Computers & the Humanities 281

Object Properties

As we discussed before, every Revolution object has properties, or characteristics that determine it appearance and function: its name, location, style, attributes, text characteristics, etc. These are established when the object is created and mostly set by choices made from the properties palette. Transcript has a full range of commands to examine and manipulate these properties within a handler. In other words, it is possible with any given object to view and change the characteristics of objects through the use of transcript commands within a handler

References to a property in Transcript have the general form:

```
the <property> of <object>
```

where

both **`the`** and **`of`** are required keywords
`<property>` is the name of the property in question.
`<object>` is a reference to a button, field, or other Revolution object.

Common Properties

As has been noted before, there are a number of properties which are common to all types of objects. Most properties can be accessed easily within a handler wherever the value of that property is appropriate. The most common methods are by either an `if`, a `put`, or a `get` statement:

```
if the <property> of <object> is <propValue>
put the <property> of <object> into <container>
get the <property> of <object>
```

where

`<property>` is the name of the property in question.
`<object>` is a reference to a button, field or other Revolution object.
`<propValue>` is the value of the property in question (including booleans).
`<container>` is a reference to any container.

You usually set properties, of course, by using the properties palette. However, you may use the `set` command within a handler to change directly a particular property of an object to a particular value:

```
set the <property> of <object> to <propValue>
```

where

`<property>` is the name of the property in question.
`<object>` is a reference to a button, field or other Revolution object.
`<propValue>` is the value of the property being adjusted (including booleans).

The power of this capability is that you can access the properties of objects within scripts, i.e., their values are available for reference and use. You can then script handlers to perform certain functions based upon the values of the various properties, including setting that property to a new value. This implies, then, that through the judicious use of handlers for various messages, you can alter the appearance and function of various objects.

Here are a few of the properties common to all objects which may be accessed and manipulated:

- **Name:** The name of an object is the string of characters that have been input as a given name to the object. Specifying just `name` gets the object's local identity: the layer and kind of object as well as the actual name. Use `short name` to get just the name by itself. With `long name` you get the object's complete genealogy.
- **Label:** The label of any object is the string of characters that have been input to serve as a label to the object. This is particularly useful with buttons.
- **ID, Number, Layer:** Each object has these three properties assigned to it dependent upon when it was created. All three properties can be accessed. Not surprisingly, a set command cannot change the ID of the object (the ID cannot be changed by any method). Revolution won't let you set the number of the object either. This has to be done manually with send farther/bring closer. But for some reason it *does* allow setting the layer, which indirectly changes the number property. Go figure.
- **Visible:** The visible property tells whether the object is showing or hidden. Hence, the value of the property is either true or false. If you were to use the `hide` statement with an object, it would make the visible property of that object have the value of false, and `show` would then change the value of the visible property to true.
- **ToolTip:** Here you can determine what tool tip (if any) is to be shown the user.

Text Properties

All of the font-related attributes of a **field** are properties that can be retrieved and set:

- **textSize:** the size of the characters
- **textHeight:** the height of the lines
- **textFont:** the font, of course
- **textStyle:** the font style of the characters

- **textAlign:** the alignment of lines

The font attributes of **other control objects** work in the same manner. In early versions of Revolution this was the *only* way. You couldn't specify font attributes for buttons with menus or the Text Properties dialog box like we have done. You had to use set commands:

- **textFont:** the font of the button name
- **textSize:** the size of the name
- **textStyle:** the font style of the name
- **textAlign:** the alignment name line

Color Properties

All the color-related attributes of an object are properties that can be retrieved and set:

- **backgroundColor:** the color of the object
- **borderColor:** the color of the border of an object
- **foregroundColor:** the text color of an object
- **shadowColor:** the color of the shadow (if the object has one)

For any of these properties, a standard color name may be used ("red", "blue", "aquamarine", "mistyRose", etc.), or the RGB values ("255,0,128"), or the HTML-style color indicator ("#A366FF").

Location Properties

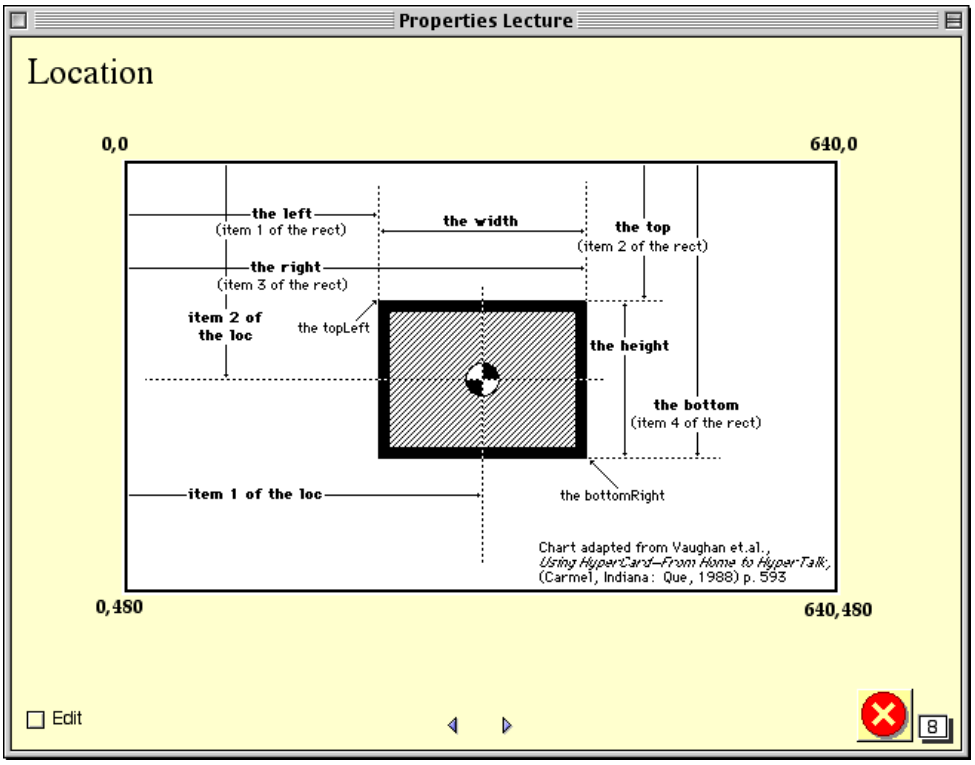
The Revolution coordinate system uses pixels as the unit of measurement and the upper-left corner of the card as the point of reference. Usually two numbers are provided, separated by commas. The first has reference to X (meaning left to right), and the second has reference to Y (meaning up and down). **location (loc):** The location of an object is specified as the X,Y coordinates of the center of the object in reference to the upper left-hand corner. You change the location of an object by setting its location property in an `x,y` format.

Other Position and Dimension Properties: There are several more properties pertaining to the position and dimensions of objects:

- **left, right, top, bottom:** the side properties change the position without changing the shape of the object
- **width and height:** change shape without changing location.
- **rectangle (rect):** gives all four dimensions (left, top, right, bottom) and can be used to change size, shape, and position at will.

There is no validity checking. You can set right less than left and top greater than bottom and sort of turn the object inside out!

This chart shows all of the different location and dimension properties and how they relate to each other.



Button Properties

There are two properties unique to buttons that are closely related:

- **Hilite:** The hilite property tells if the button is hilited or not, i.e., how the button *appears*. When set true the button appears hilited (inversed). When set false the button looks normal. The hilite is not the same as autoHilite.

- **AutoHilite:** This property determines how the button *behaves* when clicked. When set true the button automatically inverses on the down click and returns to normal on the up click. When set false nothing happens when the button is clicked.

All button styles work in this manner except checkBox and radioButton. These two styles hilite on one full click and restore on another full click. With autoHilite you get the default behavior, which is usually adequate. However, using the hilite property you can control hiliting explicitly in the script, or have certain events take place based upon the hilite of a certain button.

Other button properties that can also be retrieved and set include:

- **icon:** The icon property is unique to buttons and can be set by specifying the number of the desired icon. Setting the icon of a button to zero removes any icon setting.
- **showName:** set to either true or false, this determines if the name/label of the button is displayed
- **disabled:** set to either true or false, this determines if the button is clickable or not

Field Properties

A number of properties unique to fields can also be retrieved and set. The possible values for these properties are either true or false:

- **vScrollbar,hscrollbar:** determines if the field has appropriate scrollbars
- **showLines:** determines if the field appears as lined paper
- **opaque:** determines if the field is transparent or not
- **disabled:** determines if the field can execute its script or not (and it "appears" disabled, depending upon the color of the text).
- **lockText:** determines if the text in the field can be altered or not
- **dontWrap:** determines if the text in the field wraps automatically or not

Later in the course we will talk more about the properties of the text within containers (fields and variables).

Graphics Properties

Graphics in Revolution have their own properties. They are very similar to those of fields and buttons, such as font attributes:

- **style:** the style of a graphic can be retrieved and/or changed (with obvious limitations: "oval," "regular," "polygon," "line," etc.
- **opaque:** determines whether the graphic is transparent or not
- **showBorder:** shows or hides a rectangular border around the graphic.
- **lineSize:** is set to a number that adjusts the size of the line defining the graphic.

Each style of graphic has unique properties also that go along with it. For example, with the "oval" graphic, the Arc Angle and start Angle can be set.

Image Properties

All of the properties which are common to objects are accessible and have the ability to be manipulated with image objects. There are some useful unique properties, however:

- **blendLevel:** This is an integer value between 1 and 100. Setting the blendLevel of an image alters its translucency, making it transparent to one degree or another (1 is completely opaque and 100 is completely transparent), allowing pixels behind it to show through.
- **fileName:** If you have referenced an image's data rather than importing it into your stack, then the fileName property contains the path to that image. Changing this to a legitimate path will change the data the image displays.

Further Exploration

There are number of properties, both shared and unique to the different object types, that will not covered in this course. These may be discovered either by reading the documentation or by using the property inspector. Upon examining the property inspector for any object, hovering the mouse over the various properties will give you the actual name of the property by which it must be referenced. Using that one may obtain the value of that property for that particular object and thereby determine how that property may be altered. Experimentation is welcome here (and even encouraged), though you may get some unexpected results.

Course Schedule
Main Page