using event driven model with Mathematica Manipulate[] programming

Nasser M. Abbasi

July 2, 2015   Compiled on July 9, 2025 at 3:21am

Contents

1 Introduction

Mathematica's `Manipulate[]` function, introduced in version 6 is a powerful and easy to use function making it easy to develop a user interface (UI) application.

Manipulate basic logic flow is simple: It evaluates its expression and displays the result of this evaluation. It does this each and every time any one of the control variables changes value.

When the variable changes, the expression is revaluated automatically   ①

Manipulate[**expression**, variable]   ②

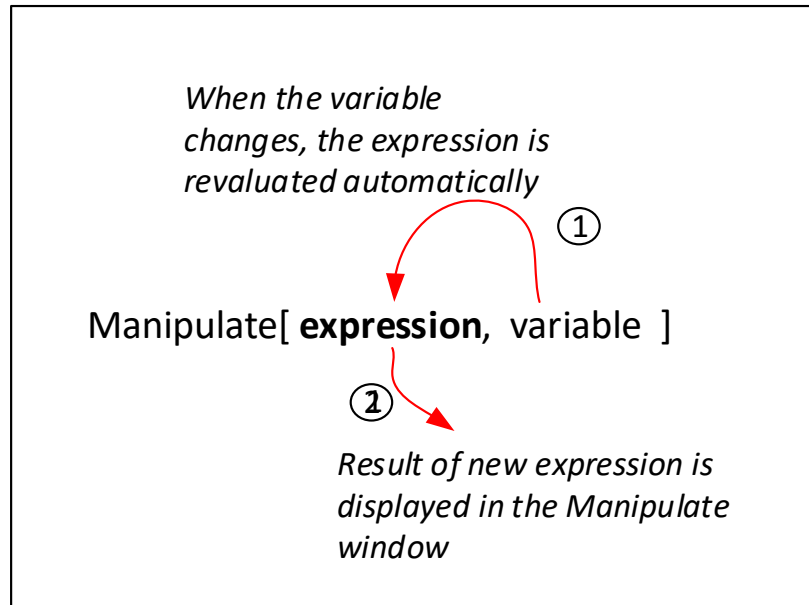*Result of new expression is displayed in the Manipulate window*

Figure 1: Manipulate logic

The Manipulate expression can be as complicated as we make it. It is in general a function of the control variables. Therefore each time any control variable changes value by the use of a slider or other UI components, the effect of this change is immediately reflected in the display of Manipulate. Manipulate evaluates the expression using the new value of the variable and updates the display automatically.

The following diagram helps illustrate the above logic flow

Manipulate[a + b,

And the result will be displayed here

③

{{a, 0, "a="}, 0, 10, 1},
{{b, 0, "b="}, 0, 10, 1}
]

② Then Manipulate will evaluate this expression

a=

b=

3

Each time you move any one of the sliders

①

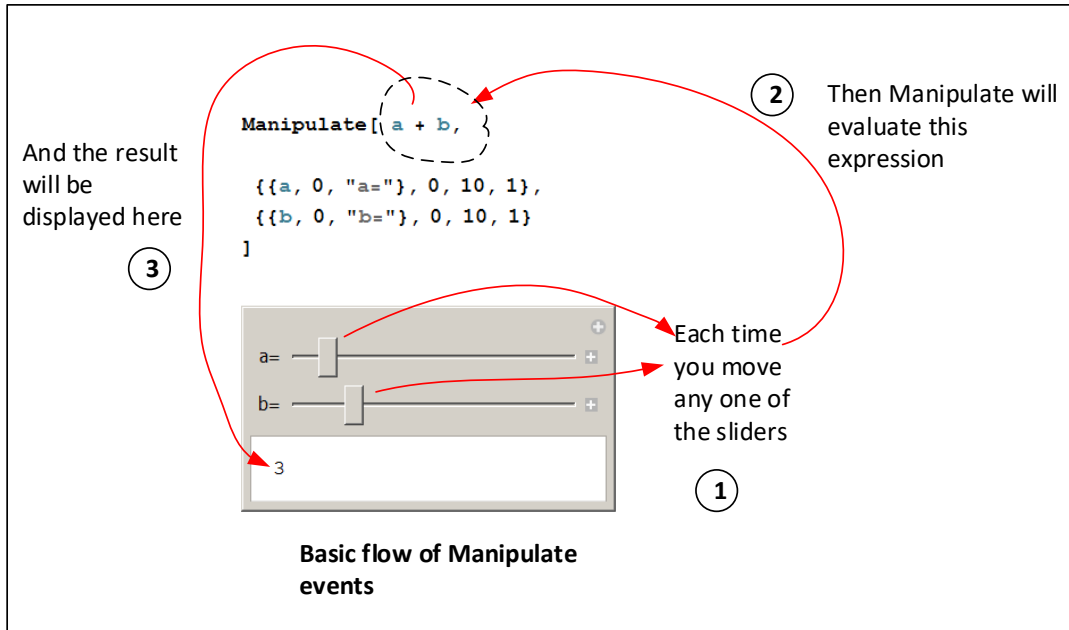**Basic flow of Manipulate events**

Figure 2: Expanded Manipulate logic

Because the evaluation of the expression occurs when any one of the control variable changes, this might not be a very efficient use of resources.

Instead, we would like to know which specific variable changed in order to customise which part of the expression to evaluate. In the following section, we show a model to use to detect which control variable changed and to only perform an action based on that specific change.

2 Using state machine for Manipulate expression

By running the Manipulate expression logic as a state machine which detects the specific variable which changed and updates an expression based on that event only we can make better use of resources.

The event will be the change of a specific control variable. The state of the machine is stored in a Manipulate control of type `None`

The state of the machine consists of the event name and any other state information needed.

This setup is similar to how UI programming is done in traditional setting where a specific event triggers a callback associated with the event.

To simulate event driven UI, an inner dynamic is added, with a `Refresh` option, inside the Manipulate expression with its own `TrackedSymbols` option.

Each control variable will have its own `Dynamic` with a `TrackedSymbols` for its own variable.

Inside this Dynamic, the event control variable is set to indicate which control variable has just triggered.

Since an inner Dynamic can have its own `TrackedSymbols`, we have effectively moved the task of detecting the change of the control variables from Manipulate down to the inner Dynamics.

The event control variable of type `None` will be used by the state machine to check which event has just occurred and then perform an action based on the event.

After the action is completed, the event is reset to special reset value.

The above logic repeats again each time a control variable changes value. Refresh is needed to be used inside the inner `Dynamic[]` and within the `Row[]` construct as will be shown below.

The above method is illustrated with examples, starting with a very simple annotated example, showing how to program a Manipulate which displays the name of the variable which was last changed using a slider.

In these examples, a string is used to indicate the event that occurred, but this can be changed, and other type of values can be used.
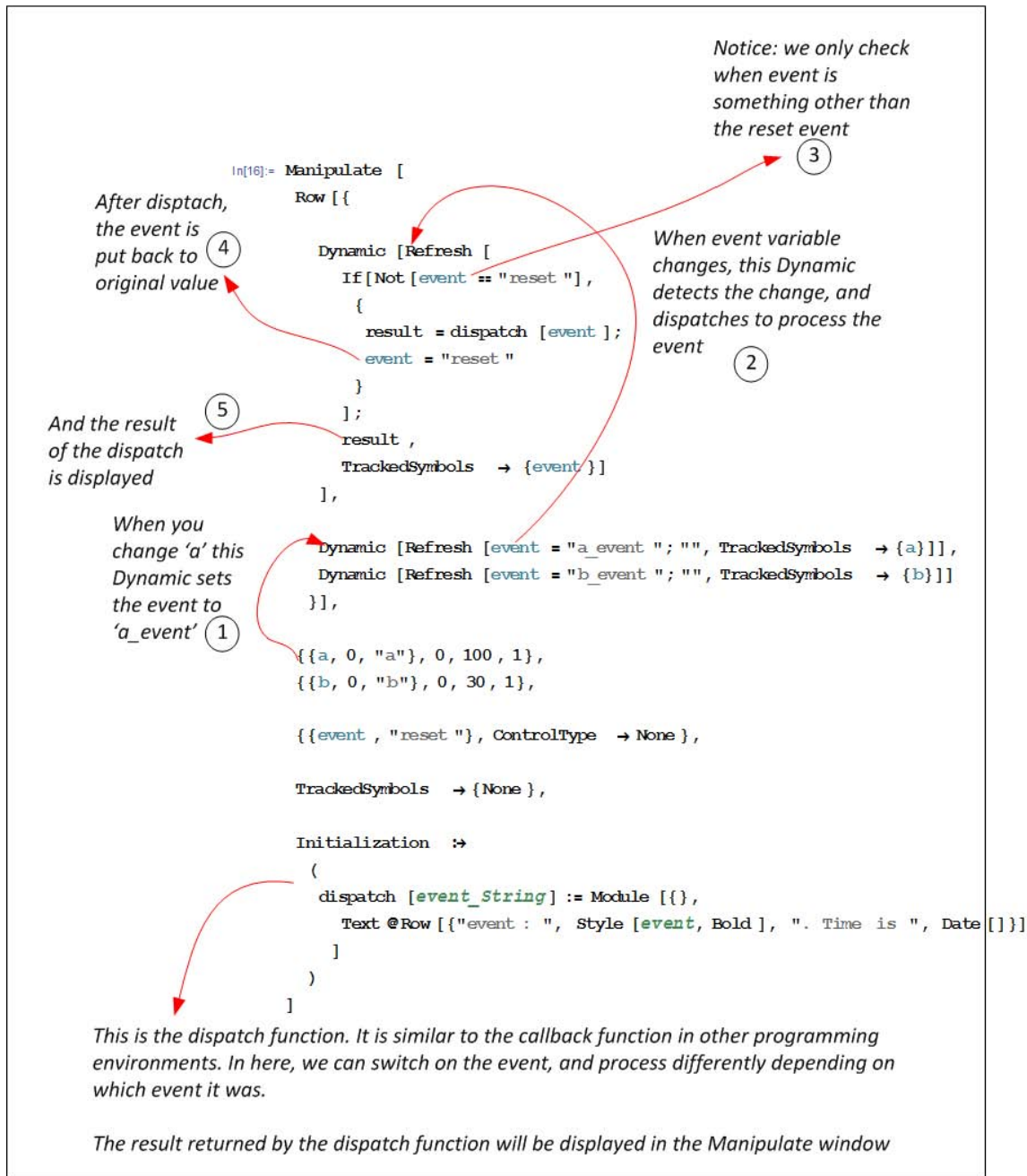
Figure 3: Manipulate code

When moving the 'a' slider in the above example, the following message will be displayed on the Manipulate window
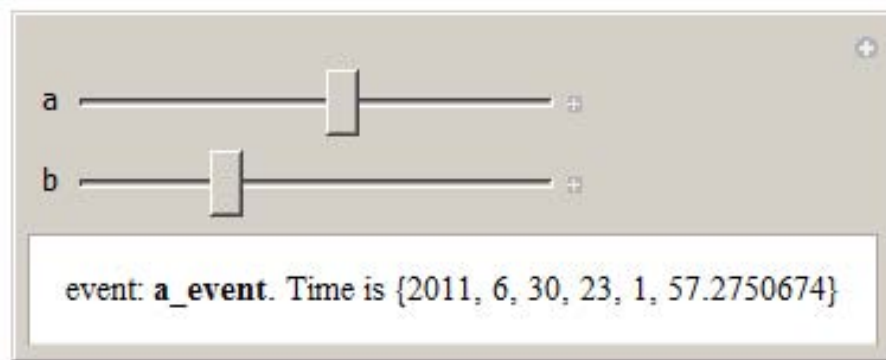
Figure 4: Manipulate output

A similar display results for `'b'` when its value changes.

To illustrate the benefit of this approach, we will now use it to show how to numerically solve an ODE. We will have one control variable for the initial condition, where each time the initial condition variable is changed, `NDSolve` have to be called to obtain a new numerical solution. In addition, we will have a second variable to indicate which color to use to plot the solution with.

Clearly, there is no need to call `NDSolve` when the color variable is changed as the initial condition did not change.

Using event driven UI programming we are now able to do this. The following digram shows the solution to the above. Notice that the solution is saved in a Manipulate control variable as well as the event name.
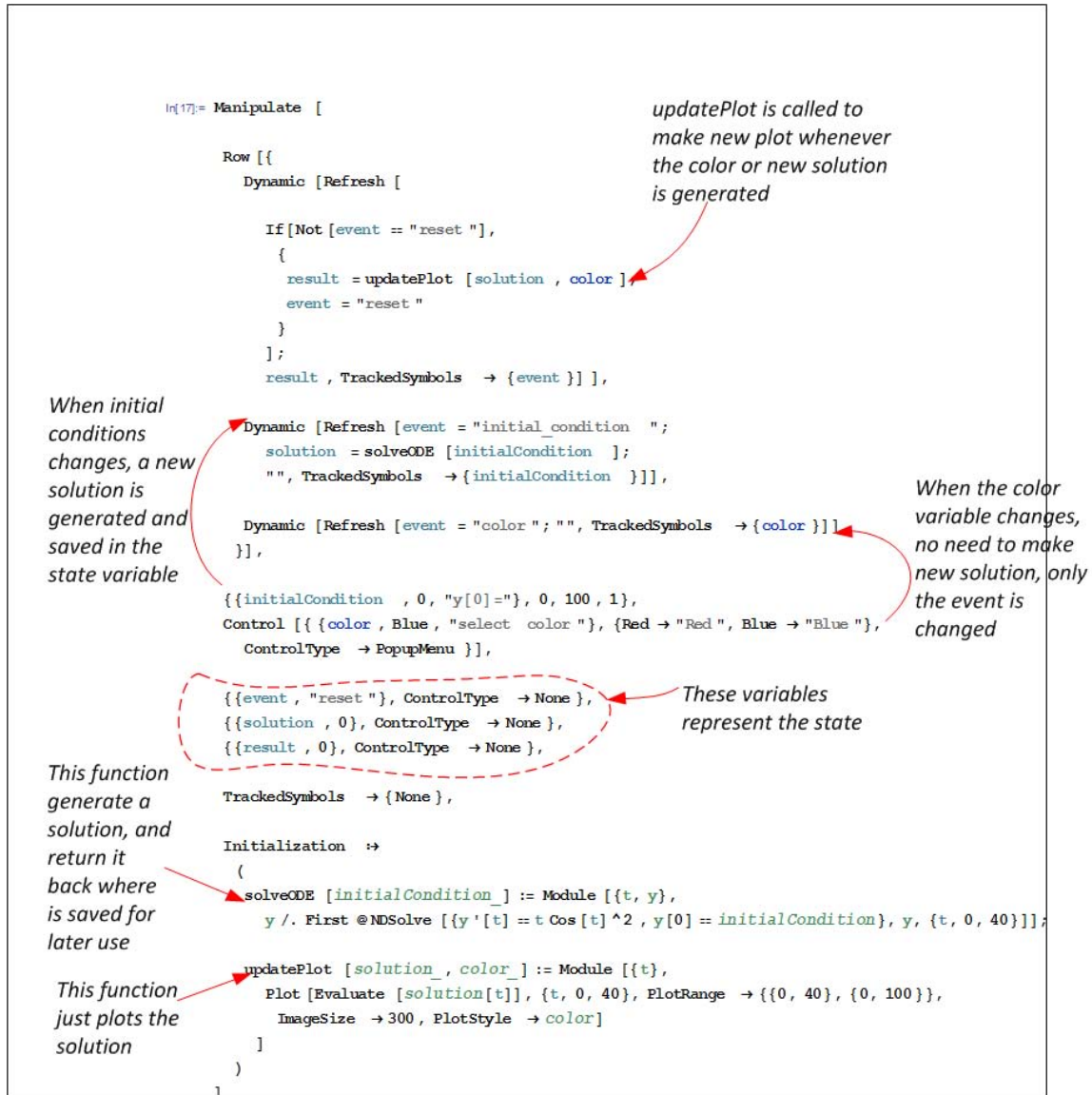
Figure 5: Manipulate code

Now the above solution is compared to the solution without using event driven model as is normally done.
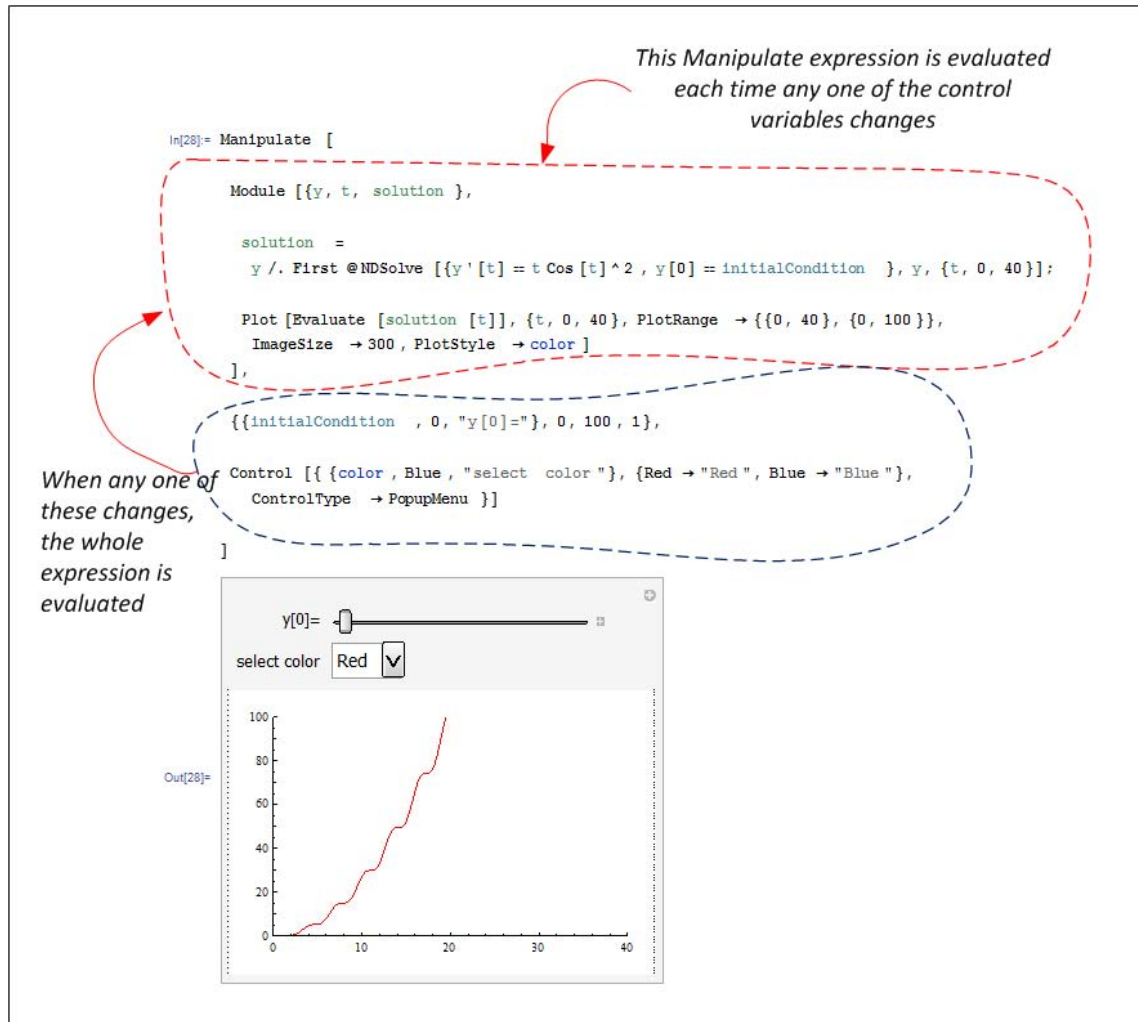
Figure 6: Manipulate code

We notice that the solution above is much simpler, however it is not efficient on resources. The whole expression is evaluated whenever any one of the variables changes value. In other words, NDSolve is called each time the color variable has changed, which is not required.

3   Conclusion

Mathematica Manipulate can be used to build UI very quickly. Using event driven model can improve the efficiency of using Manipulate by giving the user more control of what action to do based on which variable changes.

Using event driven UI is much more efficient on resources, but it requires more logic to be added to the Manipulate expression.

Using event driven UI programming makes using Manipulate more similar to the event/callback model used by other UI systems where the event is viewed as the change of a specific control variable, and the callback function is the inner Dynamic which detects the change in that specific variable.

4   References

1. Mathematica documentation on Manipulate and Dynamics
2. Useful discussions with John Fultz (Wolfram research) on the subject. Internet newsgroup posting