

Guided Tour: Macros and Events

(estimated time: 30 minutes)

Overview

This document describes how macros and events can be used to automate tasks in a database. A **macro** is a named set of one or more actions. An **event** is an action recognized by an object for which you can define a response. By placing the macro's name as the value for an event property, the macro runs whenever the event occurs. In other words, the **macro** is the **response** to the event.

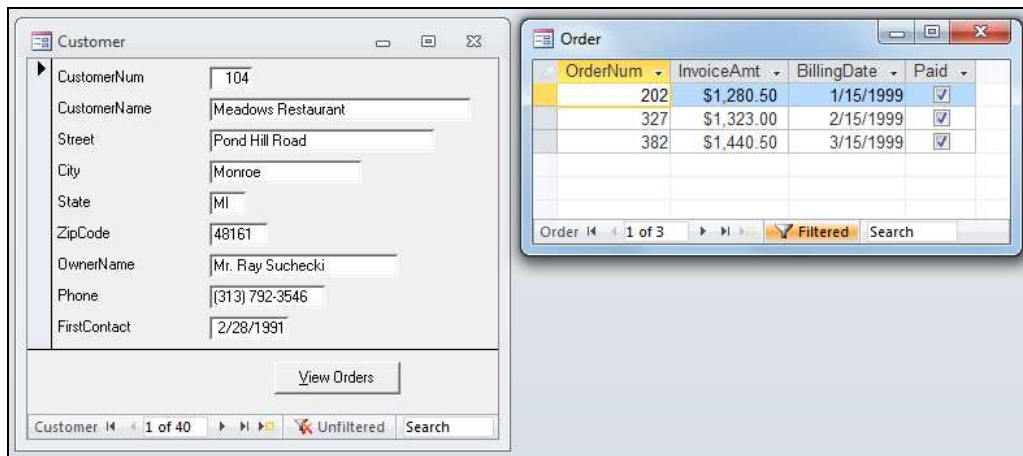
This *Guided Tour* features three events:

- The **Click** event, which occurs when a user clicks a command button object.
- The **Current** event, which occurs when a form displays a record. It fires when a form shows its initial record and each time a user navigates to a different record on the form.
- The **Close** event, which occurs when a form object closes.

Test Drive

Before you examine how macros are used in this database, do the following to become familiar with the functionality they provide:

1. Download the **Macros and Events.accdb** database from the *Downloads* area of my web site, placing it in your *CIS117 Data Files* folder, which you designated as a Trusted Location at the beginning of the course. Since the file is in a Trusted Location, Access will permit execution of macros and Visual Basic code. [**Note:** If you see a Security Warning and an Enable Content button, you opened the file from a non-trusted folder. Click the Enable Content button.]
2. Open the database. Notice that you don't see the *Navigation Pane*. Rather, **startup options** have been used to *hide* the *Navigation Pane* and to automatically open the *Customer* form. (We'll study startup options in Tutorial 12.)
3. Click the *View Orders* button in the *Customer* form's Form Footer section. Another form pops up, showing the customer's orders.



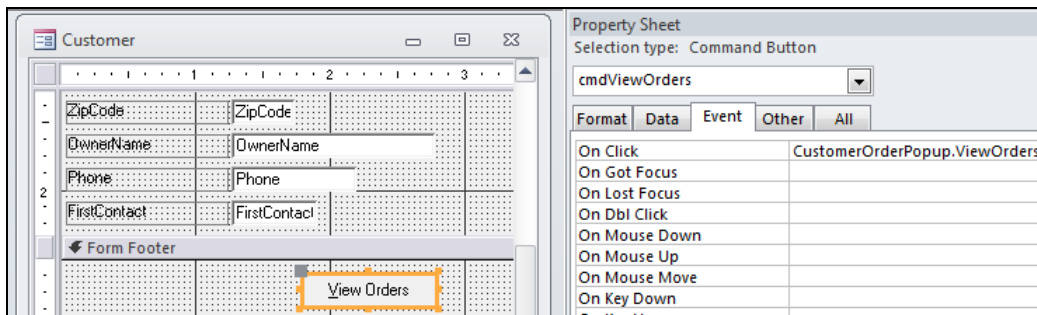
4. Now, navigate to a different record on the *Customer* form. Notice the pop-up form stays synchronized (i.e., displays the next customer's orders).
5. Navigate to a few other customers. The pop-up form continues to display the orders for which ever customer is selected in the *Customer* form.

6. Try to delete the *BillingDate* field value for one of the orders in the pop-up form. You are unable to do so because the form has been opened in read-only mode.
7. Attempt to add a new order, then try to delete an existing order. You'll be unsuccessful because the form's data is read-only.
8. Close the *Customer* form. Notice that the *Order Data* popup form automatically closes as well!
9. Press **[F11]** in the top row of the keyboard. This opens the *Navigation Pane*, which had previously been hidden.

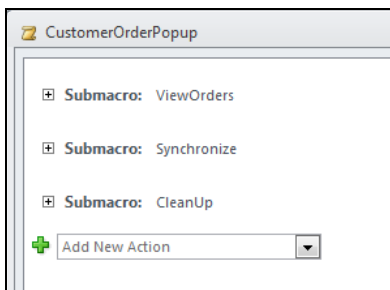
Command Button Engineering

Next, you'll discover how macros and events provide this functionality. You'll start by examining how clicking the View Orders command button caused the *Order Data* form to pop-up.

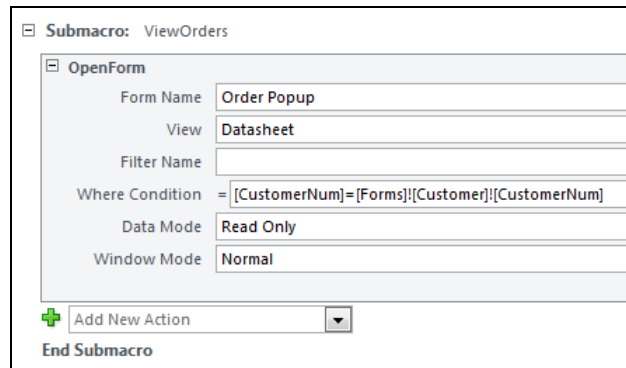
10. Open the *Customer* form in *Design View*, then scroll down until the *View Orders* command button is visible.
11. Right-click the *View Orders* command button and open the *Properties* sheet.
12. In the *Properties* sheet, click the *Event* tab. Scroll until the *OnClick* property is visible. When the *Click* event occurs, a macro named *CustomerOrderPopup.ViewOrders* runs. In the next few steps you'll navigate to the macro and see how it works.



13. In the Macros section of the Navigation Pane, right-click the *CustomerOrderPopup* macro group and select *Design View*.
14. Click *Collapse All*, if necessary. This macro group contains three submacros (*ViewOrders*, *Synchronize*, and *CleanUp*).



15. Click **+** to expand the *ViewOrders* submacro. It consists of a single **OpenForm** action.
16. Point to *OpenForm*, and then click **+** to view the *OpenForm* action's arguments.



The **OpenForm** action opens the *Order Popup* form in Datasheet view and uses its **Where Condition** argument to filter the orders so the pop up form displays orders for the customer displayed on the *Customer* form. The *Where Condition* reads:

[CustomerNum]=[Forms]![Customer]![CustomerNum]

Reading from left-to-right, this means that as the *Orders Popup* form opens, it should display only orders whose *CustomerNum* equals the *Customer* form's *CustomerNum*.

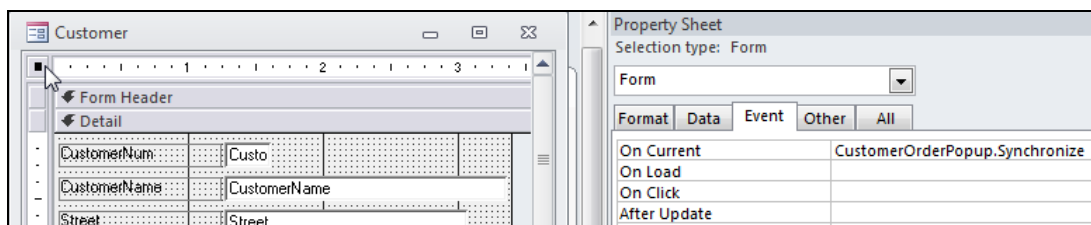
Also, notice the **Data Mode** argument's value. This explains why the *Order Popup* form would not allow you to add, delete, or change an order!

17. Keep the *CustomerOrderPopup* macro group open in Design View.

Current Events

Next you'll see how navigating to a different record in the *Customer* form causes different orders to appear within the *Orders Popup* form.

18. Return to the *Customer* form, in Design View, and click the Form selector button (at the intersection of the form's horizontal and vertical rulers, as illustrated in the screen shot below). This will cause the Property sheet display the Form's properties.



19. In the *Properties* sheet, click the *Event* tab. Scroll to the topmost property, *On Current*, which specifies what *Access* should do when the form's **Current** event happens. As you see, when the *Customer* form's *Current* event occurs, a submacro named **Synchronize**. runs. The *Synchronize* submacro does the work required to make the *Order Popup* form display the orders for the customer currently displayed on the *Customer* form.
20. Return to the *CustomerOrderPopup* macro group. Click *Expand All* in the *Macro Tools Design* tab.
21. Scroll so the *Synchronize* submacro actions and arguments are visible.

```

Submacro: Synchronize
  If IsLoaded("Order Popup") Then
    /* select the Order pop up form */
    SelectObject
      Object Type Form
      Object Name Order Popup
      In Database Window No
    /* display orders for the customer displayed on the Customer form */
    ApplyFilter
      Filter Name
      Where Condition = [CustomerNum]=[Forms]![Customer]![CustomerNum]
      Control Name
    /* return focus to the Customer form */
    SelectObject
      Object Type Form
      Object Name Customer
      In Database Window No
  End If
End Submacro

```

The *Synchronize* submacro begins by using **If** to test whether the *Order Popup* form is loaded. If so, three actions are executed: *SelectObject*, *ApplyFilter*, and another *SelectObject*. We'll soon examine each. Note that these three actions are indented and appear between **If** and **End If**. On the other hand, if the test evaluates as false (because the *Order Popup* form is not loaded) none of the three actions are executed (since there is no need to synchronize it with the *Customer* form). Since there are no actions beneath **End If**, the macro doesn't do anything when the *Order Popup* form isn't loaded... there is nothing to synchronize!

Let's focus on the three actions executed then the *Order Popup* form is loaded.

- The first **SelectObject** action selects the *Order Popup* form. This must occur so that the second action, **ApplyFilter**, filters the *Order Popup* form's records instead of the *Customer* form's own records.
- Next, the **ApplyFilter** action applies a filter that is identical to the one used by *OpenForm*. This makes the popup form display the correct set of orders for the customer displayed in the *Customer* form.
- The second **SelectObject** action re-selects the *Customer* form so the user can continue working with that form. Without this action, the user would remain in the filtered *Order Popup* form and would have to click on the *Customer* form to return the focus to that form.

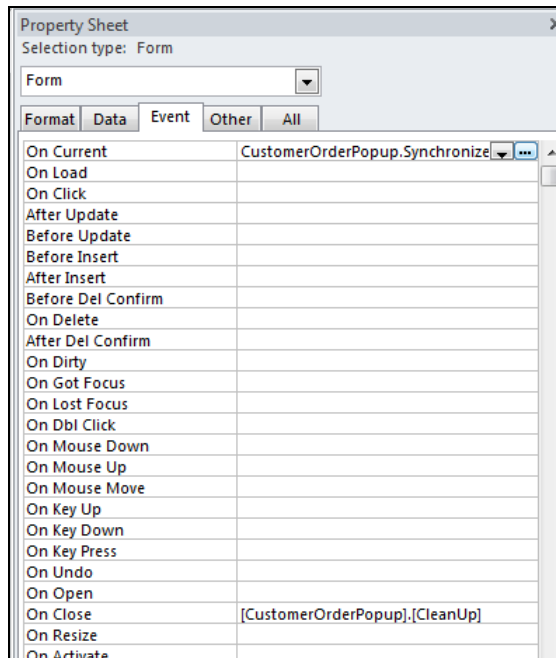
To recap: the *Synchronize* macro runs when the *Customer* form's *Current* event fires. The macro checks whether the *Order Popup* form is open. If so, it executes actions that (1) select the *Order Popup* form, (2) filter the popup form's records, and then (3) make the *Customer* form the current object. As a result, the macro causes the *Order Popup* form to display the orders associated with the customer being displayed on the *Customer* form.

22. Keep the *CustomerOrderPopup* macro group open in Design View.

Closing Ceremonies

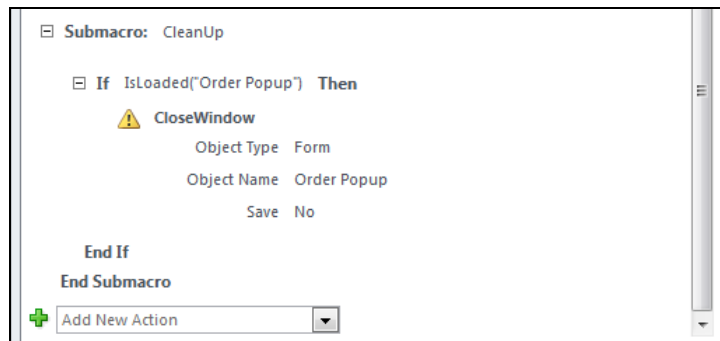
Lastly, you'll discover why the *Orders Popup* form automatically closes when the *Customer* form closed.

23. Return to the *Customer* form.
24. Verify that Form is the selected object. Scroll down in the *Properties* sheet until the form's *On Close* property appears. This property specifies what Access does when the form's **Close** event occurs.



When a user closes this *Customer* form, a macro named **CleanUp** runs.

- Return to the *CustomerOrderPopup* macro group. Scroll to the bottom of window until the **CleanUp** submacro is visible.



This **CleanUp** macro checks to see whether the *Order Popup* form is open. If so, it closes the *Order Popup* form's window. There is no need for the popup form to remain open when the *Customer* form is closing!

- Close the *CustomerOrderPopup* macro group.
- Exit *Access*.

Summary

In this *Guided Tour*, you have seen that a **macro** consists of one or more predefined **actions** that execute whenever an associated **event** occurs. By specifying the macro's name as the value for an event property, the macro runs whenever the event occurs. In other words, the **macro** is the **response** to the event.

You used two form-level events (**Current** and **Close**) and one control-level event (**Click**) to trigger submacros that performed appropriate actions in response to user activity. Doing so enhanced the usability of the forms.