Miscellaneous Objects

This chapter discusses four unrelated objects: user, transcript, folder and time. [the last two are still not discussed]

User

The user object has several messages for interacting with the user via modal notes. Each of these messages displays a note to the user. The system will not proceed until the user answers the note by tapping on one of its buttons.

```
ask user msg yes-button y no-button n with-cancel
```

This message displays a note with two buttons at the bottom. The text displayed in the note is the string msg. By default the two buttons are labeled ‘Yes’ and ‘No’, but the labels can be changed with the yes-button and no-button arguments. The note is modal and the user cannot continue until a button is tapped. When the user taps a button, the note is removed and the result of the ask message is true if the user tapped the ‘Yes’ button, and false if they tapped the ‘No’ button.

If the optional argument with-cancel is present, then a third button is added labeled ‘Cancel’. You can’t change the text of this button. If the user taps this button instead of the other two, then the note is removed and the result of the ask message is ???.

Examples:

```
a := choose 1 to 10.
 until [ ask user "Do you like the number" && a.name & "?" ]


do [ a := choose 1 to 10 ].


$ answer.
 a := 23.

answer := ask user "Change this number?"


yes-button "Double It"


no-button "Halve It"
```

```
with-cancel.
 if (answer is-not ???) then [

   if answer then [ a := a * 2 ]

   else [ a := a / 2 ].

].
```

```
tell user msg ok-button o with-cancel
```

This message is very similar to the ask message, except that it only puts up one button with a default label of ‘OK’. You can change the label of the button with the ok-button argument.

This message is often used to tell the user something has been done, or to confirm an action.

Examples:

```
 tell user "Time for dinner!"
```

```
$ confirm.

confirm := tell user "Really remove bob from the list?"

ok-button "Remove"

with-cancel.
 if (confirm is ???) then [ return ].
user canceled

santas-list remove "bob".
```

```
query user msg default-text d ok-button o with-cancel
```

This message displays a note with a write in field. The write in field starts off with the text supplied by the default-text argument (or blank by default). In addition, the note has an ‘OK’ button and an optional ‘Cancel’ button. The text of the ‘OK’ button can be changed with the ok-button argument. The user can edit the text in the write in field. If the user taps the ‘OK’ button, then the text in the write in field is returned. If the user taps the ‘Cancel’ button (if any) then ??? is returned.

Examples:

```
$ n.

n := query user "What is your name?".
 tell user "Hello" && n.


$ new-item.

a := "apples".

new-item := query user "Enter a fruit:"


default-text a with-cancel.
 if (new-item is ???) then [ return ].

a := new-item.


user choose from g
```

This message displays a pop-up menu of the items in the group g. If the user taps on one of the items, the menu is removed and that item is returned as the result of the choose message. If the user taps outside the menu, then the menu is removed and ??? is returned.

Example:

```
$ animals, new-pet.

animals := new group.

animals add "bird".

animals add "cat".

animals add "dog".

animals add "zebra".

new-pet := user choose from animals.
 if (new-pet is ???) then [ return ].
 return "Eric the" && new-pet.
```

Transcript

The transcript is a window that displays text messages. It is used only for debugging: the window will not exist if you give your completed application to someone without Glyphic Codeworks. You can have your scripts write text onto the end of the transcript at any time. This can sometimes help you understand and debug your programs. The transcript is not

infinitely long: it only holds the last several thousand characters. You can open the transcript from the System menu.

transcript put msg

This is the only message transcript understands. It appends the string msg to the end of the transcript on a new line.

Example:

```
$ n, root-n, old-guess.

n := 2.

old-guess := 0.

root-n := n.extended.
 until [ (root-n - old-guess abs) < 1e-15 ] do [

   old-guess := root-n.

   root-n := (n / old-guess + old-guess) / 2.

   transcript put "root-n is now" &&

      (format root-n using "9.################").

]
 return root-n
```