display elements are placeholders for specific information such as text, numbers, reports, data table columns, pictures, etc. Organizational objects are containers for items from any of the groups. They control how objects are displayed in a module window and relative to one another. Unlike Display objects, the emphasis is not on what is displayed, but how. Interactive items are for user input, so output can be specified or modified, modules launched, and scripts run.

Table 1 – Box Purposes

Display	Organization	Interactivity
Report	Data Filter Context	Data Filter Source
Data Table	Border	Data Filter (Local)
Graph	(H/V) Center	Column Switcher
Icon	If	Mouse Box
Matrix	Lineup	Number Col Edit
Pict	H/V List	String Col Edit
Scene	Outline	Button
Text	Panel	Check Box
Table	Scroll	Radio
Number Col	Sheet Panel	Spin
String Col	(H/V) Splitter	Col List
Col List (All)	Tab	Filter Col Selector
	Tab Page	Combo
	Spacer	List Box
		Number Edit

		Slider
		Text Edit

Six objects are greyed out and unselectable unless a data table is opened: Data Table, Data Filter(Local), Column Switcher, Col List Box, Col List Box(All), and Filter Col Selector.

The tabbed workspace in the center is where the work gets done. It starts with two tabs, a Module tab for the visual elements and a Scripts tab for the code. You can drag and drop items from the Sources panel into a module tab in the workspace. Once there, they can be arranged and rearranged interactively. A new module tab can be added for each window the application needs. Code for all modules, as well as any start-up application code, can be accessed using the Namespace combo box on the Scripts tab.

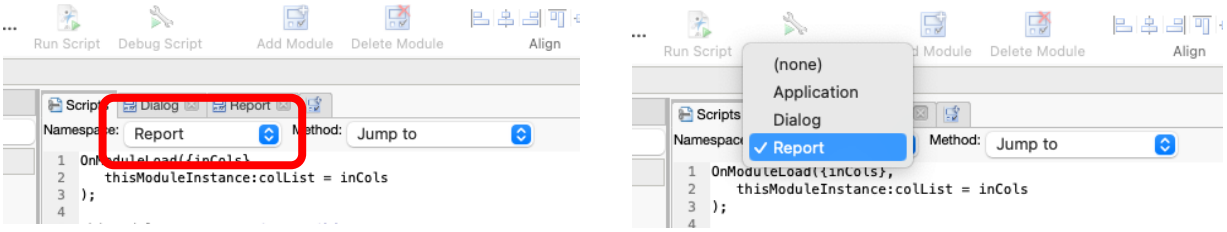

Figure 1 – Accessing module code

On the right, you'll find the Objects and Properties panels. They are similar in look and function to the Show Properties feature for report windows. Use these panels to easily set properties and make changes to objects in the workspace. You can only select one item at a time. Objects can't be moved in the Objects panel, only in the workspace area.

If you're new to AB, it helps to get feel for how things are selected and move around in the workspace. To add an item, drag it from the Sources panel. Once in the workspace, it can be easily duplicated through copy and paste. If you find yourself spending a lot of time setting object properties to get them to look exactly the way you want, it can be helpful to create a template application with the items you regularly use, set up the way you prefer. I often do this for object that have text element, since I regularly use a typeface and size different from the default. Creating a template saves time since a duplicated object retains all the properties of the copy. Objects can be copied from one application to another or one module into another.

To select an item, you can hover over it until you see a blue box highlighting it, then click on it or, using the arrow cursor and dragging a selection rectangle over one of its corners. Unlike PowerPoint, only part of the object needs to be inside the selection area. You can use this technique to select multiple objects as well. Alternatively, you can select multiple objects by holding down the shift key and clicking on them. As mentioned above, a single item can be selected using the Objects panel.

When working with organizational containers you have two choices. You can create the container first, then drop objects into it. Or you can select one or more existing items to be placed into the container, right click, and select Add Container. This works for everything except for If and Lineup. I tend to use this approach more frequently since I find it easier and quicker, particularly when I am working with multiple items. If two or more containers are nested, you can remove any container except the innermost by selecting it, right clicking, and choosing Remove Container from the pop-up menu. This is a case where selection from the Objects panel is helpful to ensure you've selected the correct container. Selecting an item from the Objects panel rather than interactively is also handy when it winds up behind other objects, if you can't find it in the

workspace, or even if you're just having a difficult time selecting it to begin with (maybe it's very small and is placed next to a lot of other small items). When a container is selected, all items in it move, are copied, and deleted, as a unit. You can also change the container type using the right click menu (Change Container).

While moving objects is straightforward, resizing them is a bit trickier and more limited. Except for the items in *Table 2*, objects are sized relative to their contents. For these items, sizing can be done interactively or by changing one of their sizing properties, except Pict, which can only be sized interactively once an image is dropped into it. All size values are specified in pixels. For Text and Text Edit, Width controls the space allocated for the box. A value of -1 allows it to resize according to its contents. Wrap specifies the number of pixels at which text wraps to a new line.

Table 2 – Objects with explicit sizing properties

Object	Sizing Property
Scroll	Width, Height
Spacer	Width, Height
Graph	Width*, Height*
Pict	Interactive only
Scene	Width, Height
Text	Width, Wrap
Col List	Size: X, Y
Filter Col Selector	Size: X, Y
List Box	Size: X, Y
Slider	Width
Text Edit	Width, Wrap

** Correspond to graphing area and not box size*

All items in *Table 2* possess the User Resizable property except Spacer, Graph, Scene, Text, and Slider. This allows the X and/or Y size values to be locked by the application.

Common Properties

The 17 properties common to (nearly) all objects are show in *Table 3*. Data Filter and Column Switcher only possess the first two and Graph does not have Padding, Border, and Margin properties. Stretching consists of six properties as indicated in the table.

Table 3 – Properties common to all objects

Property	Comment
Variable Name	String. JSL name for an object. Scoped to thisModuleInstance namespace for the module in which the object is defined or to thisApplication in the case of modules and data tables. Defaults to the box name plus an added integer (depending on the number of similar boxes used) or to Module or DataTable with an added number.
Position (X, Y)	Integer Ordered Pair. Pixel distance from upper left corner of the application window to the object.

Background Color	Selection. The color of the background inside the object. Default: None
Text Color	Selection. The color of any text that is part of the object. Several objects have this property but contain no text. Default: None
Visibility	Visible, Hidden, or Collapse. Hidden retains object spacing, but the object is invisible. Collapse collapses all spacing and makes the object invisible. Spacing for all objects inside a container are also collapsed. Default: Visible
Padding	Integer. Added space (in pixels) inside the object border. Default: 0
Border	On/off. Turns on or off frame side inside the object. Positive values for on, everything else is off. Four possible locations: top, bottom, left, right (shown as arrows). Default: off.
Margin	Integer. Added space (in pixels) outside the object border. Default: 0
Vertical Alignment	Default, Top, Center, or Bottom. Vertical positioning of an object inside a V List when there are two or more objects.
Horizontal Alignment	Default, Left, Center, or Right. Horizontal positioning of an object inside an H List when there are two or more objects.
Stretching	Six properties: Min Size (X and Y). Integer. Set to initial size of object. Max Size (X and Y). Integer. Set to initial size of object. Stretch: X – Neutral, Off, Window, or Fill. Default: Neutral Stretch: Y – Neutral, Off, Window, or Fill. Default: Neutral
Enabled	On/off. Unchecking greys out and disables the object making it unselectable. Default: on.

Border and Graph have the Frame property, where space can be added inside the frame area. The check box in the properties list allows the addition of a frame (a second frame in the case of Border) inside the padding but outside of the frame area. *Figure 2* illustrates the relationship between Padding, Border, Margin, and Frame properties.

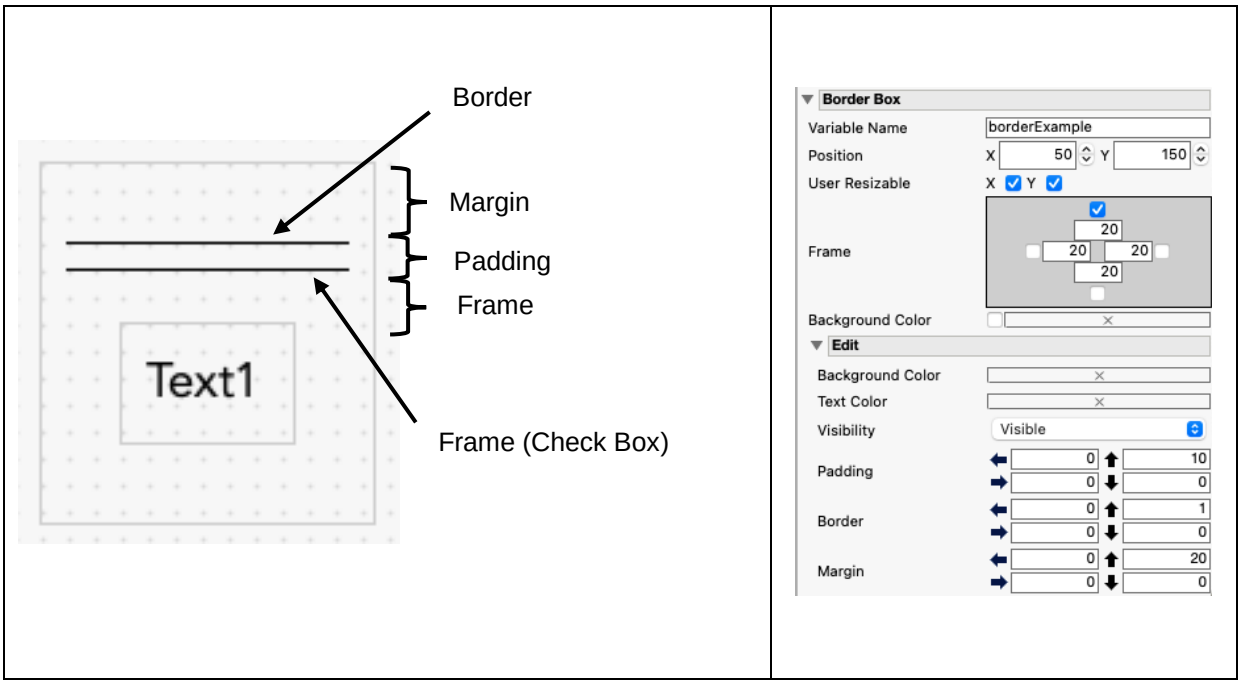

Figure 2 – Border Box properties illustrated

Several objects can contain text in a title, as part of their contents, or both. Seven allow you to control the font typeface and size using the Base Font, Font, and Font Scale properties. Base Font corresponds to typeface, style, and size given in the User Preferences (e.g., Heading, Title, Axis, etc.). Font is a selection button allowing the choice of typeface, size, style, and is operating system dependent. Font Scale takes a numeric value that acts as a multiplier for the font size. *Table 4* provides a list of these objects along with their text related properties.

Table 4 – Objects with text properties

Object	Main Property	Other Properties
Outline	Title	Base Font Font Font Scale Allow title wrapping (on/off)
Panel	Title	
Sheet Panel	Title	
Tab Page	Title	Base Font Font Font Scale
Icon	Title	
Number Col Number Col Edit	Title, Items	
String Col String Col Edit	Title, Items	
Button	Title	

Check Box	Items	
Radio	Items	
Combo	Items	
List Box	Items	Base Font Font Font Scale
Text	Text	Base Font Font Font Scale Bullet (on/off), Rotation (Horizontal, Left, Right)
Col List	Box contents	Base Font Font Font Scale
Col List (all) Filter Col Selector	Box contents	Base Font Font Font Scale
Number Edit	Number	
Text Edit	Text	Base Font Font Font Scale Bullet (on/off), Justification (Left, Center, Right) Rotation (Horizontal, Left, Right) PW Style (on/off) – On: typed characters are masked Locked – Off: Text is not editable

Working with Tab and Tab Page Boxes

Tab acts as a container for Tab Page. If an object other than Tab Page is dropped into an empty Tab a Tab Page is automatically generated. In this sense, use of a Tab Page is superfluous, except for use as a stand-alone object. Properties associated with Tab are general and apply to all tabs contained within it. To create additional tabs in Tab, right click one of the Tab Pages and select Tab > Insert Before or Tab > Insert After. You can also drag and drop a Tab Page to the left of the first or right of the last existing Tab Page. This approach is less flexible in that a new tab can only be added as the first or last tab. A tab is deleted by selecting the Tab Page, right clicking, and choosing Tab > Delete

Working with Filters

Filtering requires two elements, Data Filter Context and either Data Filter(Local) or Data Filter Source, the latter being used when visual elements are used to filter. All objects, including any reports to be filtered, must be contained inside Data Filter Context. Since Data Filter Context only take a single item, an organizational box must be used to wrap multiple objects. Filtered objects must be outside Data Filter Source to work correctly. Data Filter Source can take multiple object on which to filter. Like Data Filter Context it only takes a single (uncontained) object. Filtering can be done with either Data Filter(Local) or Data Filter Source, but not both.

Using AB out of the box, no scripting – Parameterized Applications

Applications where the number and modeling types of the column do not vary can be easily created without additional scripting. Parameterized applications generate a built-in dialog box so users can select columns for a report. To parameterize an application, the columns used in each JMP Platform report window are assigned a variable name. Most JMP platforms require a different variable for each column used, however, some platforms (mostly those under Analyze > Multivariate Methods and Graph) will take a single variable for a list of columns. Variable names are entered in the Properties panel under Roles.

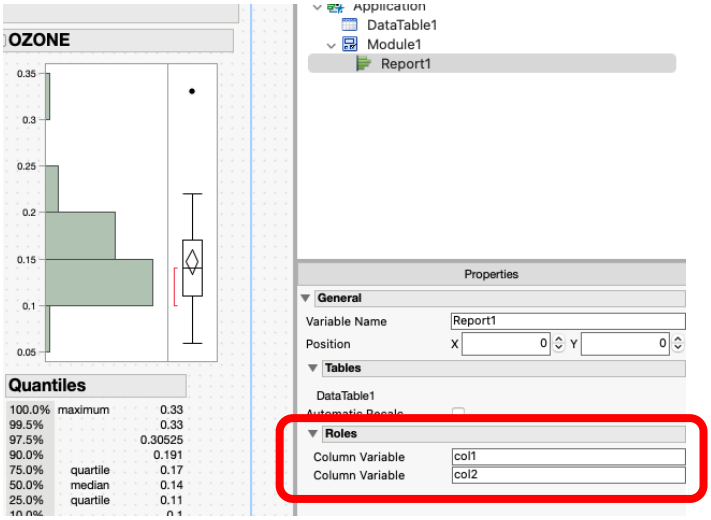

Figure 3 – Parameterized column names

To use the same column for multiple reports, use the same variable name. *Figure 4* below shows an example where the same two columns are used in two different platforms.

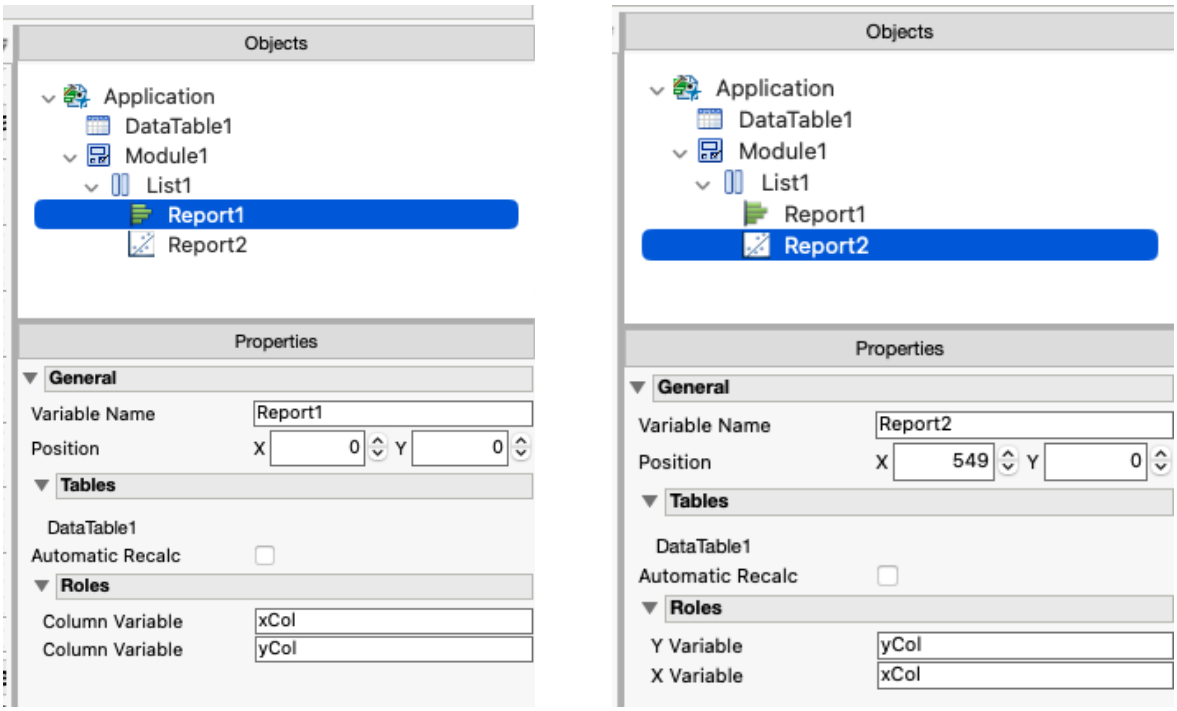

Figure 4 – Using the same columns for different reports

The names appearing in the auto generated dialog box for parameterized variables can be changed by selecting Applications in the Objects panel and supplying different values for the variables under Parameters.

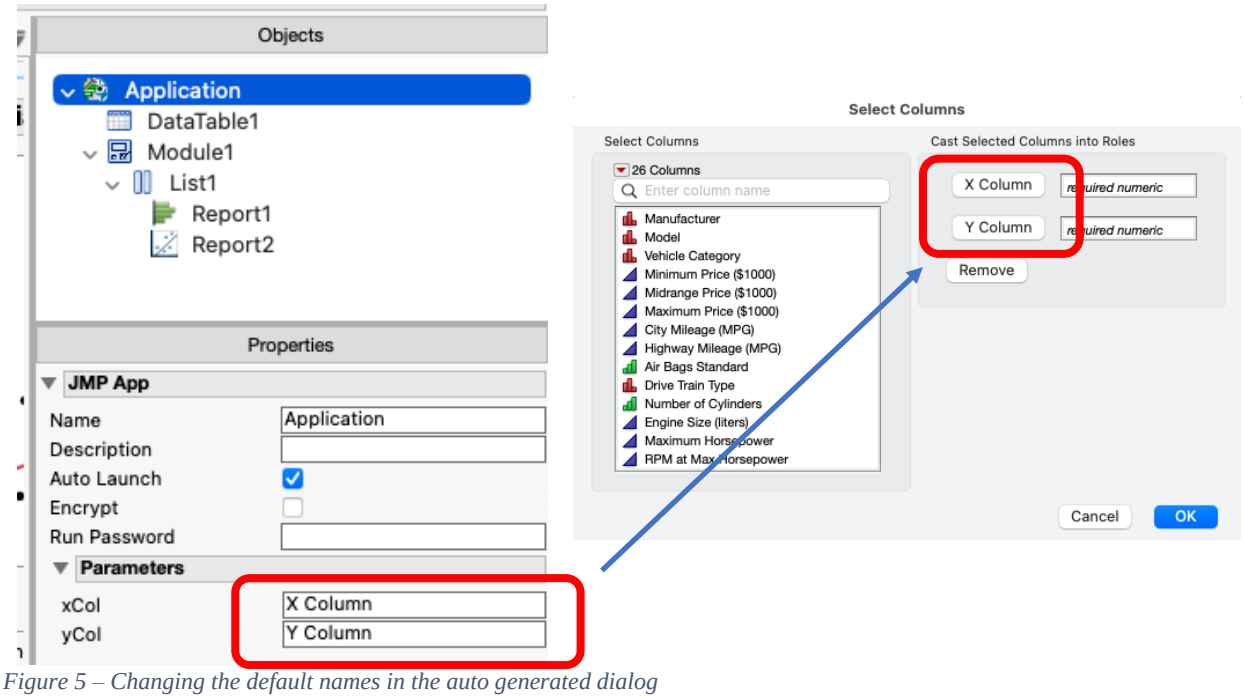

Figure 5 – Changing the default names in the auto generated dialog

You'll need to have any report element sized and set-up the way you want it to look before dropping it in the AB. Once dropped into the workspace, closing outlines, resizing graphs, running hotspot options, etc., will not be possible. Elements in H List Box will be set to the same vertical size, you

can't make one taller than the other inside the box. Likewise for sizing elements horizontally in a V List Box.

Writing Code – Namespaces

Understanding how namespaces work will make it easier to create robust applications. If you're new to the idea of scoping and namespaces, scope dictates how and where a variable is defined in the code. A namespace can be used to limit the scope of a variable to that namespace. This helps you to avoid situations where the contents of a variable are overwritten because it was inadvertently defined somewhere else in the same namespace. This can easily happen when you write code and scope all the variables to the global namespace. To learn more about scoping and namespaces, see Advanced Scoping and Namespaces in Chapter 8 of the Scripting Guide.

An AB application has two types of namespaces, both are anonymous but have reserved names to be used inside the application code. The `thisApplication` namespace is created when the application is launched and is scoped to the entire application. A locally scoped namespace is created each time a module is launched. The keyword `thisModuleInstance` is used to reference items in a specific module. If there is more than one module, there will be more than one `thisModuleInstance` namespace. In these situations `thisModuleInstance` references the module in which it appears. Variables declared in a module don't require the `thisModuleInstance` namespace qualifier. It comes in handy, though, to avoid ambiguity if variables with different scopes share the same name. There are two cases where `thisModuleInstance` is necessary. First, when instantiating the objects associated with the module. To do this, the code

```
thisModuleInstance << Create Objects;
```

is added to the module script by default. The second place is to reference the window created by module instantiation:

```
thisModuleInstance << Get Box;
```

It's important to understand that a JMP App Module Instance is different from a JMP App Module, the former being a concrete instance of the later.

Writing Code for User Interactions

A key advantage of the AB is that it makes creating and maintaining user dialogs much simpler. As mentioned previously, most properties can be set interactively in the Properties panel and visual organization of the dialog is also done interactively so you can see how it looks without having to run code. You can still message objects in the code if a property isn't listed (e.g., as of JMP 17.2 the White Box Style property doesn't appear in the Properties panel for Text Edit) or if it needs to be set at runtime. Code for this is put under the Scripts tab in the namespace associated with the module where the object appears. Any code referencing module objects must appear after they are created, that is, after

```
thisModuleInstance << Create Objects;
```

Table 5 gives the objects with scriptable properties and their names. Code associated with their behavior can either go in the Properties panel (see *Figure 6*) or in corresponding namespace section under the Scripts tab (see *Figure 7*). Scripts created in Properties panel are considered anonymous, since they need not be named and are only available to the object that defines them. Anonymous scripts make the code less cluttered. The drawback is that you must remember to look

in the Properties panel to find them. If the Scripts tab is used, the variable name of the function or expression block associated with the executed code must be put in the text entry field for the script in the Properties panel.

Table 5 – Objects with scriptable properties

Object	Script Property
Mouse	Click, Mark, Track, Drag Begin, Drag End, Drop Track, Drop Comment
Tab	Tab Close, Tab New, Select
Graph	Script
Table	Row Change
Number Col Edit	Script
String Col Edit	Script
Button	Press
Check Box	Change
Radio	Select
Spin	Press
Col List	Select
Col List (all) Filter Col Selector	Select
Combo	Select
List Box	Select
Number Edit	Script, Number Changed
Slider	Move
Text Edit	Script, Text Changed

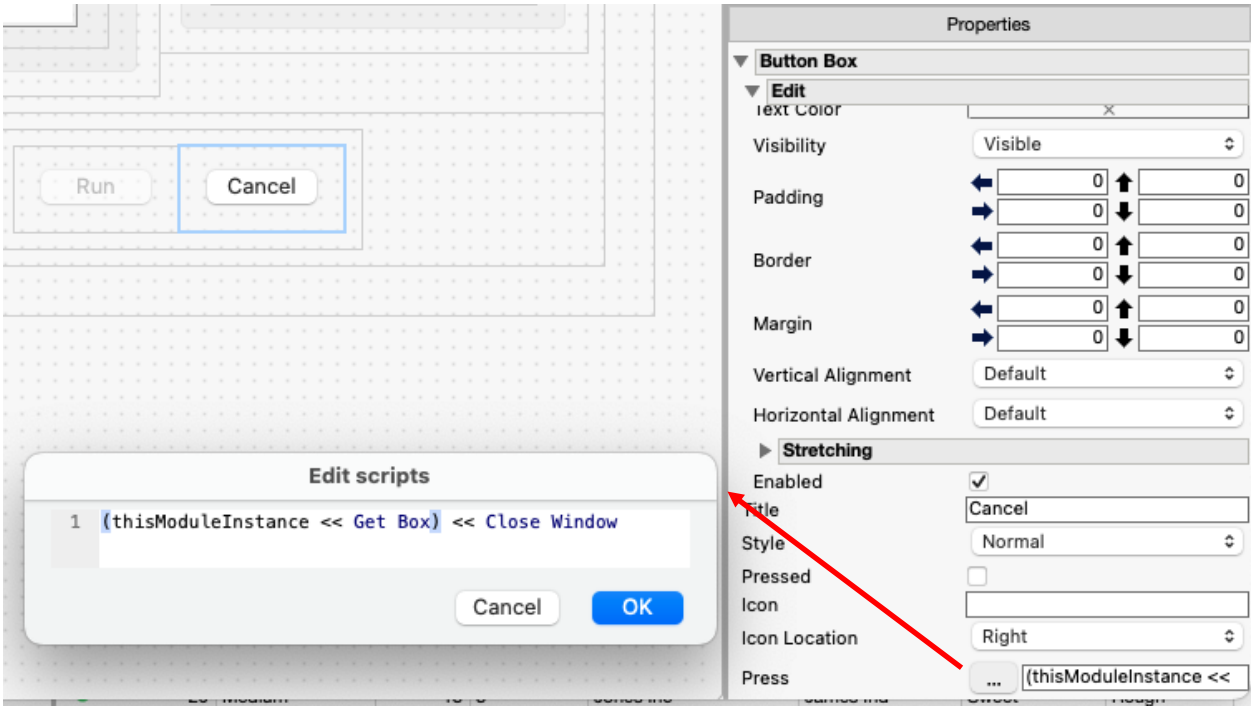

Figure 6 – Anonymous script located in property panel. Click on three-dot button to launch Edit scripts window or type directly in text field.

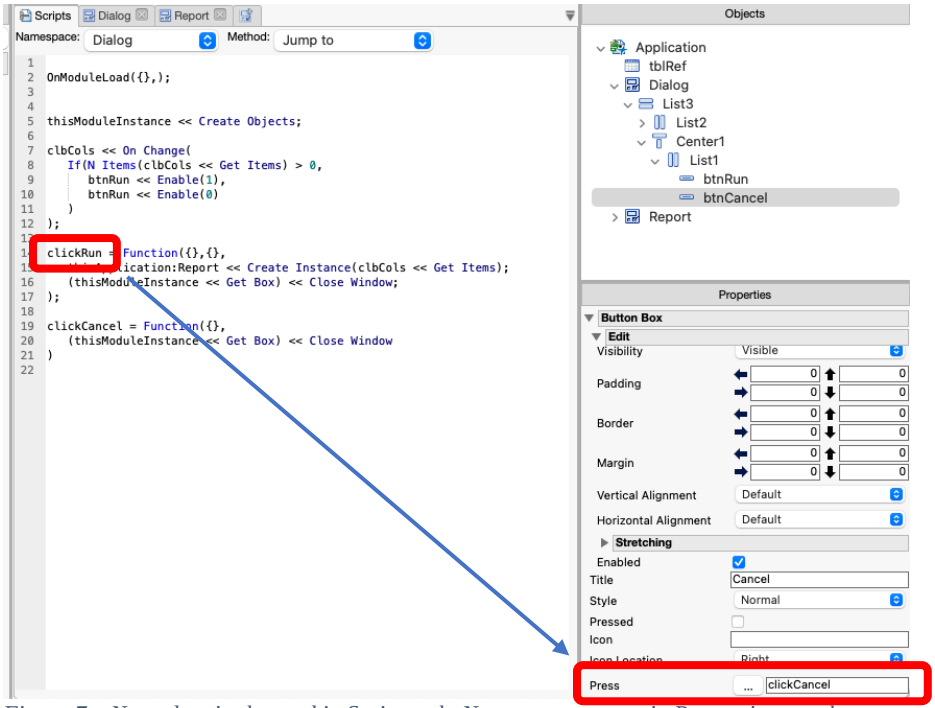

Figure 7 – Named script located in Scripts tab. Name must appear in Properties panel.

To quickly add a script to the proper area in the Scripts tab, right click the object and scroll to Scripts at the bottom of the menu. Selecting a Script property (e.g., Press for a Button) automatically generates a `Function` script. The function name is a concatenation of the object's variable name and the name of the script property. All scripts contain the optional argument

`thisBox`, which references the object associated with the function. *Table 6* gives a list of additional arguments for those objects with script properties containing more than one argument.

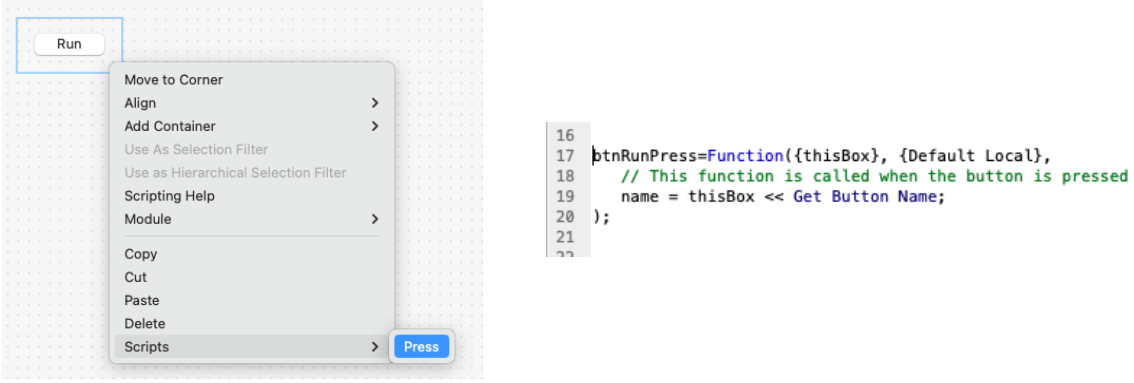

Figure 8 – Creating an object script from the workspace

Table 6 – Object with scripts containing more than one argument

Object	Script	Additional Arguments
Mouse	Click	clickpt, event
Mouse	Track	clickpt
Mouse	Drag Begin	clickpt
Mouse	Drag End	clickpt, how
Mouse	Drop Track	clickpt
Number Col Edit	Script	which
String Col Edit	Script	which
Check Box	Change	index
Spin	Press	value
Number Edit	Number Changed	numEditValue
Text Edit	Text Changed	text

Argument	Description
clickpt	X,Y coordinate of mouse location relative to the upper left of the application window. Can take negative values when cursor is outside of the window.
event	Pressed – mouse button pressed Moved – mouse moved, button still pressed Ticked – button pressed but mouse not moving
how	copy –object should be copied to new location move – object should be cleared from the source and stored to the new location ignore – drag operation canceled
which	Script currently does not work
index	1-based index of item changed
value	1 = up arrow clicked, -1 = down arrow clicked
numEditValue	Current value of number. Update does not require user to press Enter
text	Current value of text. Update does not require user to press Enter

If a function already exists for an object script, the right click menu option will have a check and the name of the script or Send for anonymous scripts.

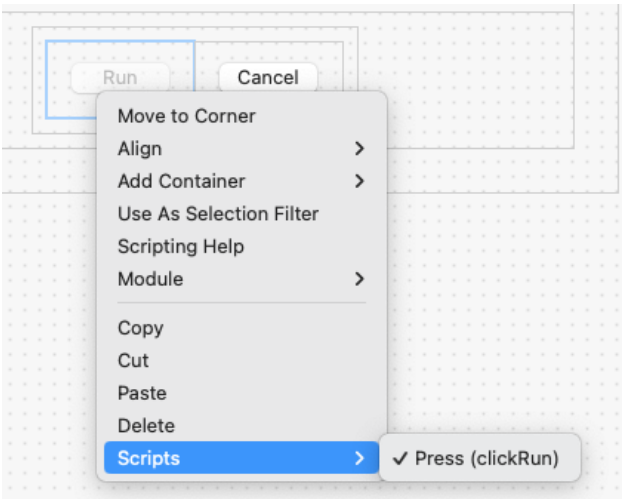

Figure 9– Object with existing script

Working with Multiple Modules

AB has different types of modules with which to work: Dialog, Dialog with Menu (has a menu at top of window, Windows only), Modal Dialog, Launcher, Report (can be saved with Save or Save As), Display Box (does not create its own window but contains contents to be used by other modules or windows). The Auto Launch property controls whether the module is launched at application start. It is on by default. To launch a module where Auto Launch is set to off you can use the code

moduleRef `<< Create Instance(`*moduleArgs*`);`

where *moduleRef* is a reference to the module and *moduleArgs* is one or more comma separated values to be passed to the module. Values passed to a module are received in the `OnModuleLoad` function, e.g.,

```
OnModuleLoad({arg1,arg2,..., argN},
   thisModuleInstance:local1 = arg1;
   thisModuleInstance:local2 = arg2;
        .
        .
        .
   thisModuleInstance:localN = argN;
);
```

where `local1`, `local2`, … `localN` are scoped to the module in which they appear, i.e., to the thisModuleInstance namespace. The qualifier `thisModuleInstance` is optional. There are no additional features providing communication between two modules, e.g., a publisher-subscriber pattern implementation, outside of functionality already present in JSL.

Working with Data Tables

When working with applications requiring a data table, it is helpful to work with a table that will not change location or name. Regardless of the table being used for the analysis, AB will require there to be some table open. Data tables from the sample data directory provide good options since they rarely change names, and the directory location can be accessed with the JMP variable `$Sample_Data`.

Running, Debugging, and Deploying Applications

Testing and debugging applications must be done from the red hotspot button to the left of Application Builder. The simplest way to deploy an applications is to save it to an Add-In. The script can also be saved to a custom menu item. To do this, change `Edit` on the last line of the script to `Run` prior to saving it. In general, the script generated by Save Script to Script Window should be left unaltered.