Computers & the Humanities 281

Scripts in Revolution Continued

All Objects Have Scripts

As stated earlier, buttons are not the only Revolution objects that possess a script. All object types in Revolution (buttons, fields, images, and graphics) have the ability to hold and execute handlers. However, an object will do nothing but sit there and look purdy unless you equip them with handlers for various messages. The method to access and edit an object's script is consistent for all of them:

- Open the object's properties and click **Edit Script** in the inspection menu.
- Click on the **Script** button in the button bar with the object selected.

However, objects differ slightly in dealing with mouse events.

Fields

A field's primary purpose is arguably to hold and display text. Consequently, the default properties for a field are aimed to achieve that end. I can put a mouseUp handler in the script of a field to do something (like put the time in the message box) when the mouse is clicked on the field. However, when I click on the field, the handler I created is not executed. Instead I get an insertion point, ready for me to type something. Since fields are intended for text, Revolution assumes I want to type text. In order for a field to receive mouse events and execute corresonding handlers in its script, the **Lock Text** property must be selected. Otherwise, the message never gets to the handlers in the script. The handlers will only get the message if the text is locked for that field.

Graphics

If graphic objects' scripts possess proper handlers, they handle mouse events differently based upon their appearance. Unlike a button or field, the clickable area of a graphic is not the rectangular area represented by its eight handles. Instead, the clickable area consists of the graphic itself, e.g., with an oval graphic, the clickable area is the oval itself and not the rectangle created by the handles that bound it when selected as an object. Also, if a graphic is opaque, i.e., it is filled with a solid color or pattern, then the entire area of the graphic is "hot" and ready to receive mouse events and execute handlers. If the graphic is transparent, however, only the line delimiting the graphic (if visible) is able to receive mouse events.

Images

Image objects behave much like graphics. They can receive messages and handle events. All pixels in an image that are transparent, however, are consequently unable to receive any mouse events.

Cards, Groups, and Stacks

Revolution gives the same possibility of functionality to those objects that contain other objects. Like other objects in Revolution, cards, groups, and stacks each possess a script and may receive messages and execute handlers. The script of these three objects can be accessed through the properties palette like other objects. Cards, groups, and stacks can handle events for which they are programmed, just like other objects. One could conceivably put a MouseUp handler at card, group, or stack level. Consequently, any click not taken by a control object on the card or group "falls through" to the card, group, or stack and is handled by it. This concept is explained in more detail below.

Keyboard Shortcuts

Since working with the scripts of objects is a major part of Revolution development, there are keyboard shortcuts for each:

- **Any Control:** Command (apple)-option (Mac) or Control-alt (Windows) + point over control.
- **Selected Object:** Command (apple) (Mac) or Control (Windows) + E.
- **Card:** Command (apple)-Shift (Mac) or Control-Shift (Windows) + C.
- **Stack:** Command (apple)-Shift (Mac) or Control-Shift (Windows) + S.

Events and Messages

By way of review, a message is an announcement that an event has occurred, ostensibly made to the object the event affects. Events include user actions (such as typing a key or clicking the mouse button) and program actions (such as completing a file download or quitting the application). Revolution watches for events and sends a message to the appropriate object when an event occurs. These messages are referred to as built-in messages, and include mouseDown, mouseUp, keyDown, openCard, and other messages described in the Transcript Dictionary.

To respond to a message, you write a handler with the same name as the message. For example, to respond to a **keyDown** message sent to a field (which is sent when the user presses a key while the insertion point is in the field), place a keyDown handler in the field's script:

```
on keyDown theKey
    if theKey is a number then beep
end keyDown
```

Every time a user typed a number, the event would generate a keyDown message that would be sent to the field. Since the field has a handler for that event, Revolution would then produce a system alert sound. In this manner an object can have several handlers within its script to accept the various messages which it may receive.

Object Hierarchy and the Message Path

As we discussed previously when talking about object properties, Revolution's object types can be arranged in an object hierarchy, with each object being owned by one on the level above it, another object owning that one, and so on. For example, a button's object hierarchy includes the button itself, the card the button is on, and the stack that the card is in, in that order.

All messages are sent to a particular object. For example, when the user clicks an object, Revolution sends a **mouseDown** message to the object that was clicked. If the object's script doesn't contain a handler for that particular message, the message is sent onward to the next object in the message path. The message path is the set of rules that determine which objects, in which order, have the opportunity to respond to a message. The message path is based on the object hierarchy.

For example, suppose the user clicks a button, causing Revolution to send a mouseUp message to the button. If the button's script does not contain a handler for the mouseUp message, the message is passed along to the card the button is on, and if the card contains a mouseUp handler, then it is executed. If the card does not handle the mouseUp message, it is next passed on to the stack in which the card is located.

These are the general rules making up the message path:

- Messages sent to a control are passed next to the card the control is on.
- Messages sent to a grouped control are passed next to the group the control is in.
- Messages sent to a group are passed next to the card the group is on, unless the card has already received the message.
- Messages sent to a card that has no groups placed on it are passed next to the stack in which the card is found. Messages sent to a card that has groups placed on it are passed next to any groups on the card that haven't already received the message.
- Messages sent to a substack are passed next to the substack's main stack.
- Messages sent to a main stack are passed next to the Revolution development environment, or if the stack is open in a standalone application, the are sent to the main stack you designated when you built the standalone.

Since all the objects in Revolution have a script, each of them could contain a mouseUp handler to handle a mouseUp event. When a mouseUp event occurs, Revolution looks through each script in sequence, looking for a mouseUp handler. Messages that are trapped by a handler don't get sent any further along the message path, so each message triggers only one handler unless that handler explicitly passes the message along to the next object. For example, when a mouseUp event occurs, Revolution goes through the following hierarchy of scripts:

1. Topmost **Button** or **Field**, **Image** or **Graphic** at the click point. Other objects at this level are ignored.
2. The **Group** the control is in, if the control is grouped.
3. Current **Card**.
4. Any other **Groups** on the card that haven't already received the message.
5. Current **Substack**.
6. The **Mainstack**.
7. The **Revolution Environment**, if you are in editing mode.

The first mouseUp handler the message finds is exectued and the process stops. In the message hierarchy only the topmost button or field gets the message. If there is no mouseUp handler there, Revolution next looks at the card script, even if an underlying button or field does have a mouseUp handler. A message cannot jump to other control objects at the same level without explicitly being told to do so.

Practically speaking, this is another powerful feature of Revolution. Suppose you have a card on which you have ten buttons that perform essentially the same function (play an audio clip, for example). Initially you would be tempted to script each button individually. However, any changes you make to one button's script would have to be changed in the other nine as well to ensure they all act uniformly. Knowledge of the message hierarchy, however, allows me to create a more general handler that I can place in the card script. Such economy is not only elegant, but it saves time with revisions as well. I realize that this is an abstraction, but the practical uses of the object hierarchy will become evident later.

Unhandled Messages

If a built-in message passes through the entire object hierarchy without finding a handler, it is ignored. If a custom handler is called and the message reaches the end of the object hierarchy without finding a handler, it causes an execution error.

Target/Me

There are two special-purpose variables defined in Transcript that are related to the process of handling messages. The object that the message was originally sent to is called the message's **target**. You can get the target from within any handler in the message path by using the target function. **Me** references the object that contains the handler that handles the message. Hence, in a case where the message is passed on and handled by another script, the **target** is distinct from **me**.

Again, practically speaking, these two terms are integral to the function of the abstract example of ten buttons given above. If I have a handler on the card for a certain event, the ten buttons (sans the handler) will pass the message up the hierarchy to the card, which will then handle the event. In this case, the button clicked on is the target and can be referred to as the "target" in the handler of the event. Since the card possesses the handler and handles the event, it can be referred to as "me" within that handler. In this case the handler in the card script could check to ensure that the target is actually one of the ten buttons before executing its commands (to avoid messages passed from objects other than the buttons). If against better judgment you placed a handler in each of the buttons, the button (with a handler for a given event) is both the target and me, as it first receives the message and handles it. As before, the power of this concept will become more clear as we encounter some less abstract exercises in the course.

Course Schedule
Main Page