Messages:

Responding to the user

Messages drive applications to do things. Whenever a user does anything, a message is sent to the relevant objects telling them what happened. In this tutorial, you'll learn about some common messages and how they trigger scripts. You will also learn how messages pass from object to object as they look for a script to intercept them.

Key topics covered in this tutorial

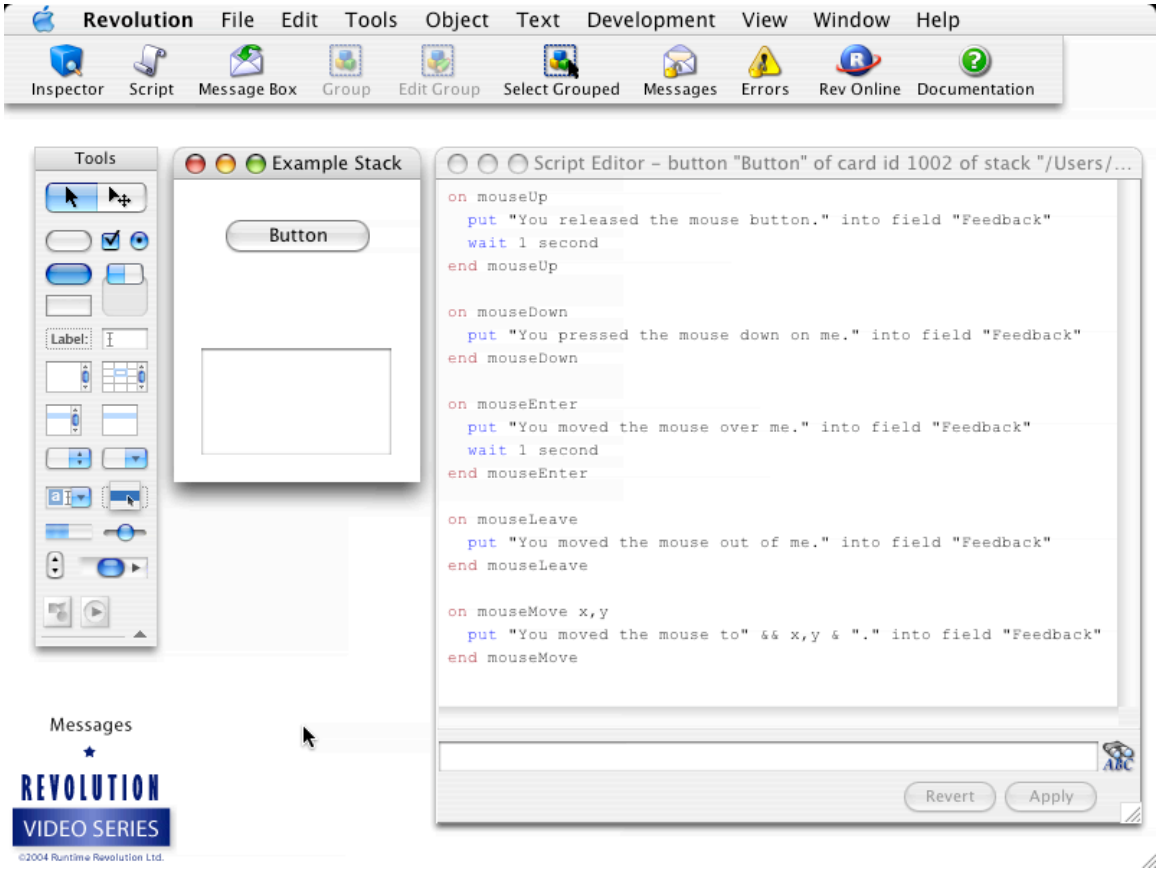
- Applications are driven by messages which get sent in response to user actions
- Common messages
- What happens to messages that aren't intercepted
- How to intercept messages from several objects at once

See also: Documentation: Messages and the message path, Object types and object references

Whenever a user interacts with an object, a message is sent to that object telling it what the user did. For example, if the user clicks on a button, a message is sent to the button saying 'mouseDown'. As the user releases the mouse button, a message is sent to our button saying 'mouseUp'.

Messages are sent whenever you click on a button or any other object, move the mouse, or type any key on the keyboard. Some messages go to buttons, some to text fields, and some go directly to the card – depending on what the user does. These messages are used to trigger an application to do things – so a button can contain a script which intercepts the 'mouseDown' message and then responds by displaying a dialog for example. If that is the only script in the button, the other messages that are sent to the button, such as mouseEnter when the user rolls the mouse over the button, are simply ignored.

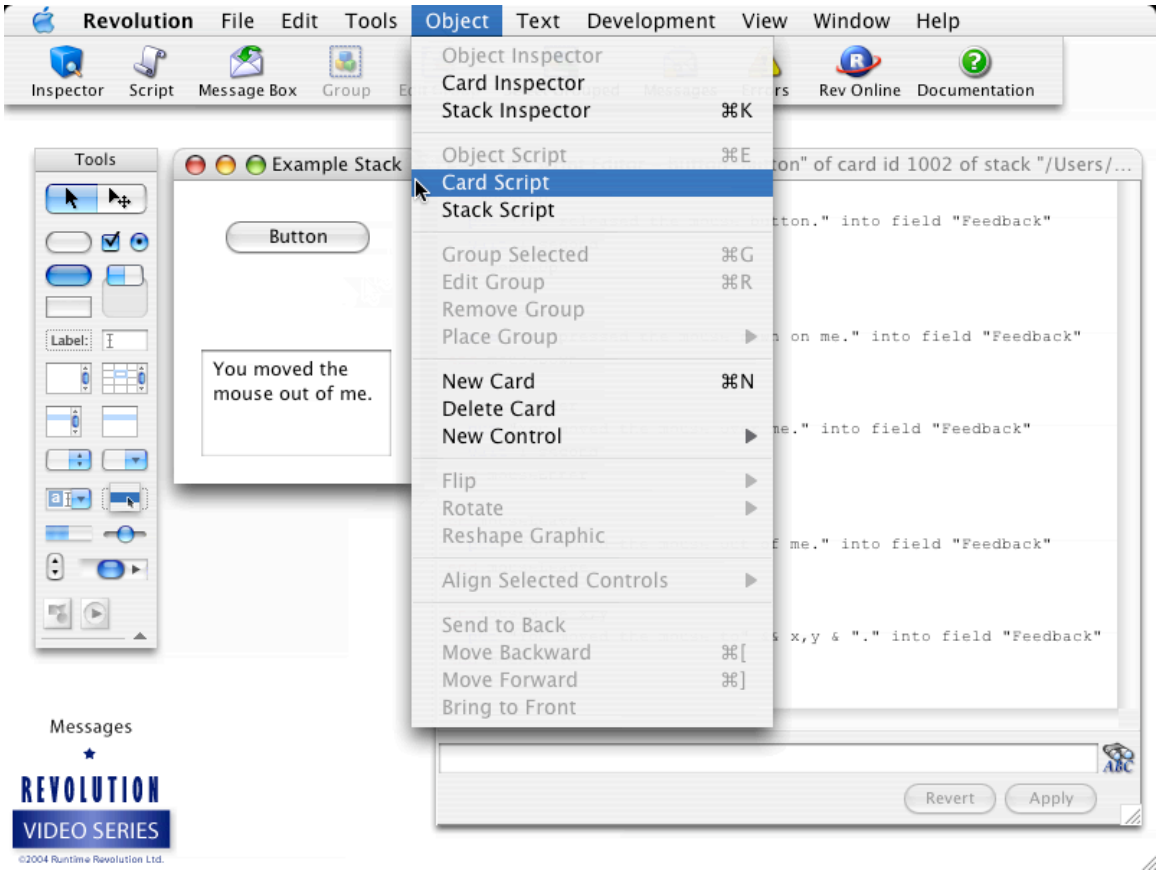
Let's take a look at some common messages. You can see we have a button and a field called 'Feedback'.



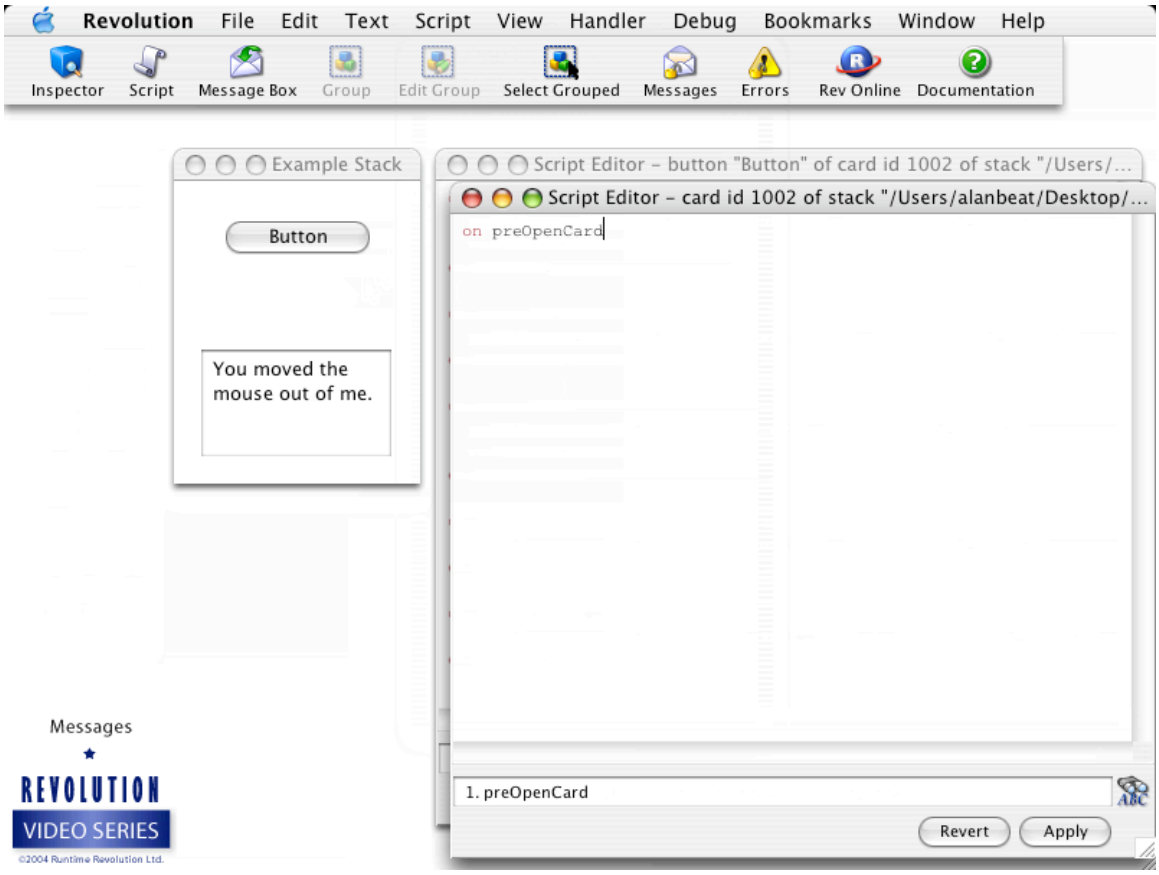
In the script for the button you can see the messages we've listed for the button to respond to. If we move the mouse into the button you can see that the mouseEnter message is sent to the button, and the button responds by putting text into the feedback field. As we move the mouse around inside the button, you can see from the field read-out that the mouseMove message is being sent. When we press the mouse button down, you can see a mouseDown message is sent. And when we release the mouse, you can see the mouseUp message being sent. Finally, if we move the mouse outside of the button, you can see the mouseLeave message is sent. If you want to try this script out for yourself, make sure you create a button and a field, and name the field 'Feedback'. You can copy the scripts from the appendix at the end of this tutorial.

Revolution also sends messages when you perform many actions in a script. For example, if you change card or open a new stack, the card and stack will receive messages telling them that they

are being opened and closed. You can use the preOpenCard message to make changes to a card before it is displayed. Or, use the openCard message to start an action once a card is opened and after it has been displayed on the screen. Let's take a look at how you add a preOpenCard message handler to a card. First, we need to edit the card script.



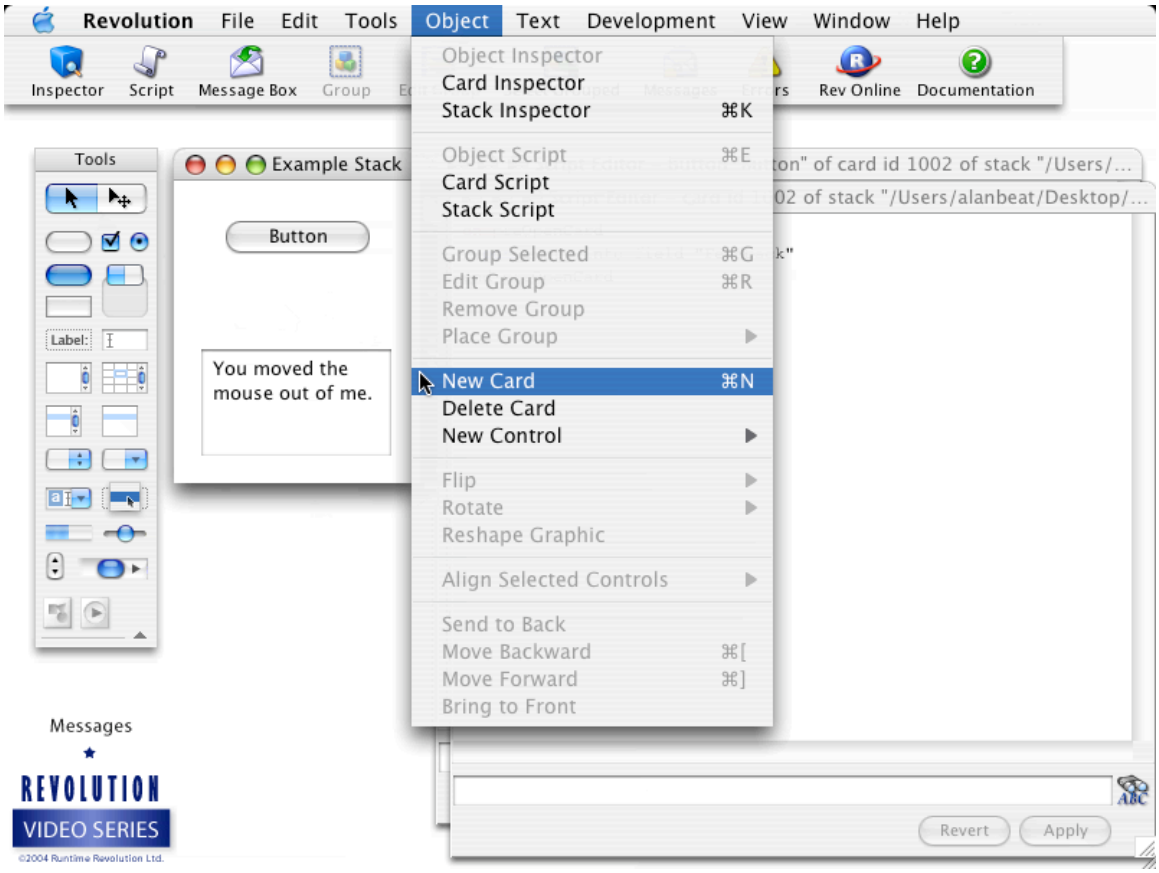
Now we'll add 'on preOpenCard'.



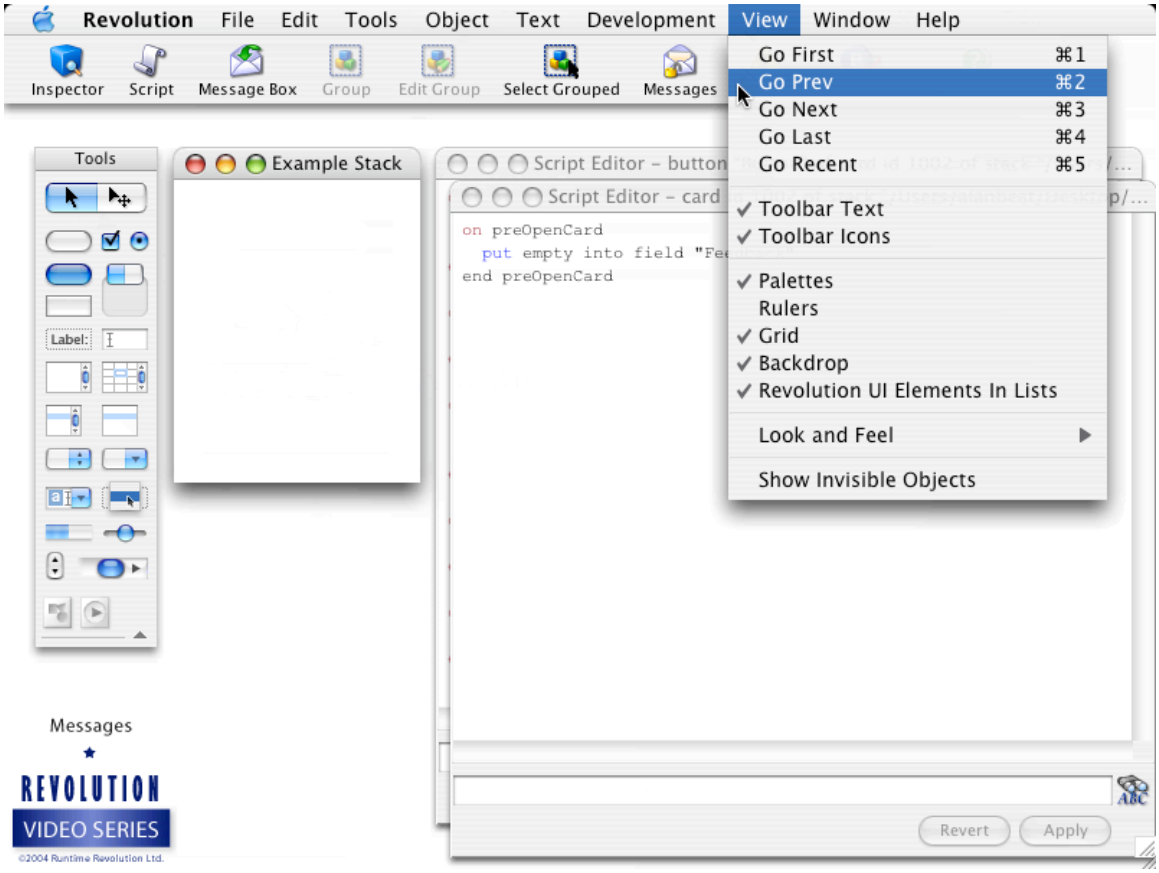
Let's say that we wanted to make sure the feedback field on this card is empty before the card is displayed on the screen.

```
put empty into field "Feedback"
```

Notice that the field still contains text from our last example. Let's create a new card,

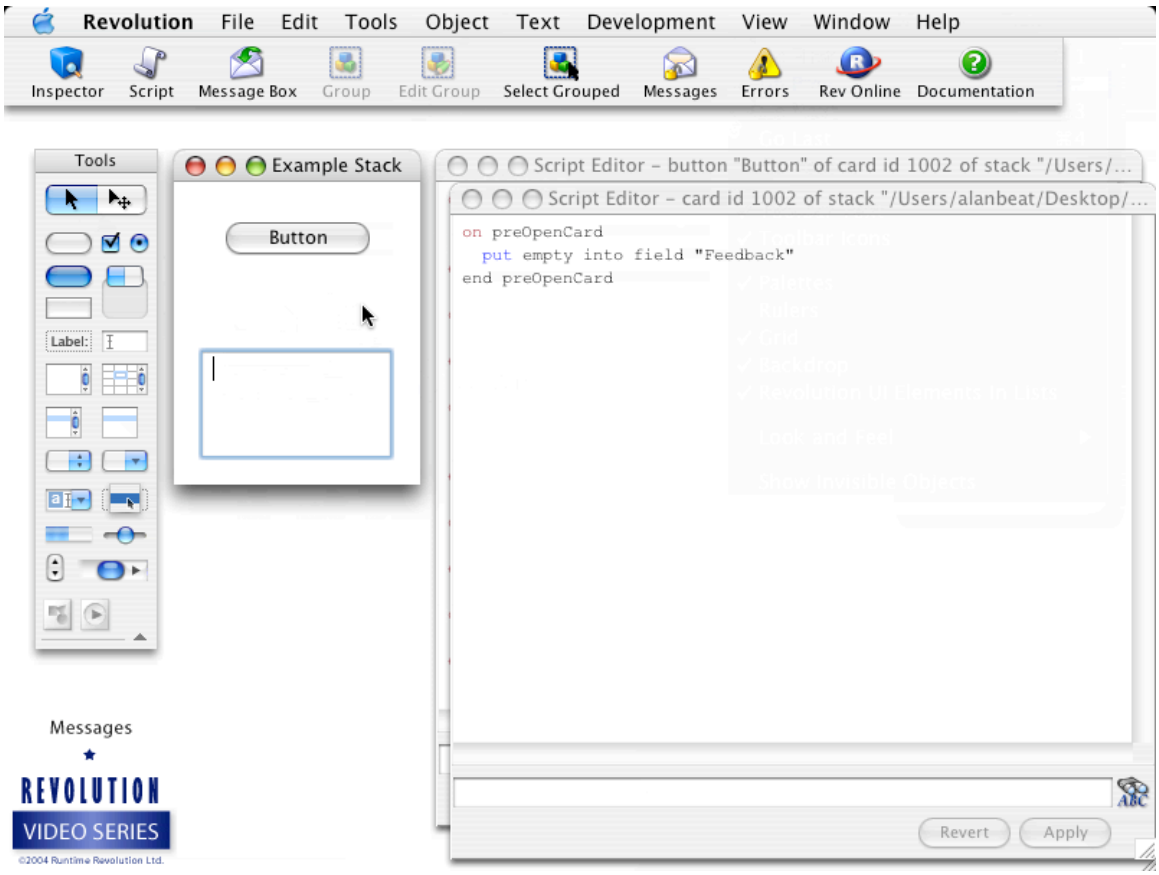

Revolution File Edit Tools Object Text Development View Window Help
Inspector Script Message Box Group Rev Online Documentation
Object Inspector
Card Inspector
Stack Inspector ⌘K
Object Script ⌘E
Card Script
Stack Script
Group Selected ⌘G
Edit Group ⌘R
Remove Group
Place Group
New Card ⌘N
Delete Card
New Control
Flip
Rotate
Reshape Graphic
Align Selected Controls
Send to Back
Move Backward ⌘[
Move Forward ⌘]
Bring to Front
Tools
Example Stack
Button
You moved the mouse out of me.
Label:
Revert
Apply
Messages
REVOLUTION
VIDEO SERIES
©2004 Runtime Revolution Ltd.

then go back to the first card.

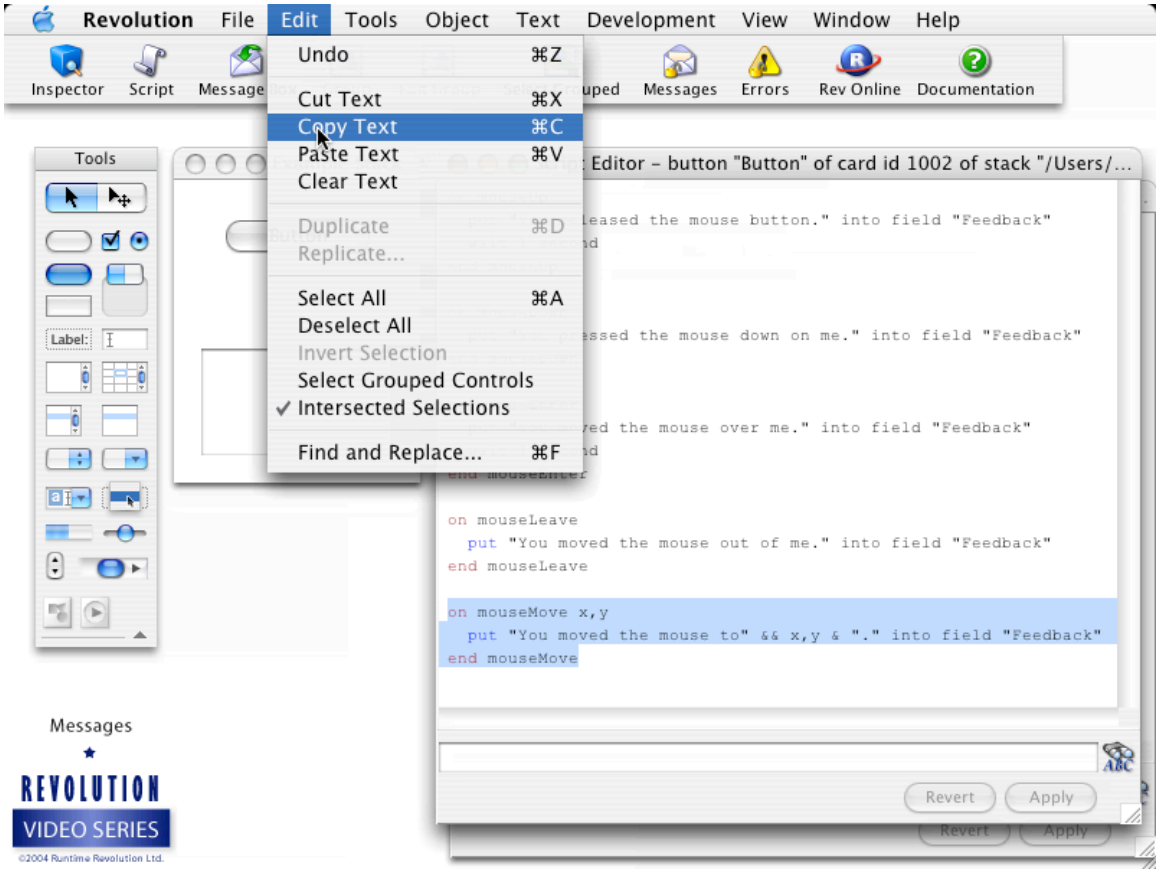


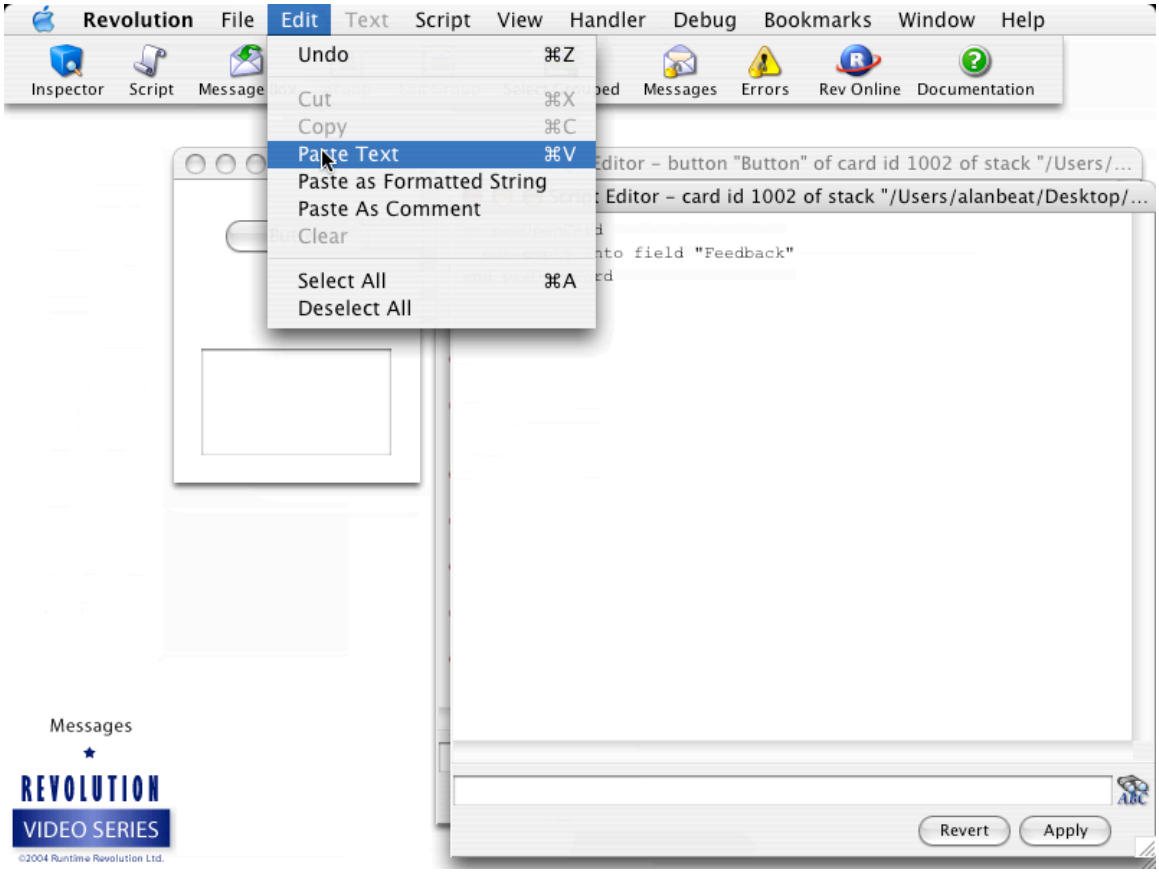
When we go back to the first card, the preOpenCard handler will run, and the field 'Feedback' should be empty.

And as you can see, the field is now empty!



You will notice that messages are always sent to the object that they affect. For example, preOpenCard was sent to the card that was about to be opened. The mouse messages were sent to the button that was underneath the mouse. Let's see how this works, by copying the mouseMove message handler in the button script into the card script,


Revolution File Edit Tools Object Text Development View Window Help
Inspector Script Message Messages Errors Rev Online Documentation
Undo ⌘Z
Cut Text ⌘X
Copy Text ⌘C
Paste Text ⌘V
Clear Text
Duplicate ⌘D
Replicate...
Select All ⌘A
Deselect All
Invert Selection
Select Grouped Controls
Intersected Selections
Find and Replace... ⌘F
Tools
Label:
Editor – button "Button" of card id 1002 of stack "/Users/...
on mouseLeave
put "You moved the mouse out of me." into field "Feedback"
end mouseLeave
on mouseMove x,y
put "You moved the mouse to" && x,y & "." into field "Feedback"
end mouseMove
Revert
Apply
Messages
REVOLUTION
VIDEO SERIES
©2004 Runtime Revolution Ltd.



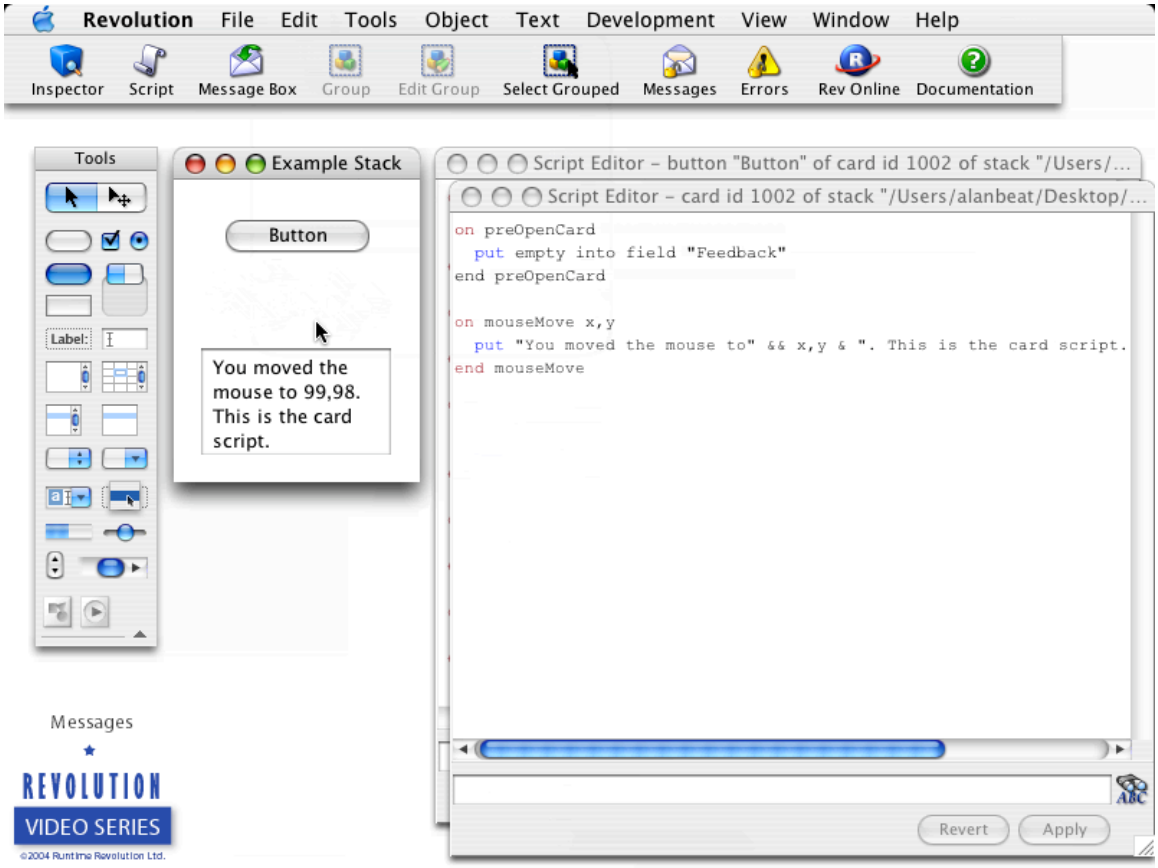
and modifying it slightly so we can see when the message is getting sent to the card instead of the button.

We'll modify this handler so we can clearly see that the message is getting sent to the card, not the button.

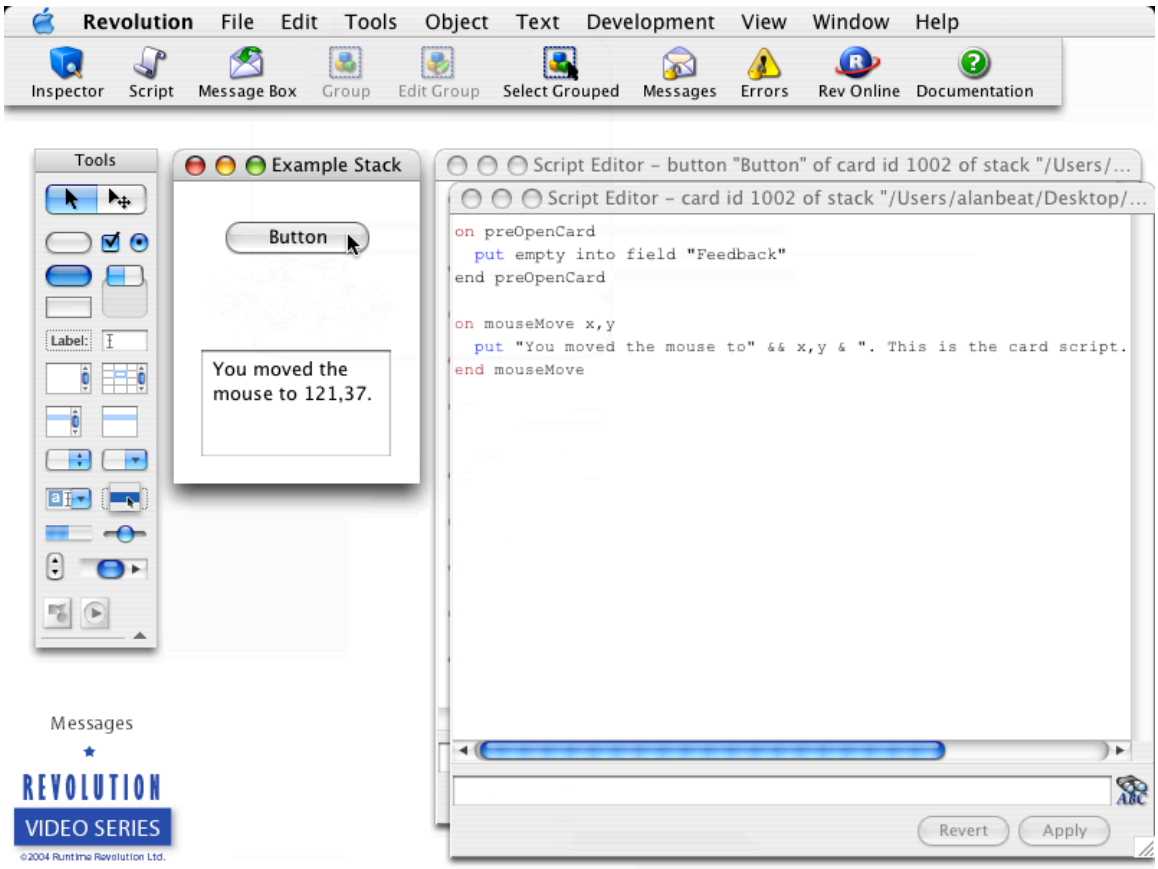
```
on mouseMove x,y
  put "You moved the mouse to" && x,y & ".  This is the card script"\
    into field "Feedback"
end mouseMove
```

(Note that here this script is too long to fit on one line without wrapping so we've used the Transcript line continuation character '\' to separate it into two lines. Revolution will still treat it as a single line of script when it executes.)

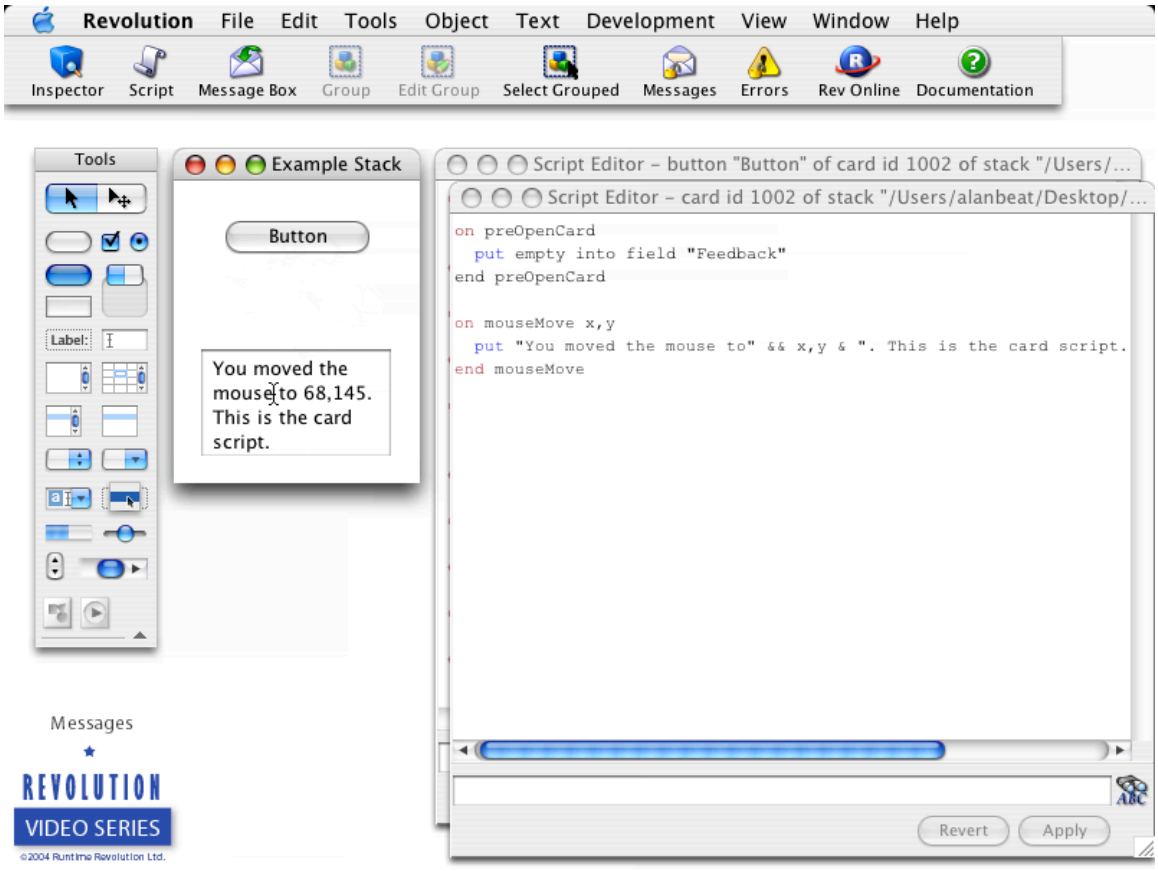
Now, if we move the mouse over the card we can see the message is being sent to the card.



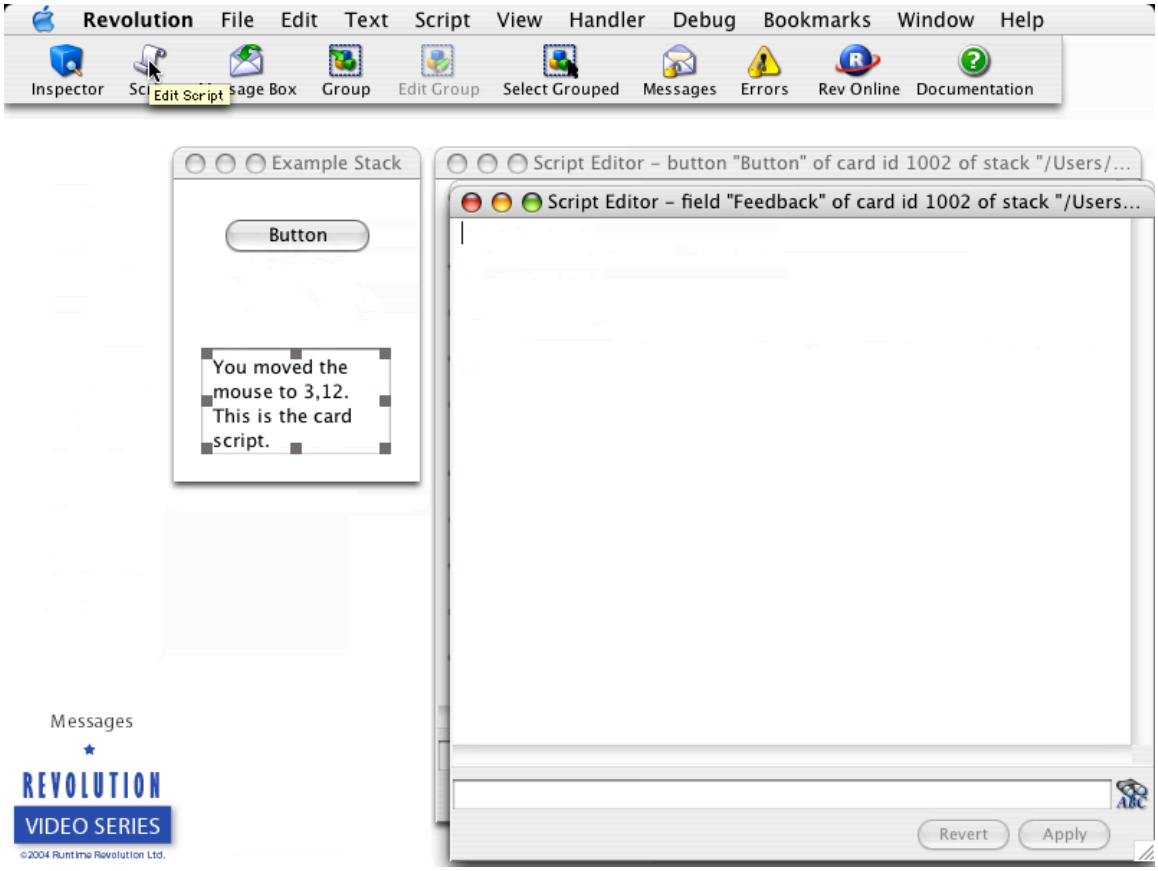
If we move the mouse over the button we can see the message is being sent to the button.



Let's see what happens when we move the mouse over the field?



Wait a minute – it seems that the card is getting the message. But the mouse is over the field? Let's take a look at the field script – its script is empty!



What is happening here is that we've just discovered Revolution's message path.

Although messages are initially sent to specific objects, they don't just stop there. A mouseDown message is first sent to the button, then it is passed on to any background group, then to the card and finally to the stack. Along the way, as it travels through the object layers, it can be intercepted at any stage by a script starting with 'on mousedown', whether the script is in the button, or elsewhere, such as in the card itself.

Why is this important? Well, this feature means that you can have one script, perhaps at the card level, which can intercept a type of message which may have originated from any of the objects on the card. For example, we might want all the buttons on the card to do different things when they are clicked – so each would contain its own mouseDown script. But we also want all of our buttons to beep when they are released, so we can put a single mouseUp script in the card to do that – saving us the effort of putting it into every button. We know that all mouseUp messages will travel a path taking them through the button and on to the card where they will now be intercepted.

This message path makes it easy to write common script handlers for messages. But what if you want to block the message somewhere so that it doesn't get handled by an object higher up in the hierarchy? Let's say you didn't want the mouse read-out to continue to appear when we move the mouse over the feedback field. We'll add a simple handler to the field.

```
on mouseMove
  exit mouseMove
end mouseMove
```

Now we will find that the message is only reaching the card when the mouse is over the card; when it is over the field, it is blocked. When Revolution encounters a handler in an object, it always stops without passing the message further up the path.

This leads us to one final feature of the message path hierarchy that is often useful. What if you want the field to do something with the mouseMove message when we move the mouse over it, and then still allow the card to receive and act on the mouseMove message too? In other words, what if you want both the field and the card to act on the same message? To do this, you can change the mouseMove message handler in the field to tell Revolution to let this message pass up to the next object on the message path when it has finished.

```
on mouseMove
  beep
  pass mouseMove
end mouseMove
```

Now when we move the mouse over the field, we should get a beep from the field, and a read-out of the mouse co-ordinates from the card.

Appendix: Scripts used in this tutorial

```
on mouseUp
  put "You released the mouse button." into field "Feedback"
  wait 1 second
end mouseUp

on mouseDown
  put "You pressed the mouse down on me." into field "Feedback"
end mouseDown

on mouseEnter
  put "You moved the mouse over me." into field "Feedback"
  wait 1 second
end mouseEnter

on mouseLeave
  put "You moved the mouse out of me." into field "Feedback"
end mouseLeave

on mouseMove x,y
  put "You moved the mouse to" && x,y & "." into field "Feedback"
end mouseMove
```

```
on preOpenCard
  put empty into field "Feedback"
end preOpenCard

on mouseMove
  exit mouseMove
end mouseMove

on mouseMove
  beep
  pass mouseMove
end mouseMove
```