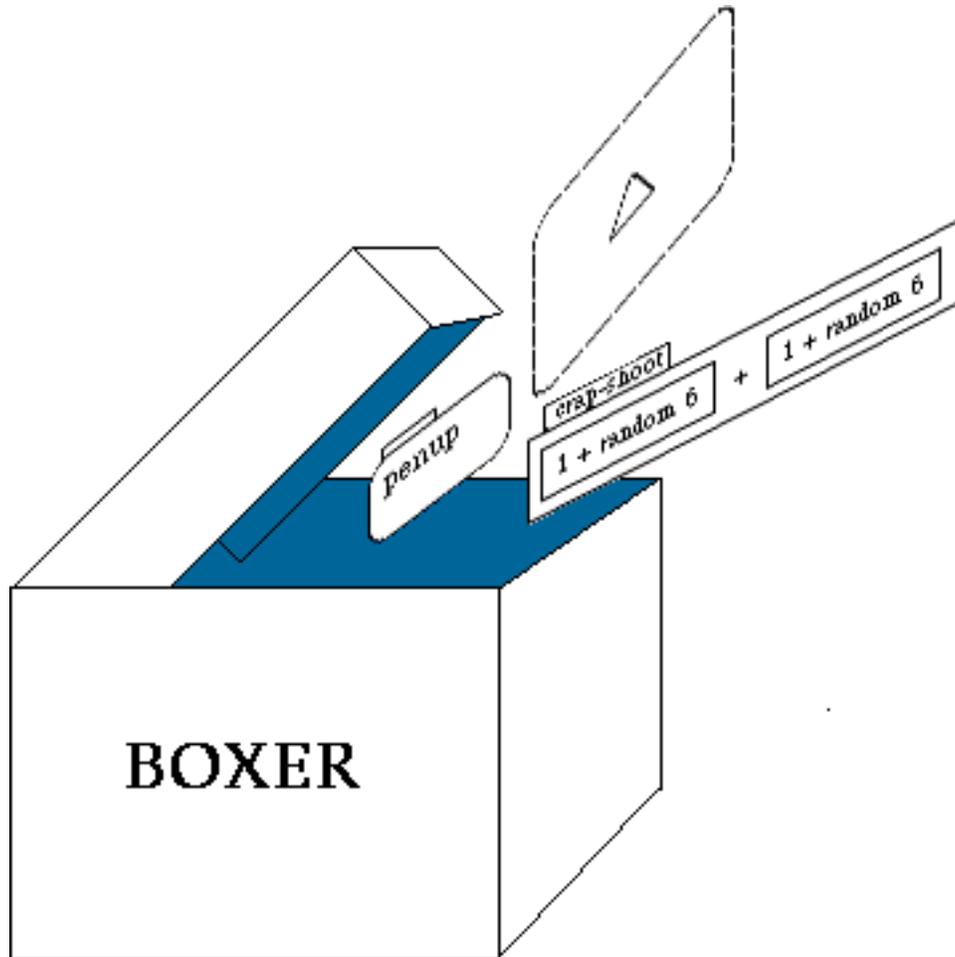


BOXER MANUAL V2



"A box is all things."
Ancient Proverb

This manual is an abridged version of the on-line Boxer Command Manual. The latter is generally more up to date, more complete, and contains working examples. Use these manuals in conjunction with the Boxer Structures document, which provides an explanation of the general Boxer forms and mechanisms on which the commands listed here operate.

Compiled & Edited by
Rafael Granados
September 1994
Revised, December, 1999

Berkeley Macintosh Boxer Legal Notification

Berkeley Boxer and related documentation are ©; Copyright 1999, Andrea A. diSessa and Edward H. Lay (“the authors”). Prior copyright by the Regents of the University of California has been assigned to the authors. The University of California makes no representation with respect to Berkeley Boxer and assumes no liability whatsoever. Berkeley Boxer may contain code from the original Boxer development, copyright of MIT. All rights reserved, except for non-exclusive license expressly set forth below.

Permission to use, copy, and distribute this software and its documentation (with the exclusion of Boxer Structures) for educational, research and non-profit purposes, without fee, and without a written agreement is hereby granted, providing at the above copyright notice and the following seven paragraphs appear in all copies.

Contact Andrea A. diSessa (1053 Park Hills Rd., Berkeley, CA 94708, 510-845-6561) concerning permission to use Boxer for commercial purposes, or to incorporate this software into existing or new commercial products.

IN NO EVENT SHALL THE AUTHORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN “AS IS” BASIS, AND THE AUTHORS HAVE NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Berkeley Boxer for the Macintosh is implemented with Digitool Macintosh Common Lisp (“MCL”).

DIGITOOL, INC. (“DIGITOOL”) AND ITS LICENSOR MAKE NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING DIGITOOL MCL. DIGITOOL AND ITS LICENSOR DO NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF MCL IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

IN NO EVENT WILL DIGITOOL, ITS LICENSOR, THEIR DIRECTORS, OFFICERS, EMPLOYEES OR AGENTS BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGE (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOST OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OR INABILITY TO USE MCL, EVEN IF DIGITOOL, AND/OR ITS LICENSOR HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

BECAUSE SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. Digitool’s and its licensor’s liability to you for actual damages for any cause

whatsoever, and regardless of the form of action (whether in contract, tort (including negligence), product liability or otherwise) will be limited to \$50.

Table Of Contents

BOXER MANUAL V2.....	1
CHAPTER 1.....	1
CONTROL STRUCTURE	1
1.1 SELECTION	3
1.2 REPETITION.....	6
1.3 EVALUATION	8
CHAPTER 2.....	12
<i>DATA MANIPULATION</i>	12
2.1 DATA COMPARISON	16
2.2 DATA INFORMATION: BOX SIZE	19
2.3 DATA INFORMATION: CONTENTS TYPE.....	21
2.4 DATA INFORMATION: CONTENTS LOCATION	23
2.5 DATA INFORMATION: ACCESS BY ITEM	25
2.6 DATA INFORMATION: ACCESS BY ROW.....	28
2.7 DATA INFORMATION: ACCESS BY COLUMN	31
2.8 DATA INFORMATION: ACCESS BY ROW & COLUMN.....	33
2.9 DATA INFORMATION: ACCESS BY NAME	34
2.10 DATA CONSTRUCTION	37
2.11 DATA CONVERSION.....	40
2.12 CONVERSION BY ITEM	42
2.13 CONVERSION BY ROW.....	45
2.14 CONVERSION BY COLUMN.....	48
2.15 CONVERSION BY ROW & COLUMN	51
2.16 CONVERSION BY BOX.....	52
2.17 CONVERSION BY CHARACTER.....	55
2.18 ACCESSING NAME INFORMATION	56
CHAPTER 3.....	59
<i>GRAPHICS</i>	59
3.1 MOVES AND TURNS.....	65
3.2 SETTING POSITION AND HEADING	67
3.3 HIDE & SHOW	70
3.4 PEN.....	72
3.5 CLEARING GRAPHICS.....	84
3.6 GRAPHICS SIZE & MODE	85
3.7 SNAPPING & CHANGING	87

3.8 COLOR.....	89
3.9 SPRITE INFORMATION & PROPERTIES.....	102
3.10 UPDATE PROPERTIES.....	109
3.11 OTHER INFORMATION.....	110
3.12 SPRITE SIZE, SHAPE & HOME.....	112
3.13 MOUSE INPUT & CLICKS.....	114
3.14 MOUSE POSITION.....	116

CHAPTER 4..... 119

<i>ARITHMETIC & LOGIC</i>	119
4.1 ARITHMETIC OPERATORS & FUNCTIONS.....	123
4.2 COMPARISON OPERATORS & FUNCTIONS.....	126
4.3 NUMBER TYPE.....	129
4.4 VALUE INFORMATION.....	131
4.5 OTHER NUMERIC FUNCTIONS.....	133
4.6 EXPONENTIAL & LOG FUNCTIONS.....	136
4.7 TRIGONOMETRIC FUNCTIONS.....	138
4.8 NUMBER CONVERSION TO INTEGERS.....	140
4.9 NUMBER PRINTING CONTROL.....	143
4.10 LOGIC.....	144

CHAPTER 5..... 146

<i>Triggers</i>	146
5.1 TRIGGERS.....	148

CHAPTER 6..... 152

<i>MISCELLANEOUS COMMANDS</i>	152
6.1 MISCELLANEOUS COMMANDS.....	154

CHAPTER 7..... 155

<i>ENVIRONMENT: INPUT & OUTPUT</i>	155
7.1 OUTPUT.....	158
7.2 INPUT: KEYSTROKE BINDING.....	160
7.3 INPUT: MOUSE BINDING.....	162
7.4 INPUT: MOUSE POLLING.....	164
7.5 REQUEST FOR INPUT HANDLING.....	170

CHAPTER 8..... 172

<i>ENVIRONMENT: FILE COMMANDS</i>	172
8.1 STANDARD FILE COMMANDS.....	175

CHAPTER 9..... 183

ENVIRONMENT: KEYSTROKES AND EDITING 183
9.1 MOVING IN BOXER 184
9.2 MAKING BOXES 185
9.3 CUT AND PASTE..... 186
9.4 OTHER KEYSTROKES 187
9.5 KEYS THAT PRINT 189
9.6 CURSOR LOCATION 192
9.7 BOX-SIZING 193

CHAPTER 10..... 194

ENVIRONMENT: NETWORKING 194
10.1 NET BOX FILE/DIRECTORY COMMANDS..... 197
10.2 MAIL 202

CHAPTER 11..... 205

ENVIRONMENT: MISCELLANEOUS..... 205
11.1 SYSTEM-PREFERENCES 209
11.2 RESULT APPEARANCE 210
11.3 EVALUATOR-SETTINGS..... 211
11.4 GRAPHICS SETTINGS 215
11.5 EDITOR SETTINGS 218
11.6 FILE SYSTEM SETTINGS 223
11.7 NETWORK SETTINGS..... 224
11.8 COMMUNICATIONS SETTINGS 225
11.9 MISCELLANEOUS SYSTEM COMMANDS 226
11.10 EXTENSIONS 233
11.11 SERIAL PORT: SETUP 235
11.12 SERIAL PORT: READING 237
11.13 SERIAL PORT: WRITING..... 239
11.14 SERIAL PORT: PREFERENCES 240

CHAPTER 1

Control Structure

In general, Boxer executes procedures one command at a time, left to right, top to bottom -- just like reading text. However, for many reasons, you may want to change this behavior. The two most frequent non-linear control patterns are *repetition*, in which a segment of code is evaluated several times, and *selection*, in which some code may be executed, or not, depending on circumstances. **if** is the main control *selection* command, and **repeat** is the main repeater.

Boxer also includes other control options. You may choose to execute data in Boxer using the **run** command, effectively changing a data box into a *doit* box. You may also choose to deliberately **stop** some currently running procedure. Or you may change where execution takes place with **tell (ask)**

1. SELECTION

if
ifs
unless
when

These are *conditional* commands that execute some code, or not, depending on some condition.

2. REPETITION

repeat
loop
for-each-item
for-each-row

These are *iterators* that run some code over and over.

3. EVALUATION

stop
stop-loop
tell
ask
run
@
^

These commands all change the normal modes by which Boxer executes code. They change place of execution (**tell**); they change where Boxer looks to find a variable (**tell** and **^**); they cause data to be executed like a procedure (**run**); and they allow you to construct command lines on the fly out of computed information or information in data boxes (**@**).

1.1 SELECTION

if

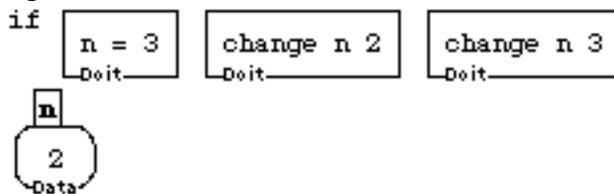
Syntax

if <condition> <action> <optional-alternate-action>

Description

Runs <action> or <optional-alternate-action> depending on whether <condition> is **true** or **false**. If the <optional-alternate-action> is not present, nothing happens when the condition is **false**. Always enclose actions in boxes unless they are single words. Note that **if** can be used as a function (that is, to return a value) as well as a command. This happens if expressions are used in place of <action> and <optional-alternate-action>. The value returned by **if** is the result produced by whichever expression is evaluated. It is best to put the <action> and <optional-alternate-action> in doit boxes: for clarity, for inspectability (some ifs might be long -- a doit box can be shrunk to hide it for easy inspection of the rest of the command), and to avoid bugs due to Boxer reading your **if** in a different way than you intended.

Example



ifsSyntax

```

ifs <condition 1> <action 1>
      <condition 2> <action 2>
      .
      .
      else <optional-alternate-action>
Data

```

Description

A command to handle multiple conditions. It takes as input a data box, with each line having (1) a condition and (2) an action (no alternates). **ifs** executes the action for the *first true statement only*. You can have as many or as few condition-action pairs as you want. The **else** clause is optional. If you include an **else** and none of the conditions are true, the <optional-alternate-action> gets run. Like **if**, the actions or alternative-action may end with a value, which will be returned as the value of the whole **ifs**. However, unlike **if**, the <actions> need not be a single token (word or box), so you need not include multiple command actions in a doit box if you chose not to.

Example

```

ifs
  1 < 2 beep beep beep
  else beep
Data

```

unlessSyntax

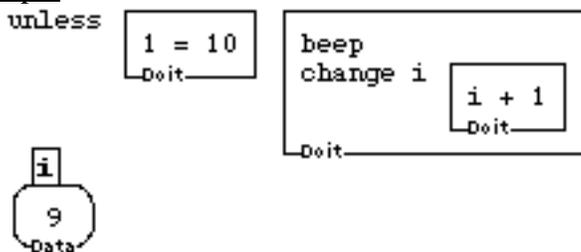
```

unless <condition> <action>

```

Description

This command will execute the action unless the condition is true. It is the same as saying *if not*, except that you can't have an alternative action. You need not enclose <action> in a box; unlike **if**, <action> need not be a single word or box.

Example

when

Syntax

when <condition> <action>

Description

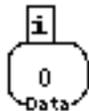
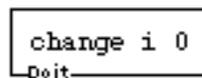
This command is the same as **if**, except that you can't use it to specify an alternate action. It's just a nice, natural-language way to say **if**, and it allows you to type an <action> of several words without placing them in a doit box, as with **unless**.

Examples

```
when done? beep beep beep
```



```
when i = 9
```



1.2 REPETITION

repeat

Syntax

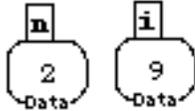
repeat <number> <action>

Description

This command takes two inputs, a number and an action. The number specifies how many times the action should be executed. The first input is rounded to the nearest integer, in case it is not already an integer.

Example

```
repeat n
  change i i + 1
  beep
  redisplay
Doit
```



loop

Syntax

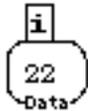
loop <action>

Description

This command creates an infinite loop. It repeats over and over until you manually stop, use a **stop** or **stop-loop** command.

Example

```
loop
  change i
    i + 1
  beep
  redisplay
Doit
```



for-each-item

Syntax

for-each-item <variable> <data box> <action>

Description

This command first changes the value of the specified variable to be the first item in the input box, and then executes the action (which may involve the value of the variable). Then it changes the value of the variable to the second item and executes the action, and so on, once for each item.

Example



for-each-row

Syntax

for-each-row <variable> <data box> <action>

Description

This command is the same as **for-each-item**, but with rows. It sets the value of the specified variable to the first row of the input box and executes the action. Then it sets the value to the second row and executes the action, and so on, once for each row.

Example



1.3 EVALUATION

stop

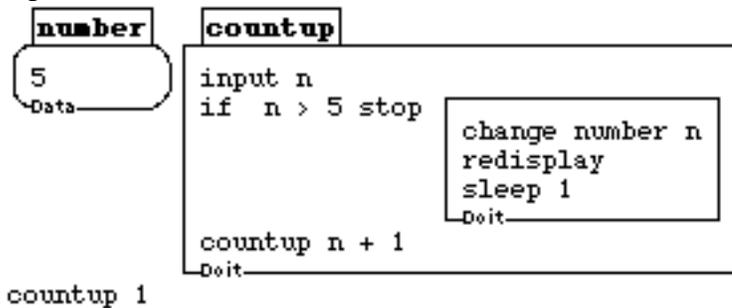
Syntax

stop <procedure-name>

Description

This command stops a procedure. It is an unusual Boxer primitive in that it has an optional input: if you specify the name of a procedure, it stops that procedure (if it is running!). If you don't specify the name of a procedure, it stops whatever box it is in. Note that **stop** returns a value if one is computed as the last command before **stop**. Because of this, **stop** is as powerful as catch and throw in Lisp or Logo.

Example



stop-loop

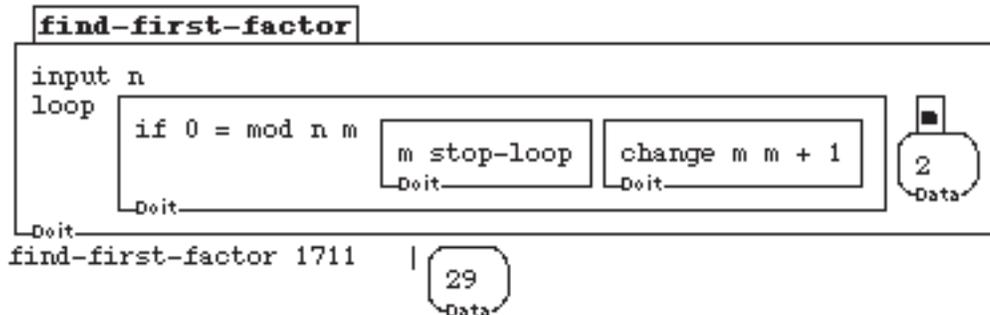
Syntax

stop-loop

Description

This command stops the currently running loop (see **loop**). Loops may be nested within one another, and **stop-loop** will stop the innermost loop. Note that **stop-loop** will not work with named user procedures (see **stop** for stopping named procedures.)

Example



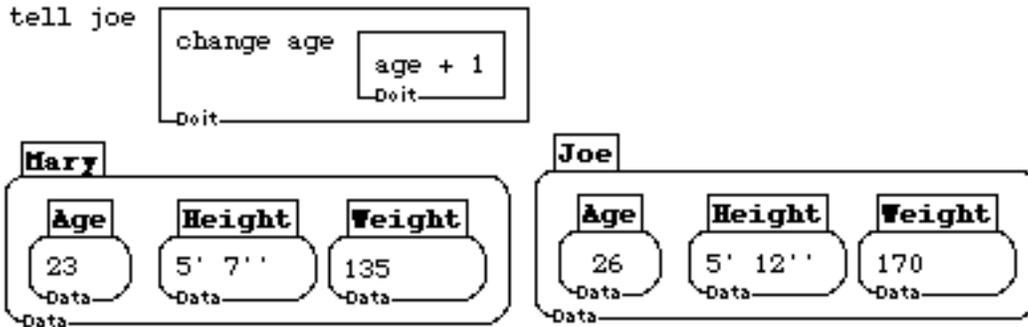
tellSyntax

tell <box> <action or variable name>

Description

tell takes the <action or variable name> as a message and executes it in inside <box>.

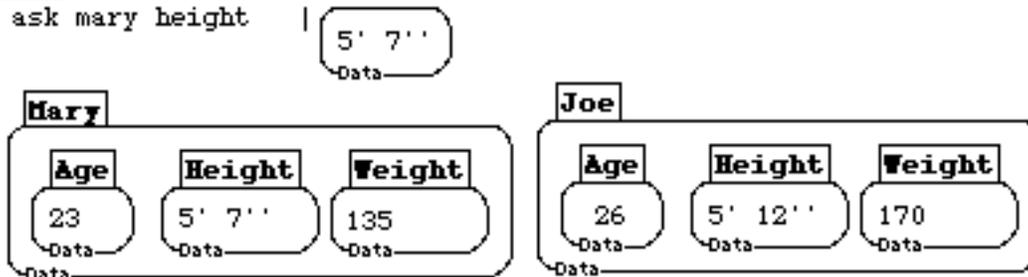
Usually, when you execute an action or the name of a variable in Boxer, it looks for that action or variable starting in the box you're currently in. If it doesn't find the variable, it looks in the box that contains the one you're in, and so on outward. With **tell** you can direct Boxer to execute the action or find the variable in a particular named box. So **tell** asks the box you specify to execute the action or find the variable as if you had done this from within that box. **tell** <box> <message> operates as if you moved your cursor into <box>, typed the <message> and pressed the doit key. Except it doesn't leave any "mess" (typing) in the <box>, and it returns any results back to the place that **tell** was executed. **ask** is a synonym for **tell**. You can think of **ask** as a polite form, or a form better suited to *asking* for information than *telling* a box (to execute a variable name or procedure) for some information.

Example**ask**Syntax

ask <box> <action or variable name>

Description

ask is a synonym for **tell**.

Example

runSyntax**run** <input>Description

run takes its input (a data box) and executes it as a procedure. That is, it effectively turns a data box into a doit box. It will return a value if running its input results in a value.

Example

```
run
  repeat 3 beep
```

The diagram shows a rectangular box with rounded corners. Inside the box, the text 'repeat 3 beep' is written. Below the box, the word 'Data' is written, indicating the type of the box.

@Syntax**@** <data-box>Description

@ in front of a data box, or in front of a procedure that will provide it with a data box, tells Boxer to strip away the box and use what's inside. You can read it as "*the contents (insides) of ...*" It is handy in case you want some text to appear in a line of code, but that text needs to be computed or exists in a data box. **@** was designed to be used with **build**, but was extended to be useful in "*computing lines of code to execute.*" **@** is very special in Boxer. It does not return a result in the usual way. Instead, Boxer replaces every **@** with the result of running what follows it, and then executes the resulting line of code. It is similar to **unbox**, except more powerful. (**unbox** always returns a value in a data box, hence that value never gets executed, as it does with **@**.)

Example:

```
@command @variable 7
```

The diagram illustrates the execution of the code '@command @variable 7'. It shows three data boxes in a row. The first box is labeled 'z' and contains the value '7'. The second box is labeled 'variable' and contains the value 'z'. The third box is labeled 'command' and contains the text 'change'. Each box has the word 'Data' written below it.

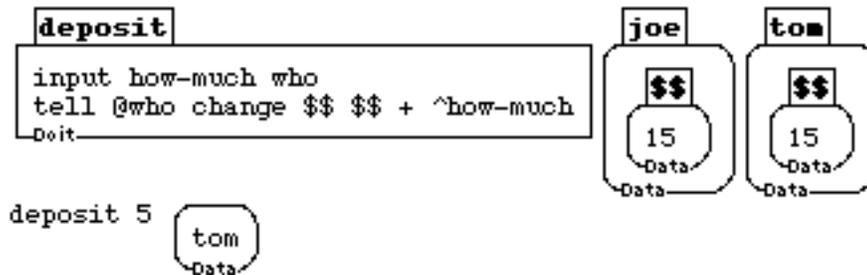
^

Syntax

^ <a name>

Description

^ is used before a name. It means "use the box with this name from the place where the last **tell** (**ask**) was executed." This is an important feature when you want to send along the value of a variable, or a certain procedure, in the message *told* to a box that ordinarily wouldn't have access to the variable or procedure. Read ^ as "*this* <variable, procedure>" or as "the local <variable, procedure>". In the example, the **tell** causes **change** to be executed inside **tom**, who does not have access to **how-much** (inside **deposit**) unless ^ is used.

Example

CHAPTER 2

Data Manipulation

This chapter discusses the manipulation of data. In BOXER, the primitive data type is the data box (Figure 1a). A named data box is a variable (Figure 1b). Data boxes can be filled with text, numbers, or other boxes (Figure 1c). By nesting data boxes, you can create hierarchical data structures. The parts of a hierarchical data structure can be individually named (Figure 1d. See also "access-by-name" in "data-access"). There also exists a mechanism for creating arbitrarily connected networks of boxes (see Boxer Structures on Ports).



Data is passed into a procedure in a data box, and returned from a procedure in a data box (see procedure inputs and outputs). Data may be treated as a list (one-dimensional), as a sequence of rows, as a sequence of columns, or as an array (two-dimensional). There are accessors (that fetch part of the data) and mutators (that change part of it) for each of these modes.

Graphics boxes (see sprite graphics) have most of the behaviors of data boxes, except they have a graphical presentation on which pictures can be shown and in which sprites can wander and draw. Numbers are a special kind of data that do not have to be typed in a data box. But executing a number will return the number in a data box. Not having to type numbers in boxes to designate them as data is merely a convenience. **true** and **false** are special in exactly this way.

The following sections present examples of commands for comparing boxes, getting information from data structures, accessing data from data structures, constructing data, and for accessing information on commands and on things named by the user. The information gathered through these commands can be used in several ways, such as to check if a piece of data is of the proper type (e.g., **if number?** X ...), as arguments to selectors, as termination conditions (**if empty?** X **stop**), as parameters for iteration (**repeat number-of-items** X ...), or just to let you know what's there.

2.7 DATA INFORMATION: ACCESS BY COLUMN

first-column	A box may be considered a sequence of columns, and there are corresponding accessors and mutators that are very similar to those for a box considered as a list, (i.e., as a sequence of items). Columns are generally less useful than rows, so these commands are more for completeness than general use. Note that a column is defined by the number of the item in each row -- spacing is irrelevant.
column	
last-column	
butfirst-column	
butcolumn	
butlast-column	
columns	

2.8 DATA INFORMATION: ACCESS BY ROW & COLUMN

rc	This command views the box as a two dimensional array of elements.
-----------	--

2.9 DATA INFORMATION: ACCESS BY NAME

ask (tell)	Two methods are provided for getting the pieces of a box by name. Ask (synonym tell) uses message passing. "Dot notation" allows you to specify a chain of boxes within boxes by name.
.	
self	
superior	

2.10 DATA CONSTRUCTION

build	Boxer basically only has one very powerful and graphical means of constructing complex data objects out of pieces: build . With build , you may also create newly named boxes whose names are computed. Join-bottom and join-right are convenient abbreviations. Boxify and datafy are also conveniences.
join-bottom	
join-right	
boxify	
datafy	

2.11 DATA CONVERSION

change	These commands mutate or change boxes.
retarget	

2.12 CONVERSION BY ITEM

change-item	These commands treat the box as a list of items. This class is frequently used to fiddle with parts of a box one at a time, as a program progresses. Contrast accessors like item , first , which are usually used in list processing that systematically moves through a box
change-last-item	
insert-item	
append-item	
delete-item	
delete-last-item	
delete-items	

2.13 CONVERSION BY ROW

change-row	These commands treat the box as a list of rows. This class of functions is frequently used to fiddle with parts of a box one at a time, as a program progresses. Contrast accessors like row , first-row , which are usually used in list processing that systematically moves through a box
change-last-row	
delete-row	
delete-last-row	
delete-rows	
insert-row	
append-row	

2.14 CONVERSION BY COLUMN

change-column	Similar to row and item mutators, except oriented to columns. These are not frequently used, since columns are awkward visually in many instances.
change-last-column	
delete-column	
delete-last-column	
delete-columns	
insert-column	
append-column	

2.15 CONVERSION BY ROW & COLUMN

change-rc	These change part of a box, indexed by row and column (array) specification.
insert-rc	
delete-rc	

2.16 CONVERSION BY BOX

boxify	These are functions that do some simple conversions on data, like putting it in a box, getting a port to it, or making a copy of it. Boxer gets character data by taking words apart into letters, or, uses characters to build words (see directly below). This is unlike Logo.
datafy	
copy	
port-to	
unbox	

2.17 CONVERSION BY CHARACTER

letters	Characters are handled in Boxer by converting words into a series of letters, doing any operations on the letters (treated as individual items) and converting back to a word. letters converts to a list of letters. word converts back to a word. You are best off processing many words at the word level first, converting only one word at a time to letters.
word	

2.18 ACCESSING NAME INFORMATION

name-help	These commands get information about the names of things or ab things that are named. These include Boxer commands as well as boxes named by the user.
name?	
local-name?	
name-in-box?	
names	
top-level-name?	
target-name	

2.1 DATA COMPARISON

=

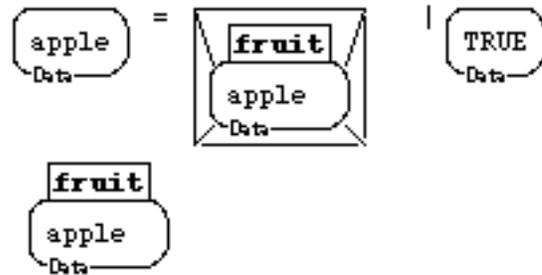
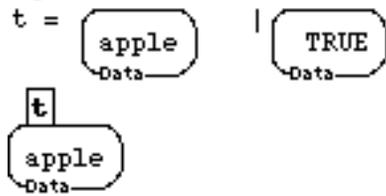
Syntax

<box1> = <box2>

Description

= checks if <box1> and <box2> have identical contents. It returns **true** or **false**. It works on numbers as well as boxes, and it treats decimals, integers and fractions as equivalent types. Note that = ignores "portness" and just checks the contents of the target box. In the last example (below), a port to a box containing "apple" is considered equal to a box containing "apple".

Examples



equal?

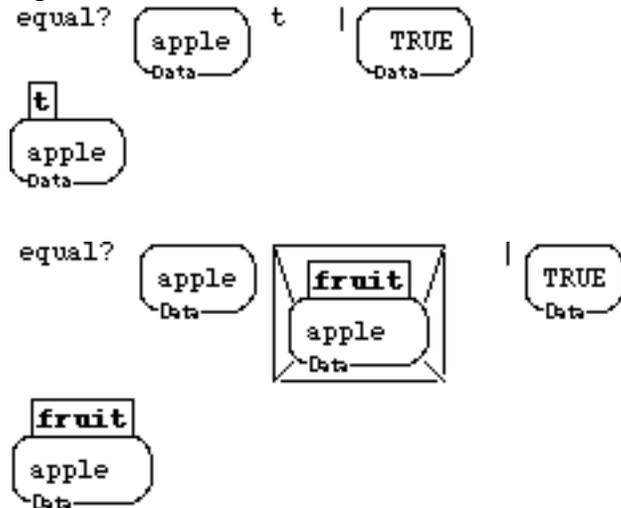
Syntax

equal? <box1> <box2>

Description

equal? checks if <box1> and <box2> have identical contents. It returns **true** or **false**. It also works on numbers as well as boxes, and it treats decimals, fractions and integers as the same if they are numerically equal. Note that **equal?** ignores "portness" and just checks the contents of the target box as is illustrated by the second example.

Examples



alpha>

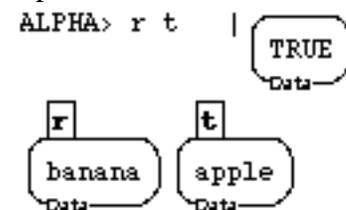
Syntax

alpha> <box1> <box2>

Description

This command checks if <box1> comes alphabetically after <box2>. It returns **true** or **false**. Spaces, boxes and special characters might have surprising behavior. Upper case letters come before lower case, and, generally, symbols come before upper case.

Example



alpha<

Syntax

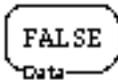
alpha< <box1> <box2>

Description

This command checks if <box1> comes alphabetically before <box2>. It returns **true** or **false**. Spaces, boxes and special characters might have surprising behavior. Upper case letters come before lower case, and, generally, symbols come before upper case.

Example

```
ALPHA< r t |
```



2.2 DATA INFORMATION: BOX SIZE

number-of-items

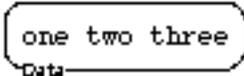
Syntax

number-of-items <box>

Description

Returns the number (an integer ≥ 0) of items in the input <box>.

Example

number-of-items  | 

number-of-rows

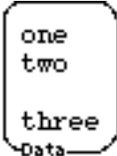
Syntax

number-of-rows <box>

Description

Returns the number (an integer ≥ 1) of rows in the input <box>.

Example

number-of-rows  | 

number-of-columns

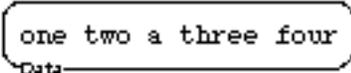
Syntax

number-of-columns <box>

Description

Returns the number (an integer ≥ 0) of columns in the input <box>.

Example

number-of-columns  | 

count

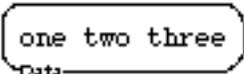
Syntax

count <box>

Description

Abbreviation for number-of-items for *Logo* compatibility. Returns an integer ≥ 0 in a box.

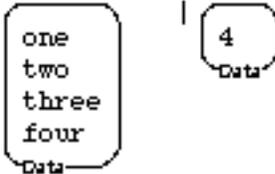
Example

count  | 

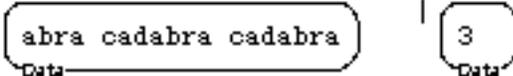
lengthSyntax**length** <box>DescriptionAbbreviation for number-of-items. Returns an integer ≥ 0 in a box.Example

length 

heightSyntax**height** <box>DescriptionAbbreviation for number-of-rows. Returns an integer ≥ 1 in a box.Example

height 

widthSyntax**width** <box>DescriptionAbbreviation for number-of-columns. Returns an integer ≥ 0 in a box.Example

width 

2.3 DATA INFORMATION: CONTENTS TYPE

empty?

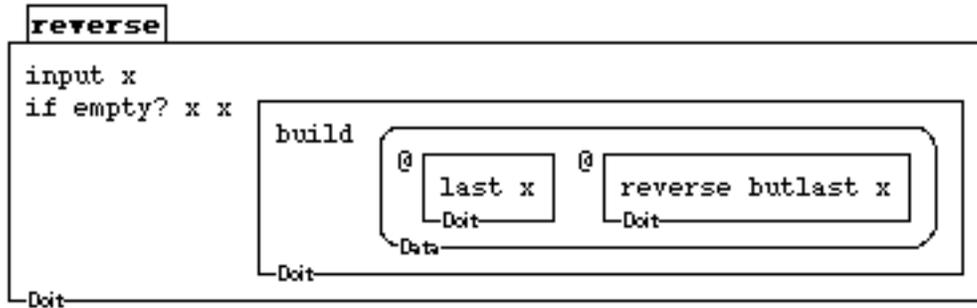
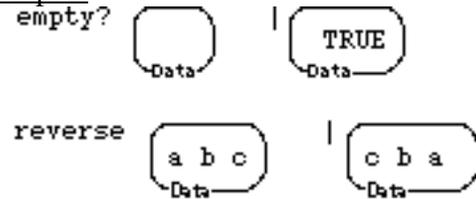
Syntax

empty? <box>

Description

Asks if its input is an empty <box>. It returns **true** or **false**. A box is empty if all of its lines are empty. The last example below shows a typical use of **empty?**, i.e., to reverse a list.

Examples



number?

Syntax

number? <box>

Description

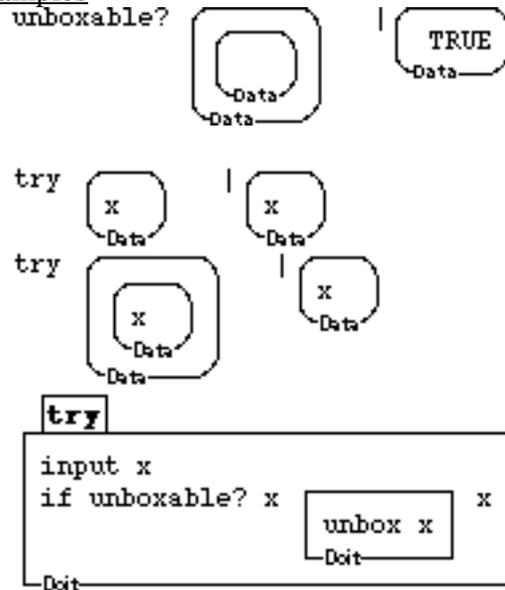
Returns **true** precisely when the input is a number.

Example

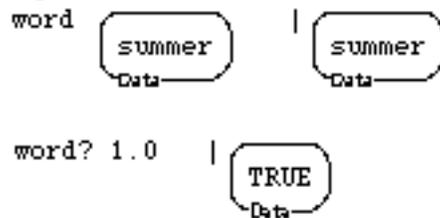


unboxable?Syntax**unboxable?** <box>Description

Returns **true** if and only if its input contains precisely one box, so that you may apply **unbox** to it. This command is useful to find out if a box can be safely unboxed with the command **unbox** (to result in another data box).

Examples**word?**Syntax**word?** <box>Description

Returns **true** or **false** depending on whether the input is a box containing a single word. Numbers count as words.

Examples

2.4 DATA INFORMATION: CONTENTS LOCATION

member?

Syntax

member? <item> <list>

Description

This returns **true** or **false** depending on whether <item> is in <list> or not. The <item> may actually be a sequence, and member? checks whether the full sequence appears in <list>.

Example

```
member? peach apple pear | TRUE
           peach orange
```

item-numbers

Syntax

item-numbers <template> <list>

Description

This returns a list of numbers that tells you at which positions <template> is found in <list>. A typical use of this command is to give you information as to the location of occurrences of a particular item in a list.

Example

```
item-numbers abc the word abc has no meaning | 3
```

row-numbers

Syntax

row-numbers <template> <box>

Description

This returns a list of numbers that tells you at which rows <template> is found in <box>. A typical use of this command is to give you information as to the location of occurrences of a particular item in different rows.

Example

```
row-numbers abra abra | 2
             cadabra
```

column-numbers

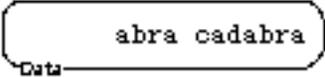
Syntax

column-numbers <template> <box>

Description

This returns a list of numbers that tells you at which columns <template> is found in <box>. A typical use of this command is to give you information as to the location of occurrences of a particular item in different columns.

Example

```
column-numbers   | 
```

2.5 DATA INFORMATION: ACCESS BY ITEM

first

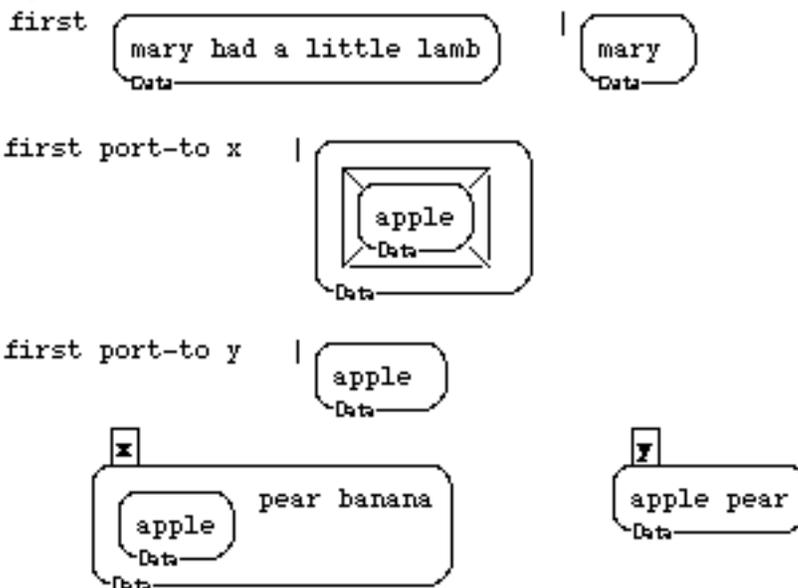
Syntax

first <box>

Description

Returns the first item in <box> (and that item will be enclosed in a box). Note that all data accessors preserve as much "portness" as possible. So if the input to **first** is a port and the first item is a box, then the result will be a box containing a port to that box (see second example below). If the first element of a port which is the input to **first** is not a box, Boxer cannot return a port to it, of course, and so you get a copy (see third example below). If the first element of the box is a port, naturally Boxer returns a box containing a duplicate port.

Examples



item

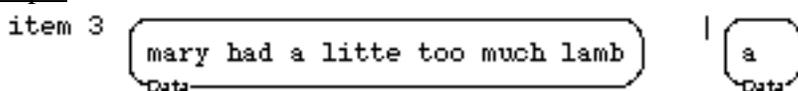
Syntax

item <n> <box>

Description

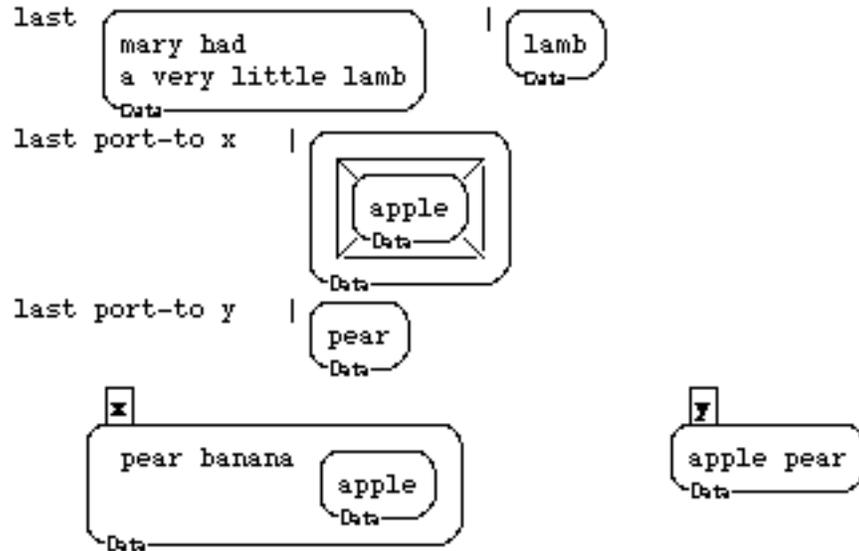
Returns item number <n> of <box> enclosed in a box. The valid range of item numbers is from 1 up to the number of items in the box. Numbers outside this range return an error box. This command preserves as much "portness" as possible. See documentation for **first**.

Example

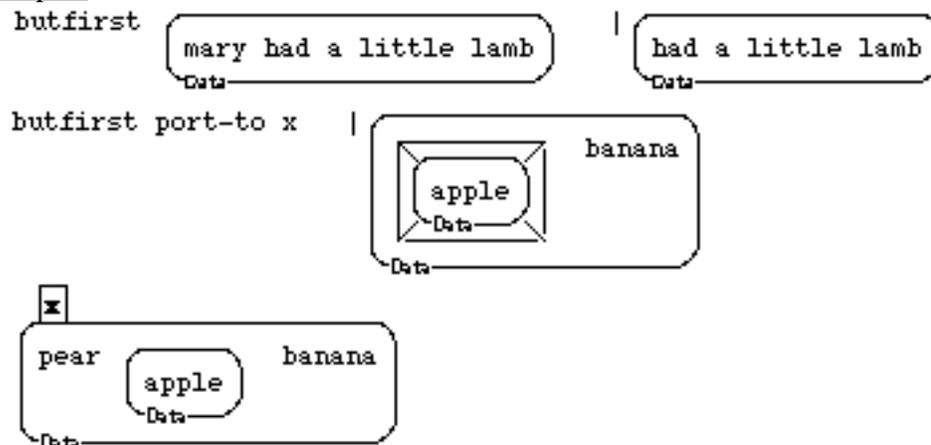


lastSyntax**last** <box>Description

Returns the last item in <box> enclosed in a box. All data accessors preserve as much "portness" as possible.

Examples**butfirst**Syntax**butfirst** <box>Description

Returns a copy of the box with the first item missing. If the box contains only one item, then **butfirst** returns an empty box. Note that all data accessors preserve as much "portness" as possible.

Examples

butitemSyntax**butitem** <n> <box>Description

Returns a copy of <box> with item number <n> removed. If n is ≤ 0 or $>$ the number of items in the box, then an error is signaled. Note that all data accessors preserve as much "portness" as possible.

Example

```
butitem 4
```


butlastSyntax**butlast** <box>Description

Returns a copy of the <box> with the last item missing. If the box contains only one item, then **butlast** returns an empty box. Used like **butfirst**, except in cases an inverted order is preferred for some reason. All data accessors preserve as much "portness" as possible.

Example

```
butlast
```


itemsSyntax**items** <start> <end> <box>Description

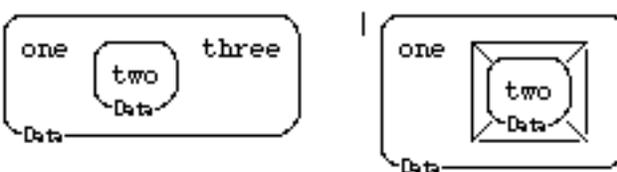
Returns a box containing all the items in <box> from <start> to <end>. <start> and <end> are integers. As usual for data selectors, items returns a port to a box, where possible, when a port is given as input.

Examples

```
items 3 4
```



```
ITEMS 1 2 port-to
```



2.6 DATA INFORMATION: ACCESS BY ROW

first-row

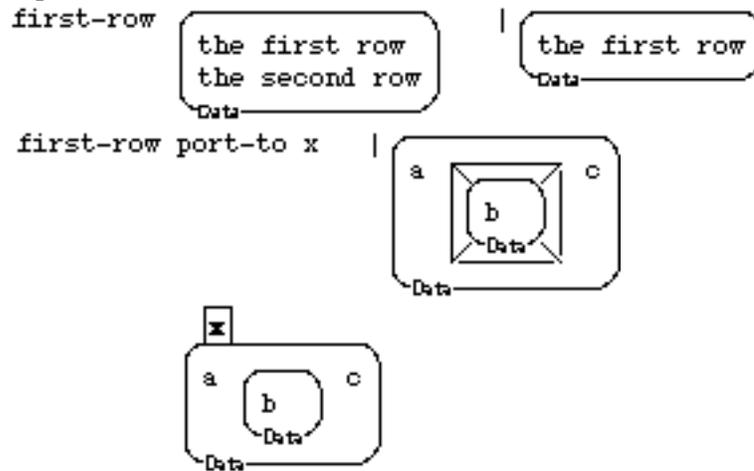
Syntax

first-row <box>

Description

Returns a box whose contents are the same as the contents of the first row of the input. If the input's first row is empty, an empty box will be returned. See **first** for documentation on typical use. **first-row** preserves "portness" as much as possible.

Examples



row

Syntax

row <n> <box>

Description

Selects row number <n> of <box> and returns a box whose contents are that row. Rows are ordered from top to bottom, and start at one and end up at the **number-of-rows** of the box. Row numbers outside this range result in an error box being returned. If the number is not an integer, an error is returned. See **item** for more comparable documentation.

Preserves "portness" as much as possible.

Example

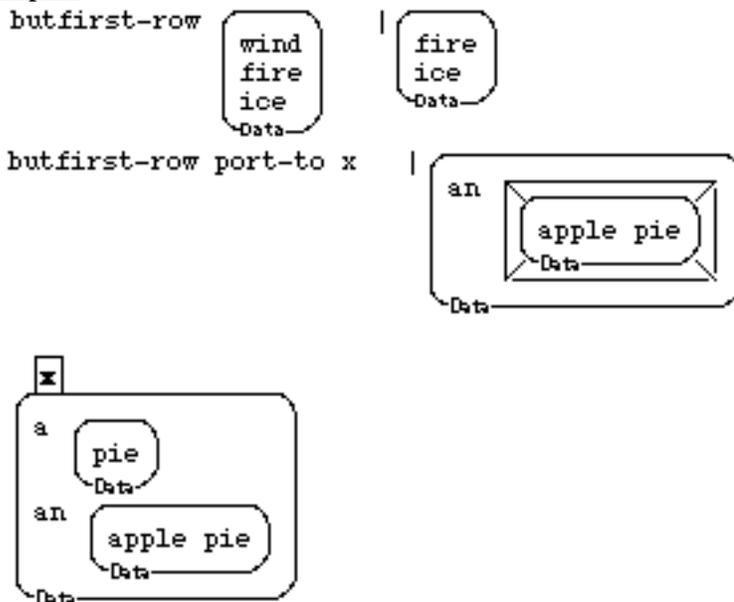


last-rowSyntax**last-row** <box>Description

Returns a box whose contents are a copy of the last row of the input. If the input's last row is empty, an empty box will be returned. See **first** for more extensive documentation on typical use. Preserves "portness" as much as possible.

Example**butfirst-row**Syntax**butfirst-row** <box>Description

Returns a copy of its input with the first row missing. Preserves "portness" as much as possible.

Examples

but-rowSyntax**but-row** <n> <box>Description

Returns a copy of <box> with row <n> missing. Preserves "portness" as much as possible.

Example

```
butrow 2
```

wind
fire
ice
Data

|

wind
fire
ice
Data

butlast-rowSyntax**butlast-row** <box>Description

Returns a copy of its input with the last row missing. Preserves "portness" as much as possible.

Example

```
butlast-row
```

wind
fire
ice
Data

|

wind
fire
Data

rowsSyntax**rows** <start> <end> <box>Description

Returns rows numbered <start> to <end>. <start> and <end> must be in order and between 1 and the **number-of-rows** in <box>. Preserves "portness" as much as possible.

Example

```
rows 3 4
```

wind
fire
ice
Data

|

fire
ice
Data

butfirst-columnSyntax**butfirst-column** <box>Description

Returns all but the first column of the input. Preserves "portness" as much as possible.

Example

```
butfirst-column
  a b c d
  e f g h
  Data_____
```

```
|
  b c d
  f g h
  Data_____
```

butcolumnSyntax**butcolumn** <n> <box>Description

Returns a copy of the <box> without column number <n>. In Boxer columns are counted logically; spacing doesn't matter. Preserves "portness" as much as possible.

Example

```
butcolumn 3
  a b c d
  e f g h
  Data_____
```

```
|
  a b d
  e f h
  Data_____
```

butlast-columnSyntax**butlast-column** <box>Description

Returns a copy of the input box without the last column. The last column (removed) is the last item of each row, independent of spacing. Preserves "portness" as much as possible.

Example

```
butlast-column
  a b c d
  e f g h
  Data_____
```

```
|
  a b c
  e f g
  Data_____
```

columnsSyntax**columns** <start> <end> <box>DescriptionReturns rows numbered <start> to <end>. <start> and <end> must be in order and between 1 and the **number-of-columns** in <box>. Preserves "portness" as much as possible.Example

```
columns 2 3
  a b c d
  e f g h
  Data_____
```

```
|
  b c
  f g
  Data_____
```

2.8 DATA INFORMATION: ACCESS BY ROW & COLUMN

rc

Syntax

rc <row> <column> <box>

Description

Returns a box containing the element that is at row number <row> (starting at 1 and moving from top to bottom) and column number <column> (also starting at 1 and moving from left to right) of the input. If there is no element at the specified row and column numbers, then an error is signaled. In Boxer columns are counted logically; spacing doesn't matter. Blank rows, however, count as a row. This command preserves "portness" as much as possible.

Example

```
rc 1 2
```

2.9 DATA INFORMATION: ACCESS BY NAME

ask

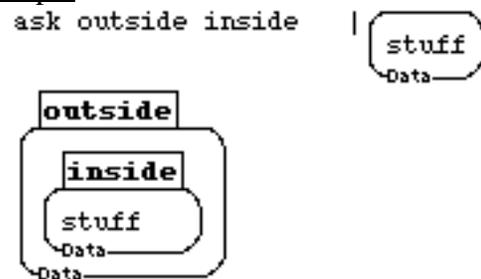
Syntax

ask <box> <whatever>

Description

Returns the box named <name> from inside the box named <box>. Can be chained as in **ask** <box1> **ask** <box2> <name>. **tell** is a synonym for **ask**. **ask** is Boxer's standard message passing command. The model is that " **ask** <who> <whatever>" takes Boxer into <who>, then types <whatever> and executes it. Any result from executing <whatever> is returned from **ask**. **ask** together with the usual variable behavior (a name fetching a copy of the data named) allows the general hierarchical data access method to work. **ask** is more general and powerful than its use as documented here. See **tell** and **ask** (message-passing) in Boxer Structures. Fine point: " **ask** " is preferable to " **tell** " in this case, even though the two are synonyms, because **tell** sounds like it wants an action, not just some data.

Example



.(dot)

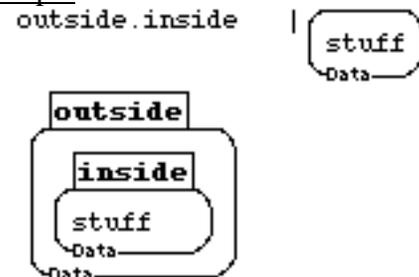
Syntax

<name>.<name2>

Description

x.y denotes the part of x named y. May be chained as in "x.y.z". This denotes the z part of the y part of box x. Think of a dotted name as specifying a path down through a series of named boxes to the part needed. It is difficult (though possible) to "compute" the names involved. box.@name does not work. In this case it is better to use **ask**. Dot notation is more compact and sometimes easier to read, but not as flexible as **ask**.

Example



selfSyntax**self**Description

Returns a port to the place where execution last started. Or, if a **tell** or **ask** was executed, it returns a port to the place told (asked). This command is useful if you want to know where execution was started, e.g., it might be started by a graphics or mouse click. See **mouse-box-on-click** for a similar function.

Example

```
ask alan self
```

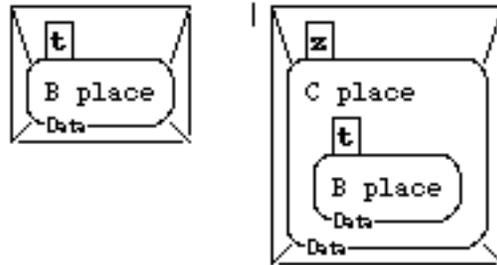


superiorSyntax**superior** <box>Description

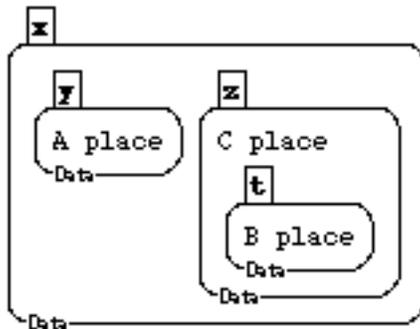
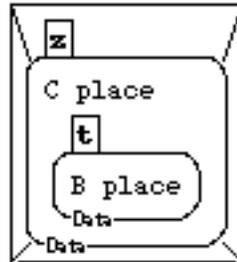
Returns a port to the box that contains <box>, the input to **superior**. **superior** is useful when you want to find out about or manipulate some aspect of the environment that some box (which you have a port to) is located in. Use (1) a data accessor (e.g., in a data search procedure that returns a port), (2) **self** (e.g., to find out which sprite instigated a sprite-mouse- command), or (3) one of the **mouse-rc** commands to get a port whose superior is useful. The input should be a port, or the name of a box (which **superior** converts to a port). The examples below are not very interesting since you know what the superior to the specified box is just by looking at the target of the port, or the (named) superior of a chained name like x.z.t. However, they help illustrate the point.

Examples

superior



superior x.z.t |



2.10 DATA CONSTRUCTION

build

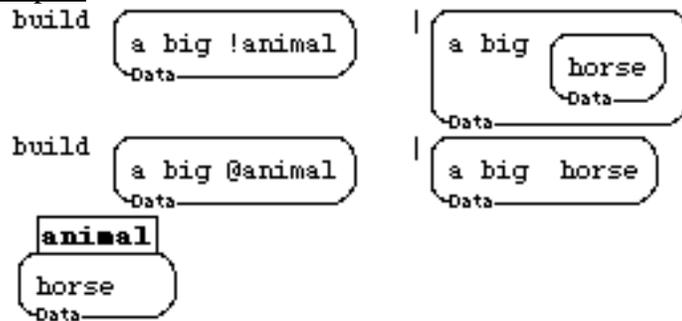
Syntax

build <template box>

Description

This command takes a single data box as input. The input serves as a template. Each part of the template that does not have a special symbol in front of it (@ or !) appears in the output exactly as in the template. Parts of the template preceded by ! are replaced by the result of executing the word or doit box following !, including the box wrapper. If @ is used in place of !, the box wrapper is removed. @ may be used in the nametab of a box to create new named variables or procedures. ! only works as part of the input to **build**. @ can be used independent of **build** to refer to the contents of a box.

Examples



join-bottom

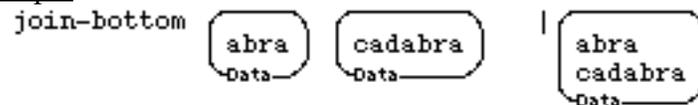
Syntax

join-bottom <box1> <box2>

Description

The contents of <box1> are vertically concatenated with the contents of <box2>. The resulting box appears as if <box1> had been placed on top of <box2> and the common box border erased. Note that **join-bottom** X Y is (almost) the same as **build** <template> where the template is a data box with row 1 containing @X and row two containing @Y. However, they may treat spaces differently.

Example



join-right

Syntax

join-right <box1> <box2>

Description

The contents of <box2> are concatenated to the right of <box1>. The resulting box appears as if <box2> had been placed to the right of <box1> and the common box border erased.

Note that join-right X Y is (almost) the same as **build** <template>, where template is a box containing @x and @y in the same row.

Example

```
join-right
```



boxify

Syntax

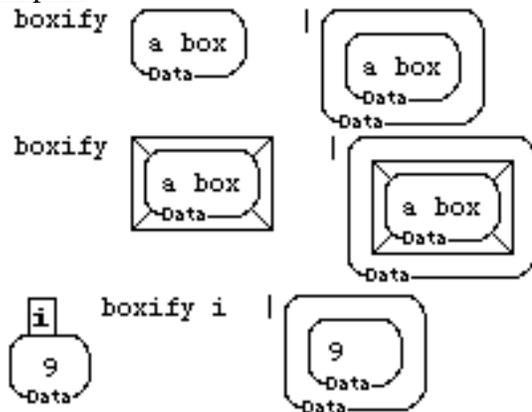
boxify <box>

Description

Returns a copy of its (evaluated) input inside a box. That is, it adds one data box wrapper to its input.

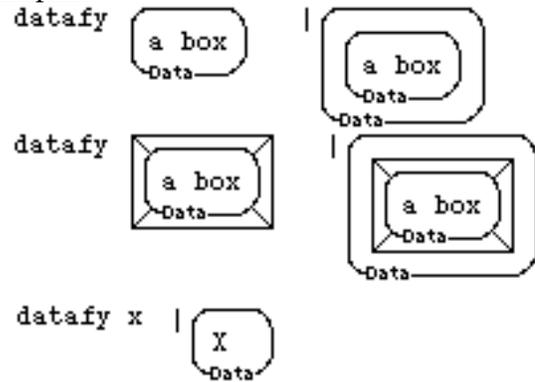
Examples

```
boxify
```



datafySyntax**datafy** <any-item>Description

Returns a copy of its (unevaluated) input inside a box. That is, it adds one data box wrapper to its input. Useful only for demonstrating data flavored inputs.

Examples

2.11 DATA CONVERSION

change

Syntax

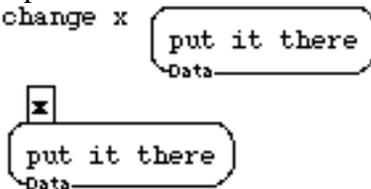
change <box> <new-box>

Description

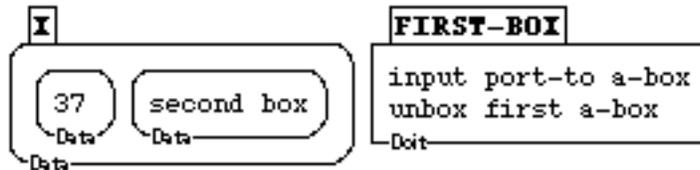
Replaces the contents of <box> with a copy of the contents of <new-box>. If <box> is a literal data box, that box gets changed. If <box> is the name of a data box, that named box gets changed. If <box> is a port or evaluates to a port, the target of the port is changed. See **retarget** for "rewiring" ports. Also, note that the closet of the box changed does *not* get changed. This is deliberate so that you can have some behavior or identification stay with a box when you change it. Note that in the last example below, if you can arrange for the first input to change to be a port to a box, that box is changed. This is simply a consequence of the "stickiness" of ports. See ports and port-flavored inputs in Boxer Structures.

Examples

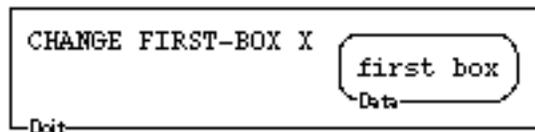
change x



change first-box X 37



Reset:



retarget

Syntax

retarget <port> <new-target>

Description

Changes the target of a port to some other box. Used for "rewiring" your world. This is "the port equivalent" of **change**. **change** is more flexible than **retarget**. You can change any box-part of a complex box by using port-to and stickiness. But you cannot do this with **retarget**.

Examples

```
retarget x y
retarget x t
```



2.12 CONVERSION BY ITEM

change-item

Syntax

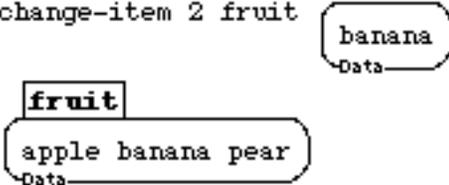
change-item <n> <box> <new-item>

Description

Changes the item number <n> of <box> to a copy of the contents of <new-item>. To remember input sequence read as: "Change item 1 (of) box (to) new item." Note that all the parts of <new-item> get stuffed into the position of the nth item. Also, note that item at position <n> is completely replaced, even if it is a box. Like **change**, if you manage to pass a port **change-item** as its second input, that target of that port will be changed. Like **change**, the last input will be copied, but any ports contained in that box will remain ports, according to the standard Boxer rules for copying boxes and ports. See **change** and **first**. See ports and "stickiness" of ports in Boxer Structures.

Example

```
change-item 2 fruit
```



The diagram illustrates the result of the `change-item 2 fruit` command. It shows a box labeled **fruit** containing the text "apple banana pear". Above this box, there is a smaller box labeled **banana** containing the text "banana". This represents the replacement of the second item in the 'fruit' box with the contents of the 'banana' box.

delete-item

Syntax

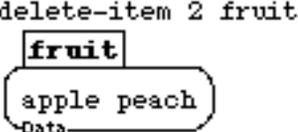
delete-item <n> <box>

Description

Changes <box> by removing item number <n> from it. This command follows the same rules of "portness" as **change**.

Example

```
delete-item 2 fruit
```



The diagram illustrates the result of the `delete-item 2 fruit` command. It shows a box labeled **fruit** containing the text "apple peach". This represents the removal of the second item from the 'fruit' box.

delete-last-itemSyntax**delete-last-item** <box>Description

Changes <box> by removing the last item from it. This command follows the same rules of "portness" as **change**.

Example

delete-last-item fruit

**delete-items**Syntax**delete-items** <start> <end> <box>Description

Removes items from <box> starting at <start> and ending at item number <end>. This command follows the same rules of "portness" as **change**.

Example

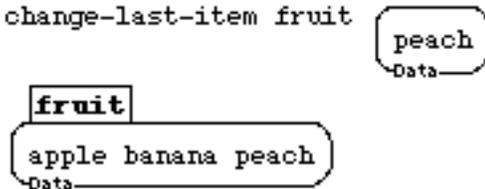
delete-items 1 2 fruit

**change-last-item**Syntax**change-last-item** <box> <new-item>Description

Changes the last item of <box> to the contents of <new-item>. This command follows the same rules of "portness" as **change**.

Example

change-last-item fruit



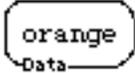
insert-itemSyntax

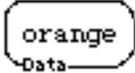
insert-item <new-item> <box> <n>

Description

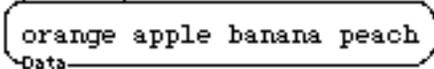
Puts whatever is in box <new-item> into <box> at the item position <n>. If more than one item is in <new-item>, all of them get inserted, starting at position <n>. The order of inputs is designed to reflect "English" usage, e.g., "**insert-item** <which item> into: <which box> at: <place>." **insert-item** is similar to change-item in its operation. This command follows the same rules of "portness" as **change**.

Example

```
insert-item  fruit 1
```



```
fruit
```



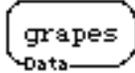
append-itemSyntax

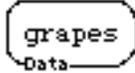
append-item <new-item> <box>

Description

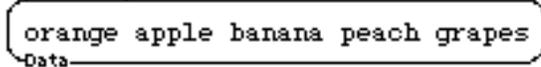
Adds the stuff in <new-item> to the end of <box>. The order of inputs is designed to reflect English usage, e.g.: **append-item** <which-item> to: <box>. As a generic append, **append-row** will probably be "neater" in accumulating stuff. This command follows the same rules of "portness" as **change**.

Example

```
append-item  fruit
```



```
fruit
```



2.13 CONVERSION BY ROW

change-row

Syntax

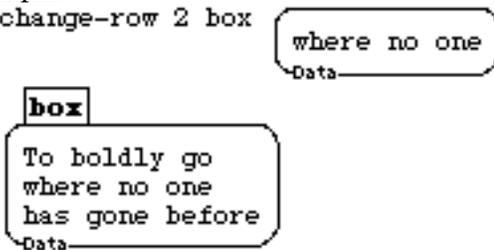
change-row <n> <box> <new-row>

Description

Changes the row number <n> of <box> to a copy of the contents of <new-row>. Multiple rows from <new-row> can replace the specified row. This command follows the same rules of "portness" as **change**.

Example

```
change-row 2 box
```



change-last-row

Syntax

change-last-row <box> <new-row>

Description

Changes the last row of <box> to the contents of <new-row>. This command follows the same rules of "portness" as **change**.

Example

```
change-last-row box
```



delete-rowSyntax**delete-row** <number> <box>Description

Deletes the specified row from its input. This command is needed because changing a row to be an empty row is not the same thing as deleting it. This command follows the same rules of "portness" as **change**.

Example

delete-row 1 box

```

box
where no one
has gone before
Data_____

```

delete-last-rowSyntax**delete-last-row** <box>Description

Deletes the last row of its input. This command follows the same rules of "portness" as **change**.

Example

delete-last-row box

```

box
To boldly go
where no one
Data_____

```

delete-rowsSyntax**delete-rows** <start> <end> <box>Description

Removes rows from <box> starting at <start> and ending at row number <end>. This command follows the same rules of "portness" as **change**.

Example

delete-rows 2 3 box

```

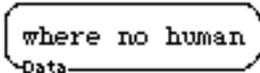
box
To boldly go
Data_____

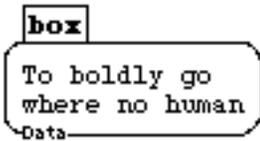
```

insert-rowSyntax**insert-row** <new-row> <box> <n>Description

Puts whatever is in box <new-row> into <box> at the row number <n>. If more than one row is in <new-row>, all of them get inserted, starting at position <n>, and the rest of the box rows gets pushed down. The input sequence is patterned after English usage: **insert-row** <new-row> into: <box> at: <row number>. This command follows the same rules of "portness" as **change**.

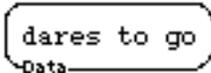
Example

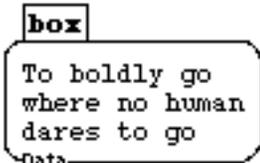
```
insert-row  box 2
```


append-rowSyntax**append-row** <new-row> <box>Description

Adds the stuff in <new-row> to the end of <box>, starting on a new row. The order of inputs is designed to reflect English usage: **append-row** <new row> to: <box>. As a generic append, append-row will probably be "neater" in accumulating stuff than append-item. This command follows the same rules of "portness" as **change**.

Example

```
append-row  box
```



2.14 CONVERSION BY COLUMN

change-column

Syntax

change-column <n> <box> <new-column>

Description

Changes the column number <n> of <box> to a copy of the contents of <new-column>. Multiple columns from <new-column> can replace the specified column. This command follows the same rules of "portness" as **change**.

Example

change-column 3 a

```
travel
one
recently
Data_____
```

```
a
to boldly travel
where no one
has gone recently
Data_____
```

change-last-column

Syntax

change-last-column <box> <new-column>

Description

Changes the last column of <box> to the contents of <new-column>. This command follows the same rules of "portness" as **change**.

Example

change-last-column a

```
first
second
third
Data_____
```

```
a
1 first
2 second
3 third
Data_____
```

delete-column

Syntax

delete-column <number> <box>

Description

Deletes the specified column from its input. It deletes the last item from each row. This command follows the same rules of "portness" as **change**.

Example

```
delete-column 2 box
```

```

box
┌───────────┐
│ To go      │
│ where one  │
│ has before │
└───────────┘
Data_____

```

delete-last-column

Syntax

delete-last-column <box>

Description

Deletes the last column of its input. This command follows the same rules of "portness" as **change**.

Example

```
delete-last-column box
```

```

box
┌───────────┐
│ To boldly  │
│ where no   │
│ has gone   │
└───────────┘
Data_____

```

delete-columns

Syntax

delete-columns <start> <end> <box>

Description

Removes columns from <box> starting at <start> and ending at column number <end>. This command follows the same rules of "portness" as **change**.

Example

```
delete-columns 2 3 box
```

```

box
┌───────────┐
│ To         │
│ where     │
│ has       │
└───────────┘
Data_____

```

insert-columnSyntax

insert-column <new-column> <box> <n>

Description

Puts <new-column> into <box> at the column number <n>. Multiple columns can be inserted if <new-column> has more than one. This command follows the same rules of "portness" as **change**.

Example

```
insert-column
```

```
1.
2.
3.
4.
Data
```

```
x 1
```

```
x
1. able one
2. baker two
3. chow three
4. dog four
Data
```

append-columnSyntax

append-column <new-column> <box>

Description

Adds the stuff in <new-column> to the right end of <box>. This command follows the same rules of "portness" as **change**.

Example

```
append-column
```

```
upon
in the west
Data
```

```
story
```

```
story
once upon
a time in the west
Data
```

2.15 CONVERSION BY ROW & COLUMN

change-rc

Syntax

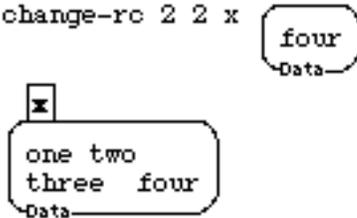
change-rc <row> <column> <box> <new-item>

Description

Changes the item at <row> and <column> to the contents of <new-item>. Multiple items can be inserted. This command follows the same rules of "portness" as **change**.

Example

```
change-rc 2 2 x
```



insert-rc

Syntax

insert-rc <more> <box> <row> <column>

Description

Inserts the contents of <more> into <box> at position indicated by <row> and <column> (and the rest of the box moves to the right and down). This command follows the same rules of "portness" as **change**.

Example

```
insert-rc hello there x 2 2
```



delete-rc

Syntax

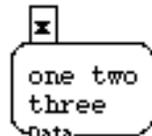
delete-rc <row> <column> <box>

Description

Deletes the element of <box> specified in by <row> and <column>. This command follows the same rules of "portness" as **change**.

Example

```
delete-rc 2 2 x
```



2.16 CONVERSION BY BOX

boxify

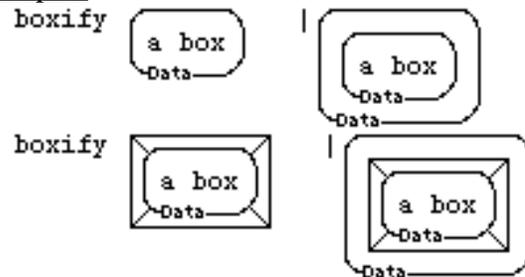
Syntax

boxify <box>

Description

Returns a copy of its (evaluated) input inside a box. That is, it adds one data box wrapper to its input.

Examples



datafy

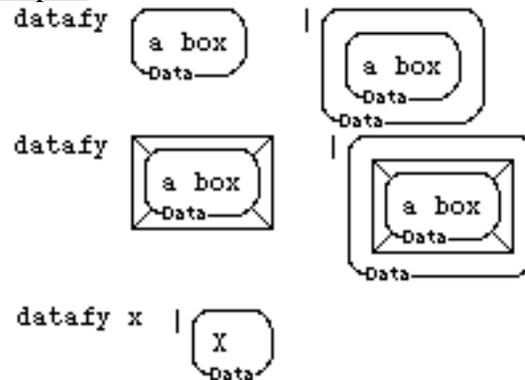
Syntax

datafy <any-item>

Description

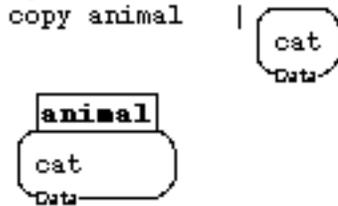
Returns a copy of its (unevaluated) input inside a box. It adds one data box wrapper to its input. It differs from **boxify** only in that it has a data flavored input, so it works on words as well as boxes.

Examples

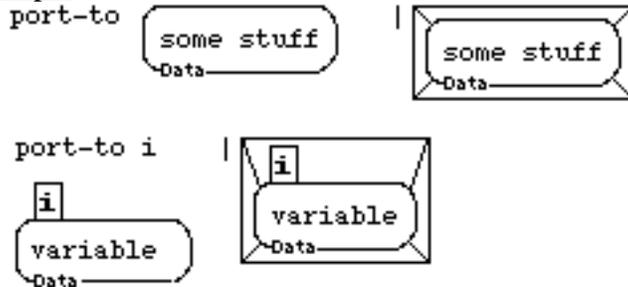


copySyntax**copy** <box>Description

This command makes a copy of its input. It is useful when you want to duplicate the data in a port. The usual copy and execute rules apply. This means ports originally with targets internal to box-to-be-copied will be completely new in the copy. Their targets are new boxes. Ports to boxes outside the copied box will become ports to those old external boxes.

Example**port-to**Syntax**port-to** <box>Description

Gets a port to the input. That is, if the input is a box, it returns a port to the box. If the input is a named data box, it returns a port to the box. See **retarget** in data-mutation. See also port-flavored inputs in Boxer Structures.

Example

unbox

Syntax

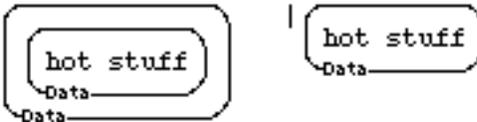
unbox <box>

Description

This command expects exactly one box inside a box as input, and it returns the inside box. It removes the outside layer of box. Use **unboxable?** to check whether the input is suitable for **unboxing**. You can unbox a box containing a variable. However, this is only useful at top level, directly on the screen. A named data box unboxed as part of a procedure will not establish a variable. See @ (special characters) for a different kind of unboxing.

Example

unbox



The diagram shows the effect of the `unbox` command. On the left, a box labeled "Data" contains another box labeled "Data" which contains the text "hot stuff". An arrow points from this nested box to the right, where a single box labeled "Data" contains the text "hot stuff".

2.17 CONVERSION BY CHARACTER

letters

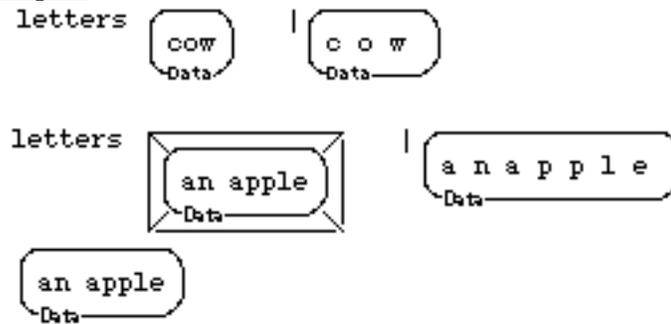
Syntax

letters <box>

Description

This command separates all the letters in its input with spaces. It does not process subboxes.

Examples



word

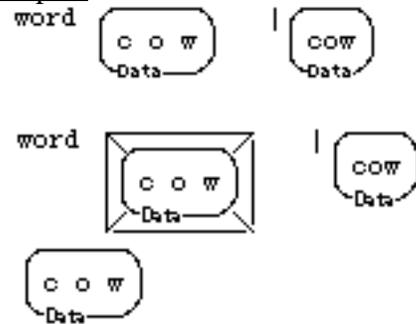
Syntax

word <box>

Description

This command takes its input and removes all spaces. It does not process subboxes or combine rows.

Examples



2.18 ACCESSING NAME INFORMATION

name-help

Syntax

name-help <pattern>

Description

Returns a box containing very brief information on all the things Boxer knows about that have the text in the <pattern> as part of their name. This includes all primitives and user defined boxes that are accessible where name-help is executed. Do not put a box around the text <pattern>. **name-help** is used mostly to find the name of a command when you have some idea of what the command name might be. Many times, you can find interesting commands by guessing what might be part of their name, like "sprite" or "graphics" or "mouse".

Example

```
name-help word |
WORD is a primitive with arguments of (BOX)
WORD? is a primitive with arguments of (BOX)
Data_____
```

name?

Syntax

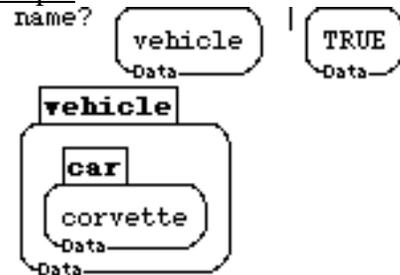
name? <name>

Description

This command tells you if any accessible box or primitive has the name that is inside its input box. This command does scoping as Boxer does usually, looking inside the current box then looking inside the containing box, and so on. **name?** only looks at the first word in its input.

Example

```
name?
vehicle | TRUE
Data_____
```



local-name?

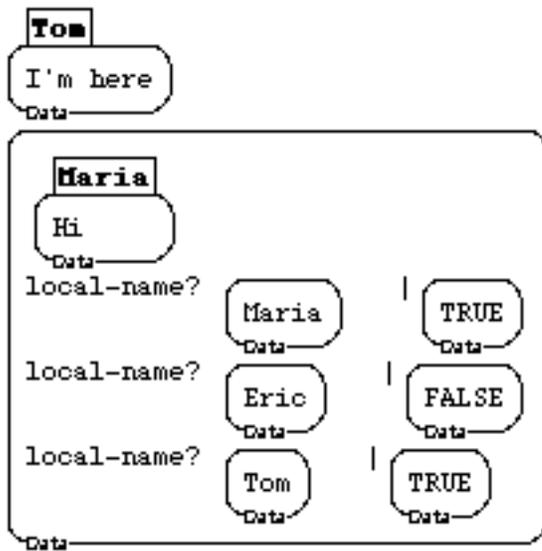
Syntax

local-name? <name>

Description

local-name? tells you if any box has the name that is inside its input box. It differs from **name?** in not returning true for primitives and it differs from **name-in-box?** by returning **true** for names that are defined, not in the particular box, but at some higher level of the box hierarchy.

Example



name-in-box?

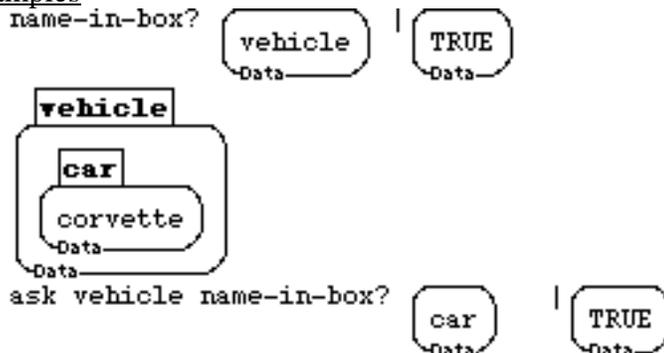
Syntax

name-in-box? <name> <box-to-check>

Description

Tells you if there is a box defined in the local environment (the box where **name-in-box?** is executed) with the name that is inside <name>. In contrast to **name?**, it does not look outside the box where it is executed, hence will never find a primitive.

Examples

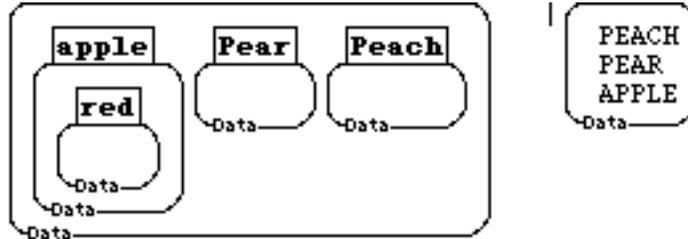


namesSyntax**names**Description

Returns a list of all the names defined where **names** is executed. It does not return the names of Boxer primitives or of boxes named in boxes superior to where it is called.

Example

names

**top-level-name?**Syntax**top-level-name?** <name>Description

Returns **true** or **false** depending on whether <name> is a Boxer primitive. If the primitive is redefined, **top-level-name?** will return **false**.

Example

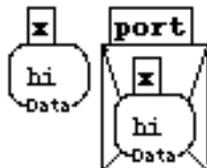
top-level-name?

**target-name**Syntax**target-name** <box-or-port>Description

This command returns the name of the target of the port given as input. **target-name** is a way of finding the name of boxes under program control. Use data accessors to get a port to the box you want, and apply **target-name** to the result. Input must be a port to work properly.

Example

target-name port



CHAPTER 3

Graphics

Graphics in Boxer appear inside "graphics boxes." These may contain line drawings, bitmap graphics (including typed text) and "sprites," which are mobile graphical objects that can have their own shapes. In general, sprites are the creatures that draw the figures inside graphics boxes. Sprites can draw lines; they can stamp their shapes or stamp a bitmap on the graphics background; similarly, they can type text. You normally have to address sprites with **tell** (synonym, **ask**) as in **tell joe forward 100**. However, you can avoid **tell** by using Turtle boxes (see turtle boxes bellow). In addition, if you happen to be inside a sprite, of course, you need not use **tell**. Finally, Graphics boxes are the place you may also create flexible "point and click" interfaces. It is easy to arrange any action to occur on clicking in a graphics box, or clicking on a sprite.

Graphic Boxes

Graphics boxes, unlike most boxes, have two presentations. Ordinarily, you can see the pictures they contain, including any sprites that that might be in them. But if you "flip" a graphics box by clicking the mouse button on its lower left corner, you will see the computational structure of the box. This includes the sprite boxes that correspond to sprite shapes appearing in the graphical presentation, and any data or procedures that are defined specially for that graphics box. Some of these procedures may define actions carried out when you click a mouse button in the graphics box (see section on mouse clicks).

Graphics boxes are data in the ordinary sense; they may be named to become variables; they may be passed as input to, and returned as outputs from procedures. Hence, a procedure can construct a complex interactive ("*clickable*") graphics box, and return that box as a tool for you to use (e.g., a calculator).

It is handy to have graphics boxes be transparent, so that any sprites inside may be directly addressed, as in **tell joe forward 100**. The default is that Boxer creates graphics boxes in transparent form, which accounts for their dashed box boundary.

Sprite Boxes

Sprites are Boxer's version of Logo turtles. You can make as many of these as you like in a graphics box. (You can have sprites outside of a graphics box too, but their shapes will not appear anywhere.) Each sprite may have its own shape. If you put a sprite inside another sprite, it becomes a subsprite. That means the subsprite moves with any commands sent to the supersprite, and any commands you execute in the subsprite will cause it to move relative to the supersprite.

Although sprites are very much like special data boxes that happen to cause their shape to appear on the graphical presentation of graphics boxes, there is one significant difference. When you execute a sprite or its name, you get a port to that sprite rather than a copy of it. This is because you generally use a sprite's name to get access to it, to send messages to it (rather than to a copy of it). If you want a copy, use the **copy** command, as in **copy joe**.

Sprite Properties

Conceptually, each sprite contains (1) a set of its own properties (local variables like **x-position**, **heading**, and so on) and (2) ways to manipulate them, such as **forward**, **setheading**, and so on. This is why you must either **tell** a sprite a command, or be inside the sprite when you execute it. When you create a sprite, you will see inside it its x and y position variables, and its heading. You may edit these directly (changes will take place when you exit the edited variable), or you can use **change**, or any of the other property-changing commands such as **forward** or **setheading**.

There are additional properties of a sprite, other than position and heading. The most useful of these are **pen** (whether the pen is **up** or **down**), **show?** (whether a sprite is showing or hidden), **shape**, **pen-color**, and **pen-width**. These normally do not appear (unlike **x-position**, **y-position** and **heading**). They can be made to appear in the closet of the sprite by simply asking for their values. E.g., **ask joe pen** will cause the **pen** property to appear as a variable in Joe's closet. Alternatively, all of these properties can be made to appear at once in the sprite closet by executing **show-sprite-properties**. That command appears in the closet of every sprite when it is created.

You can delete any property boxes if you don't want to see them. They will still work properly. You can save a lot of memory this way. Regular turtle commands won't cause them to reappear.

Special Note: All sprite commands must be addressed to a sprite (e.g. **tell joe**), or used in the presence of a turtle box (transparent graphics box). Or you may type and execute directly inside a sprite. Graphics box commands are similar.

Turtle Boxes

For compatibility with Logo, a keystroke (or menu) command is provided to create a "turtle box," that is, a transparent graphics box containing exactly one (transparent) sprite. Once a turtle box has been created, any turtle commands, like **forward**, **right**, etc., may be executed in the box containing the turtle box without **tell**. In addition, because the sprite is transparent, all sprite properties are also accessible outside both the sprite and the turtle box. E.g., you can execute **shape**, **pen-color**, and also any commands or data names that you have defined yourself for that sprite, all without using **tell**. In net, after you make a turtle box, you can immediately "do Logo" by executing turtle commands, without using **tell**.

You can only have one turtle box present in a given box. This is because multiple turtle boxes will create confusion about who is being addressed by any turtle command, and there will be conflicts in terms of sprite property names. Also, you can add more sprites inside a turtle box. However you will have to address them with **tell**.

Drawing

As a sprite moves, it will draw over the background according to the state of its pen. E.g., if the pen is down, it will draw; if it is up, it will not. You can adjust the size of the pen and its color. A sprite can stamp certain shapes, like circles, rectangles, and any picture that you may have in Boxer (You might have scanned some in.). It can also stamp its own shape onto the background, or stamp some text (**type**). Graphics boxes may "clip" a drawing (not show the sprite or its drawing once past the edge), or it may "wrap", bringing the sprite back on the graphics box from the opposite side when it crosses a box boundary. You can adjust the size of a graphics box manually or under program control. You may "snap" portions of a graphics box out of them, or change the graphics of a box all at once. You may query a sprite for the color it is over, or ask a graphics box what color is at a given position.

Colors

Colors in Boxer are simply graphics boxes whose background has been set to a particular color. If you "flip" a color box, you will see the red-blue-green percentages that specify that color. If you change those percentages either by direct editing or by program control, the color will be changed by a "modified trigger," just like x-position and other sprite properties. See "update-properties" in "sprite-information" section of graphics. Boxer includes some built in color names: **black, white, gray, red, green, blue, orange, yellow, purple, cyan, and magenta.**

Mouse clicks

Mouse clicks over a graphics box do not always do the same things mouse clicks usually do. Instead, you can define them to do anything you want. Graphics boxes are supposed to be the place where you can change the usual boxer interface to be any point-and-poke kind of interface you want.

When you click on a graphics box, a set of special commands, such as **mouse-click-on-graphics**, are executed in the graphics box. You can think of these as "messages" that are automatically sent to the graphics box. If there is no appropriately named box, you will get an error message at the top of the Boxer screen. If you do have a box named **mouse-click-on-graphics**, for example, it will get executed when you click the mouse button over a graphics box.

Similarly, clicking a mouse button while the mouse is positioned over a sprite sends a message to that sprite. Again, if there is no **mouse-click-on-sprite** box (for example), you will get an error message at the top of the screen. (There are sometimes default actions for mouse-clicks, if you do not define them yourself. For example, pressing on a sprite by default will let you drag it around.)

3.1 MOVES & TURNS

forward (fd)
back (bk)
right (rt)
left (lf)

These are the standard turtle commands that cause a sprite to move relative to its current position and heading. They are all used to address sprites, as in **tell joe forward 100** or can be used "near" a turtle box without tell.

3.2 SETTING POSITION & HEADING

home
setheading (seth)
setxy
setposition (setpos)
follow-mouse

These commands move a sprite to a particular position, or set its heading to some absolute compass heading.

3.3 HIDE & SHOW

hideturtle (hide, ht)
showturtle (show, st)
hide-subsprites
show-subsprites

These commands change the visibility of a sprite. You can find out the visibility state of the sprite with **shown?** See **sprite-information**.

3.4 PEN

pendown (pd)
penup (pu)
penerase (pe)
penreverse (pr)
set-pen-width
set-type-font
set-pen-color

These commands all adjust the state of a sprite's pen and tell a sprite to stamp a shape of some kind onto the background. The property variables **pen**, **pen-width**, **pen-font** and **pen-color** return information about the pen's current state. See List-of-properties, under **sprite-information**, in "graphics."

dot
stamp
stamp-circle
stamp-hollow-circle
stamp-ellipse
stamp-hollow-ellipse
stamp-rectangle
stamp-hollow-rectangle
stamp-wedge
stamp-arc
stamp-self
type
ctype
ltype
rtype

These commands all cause a sprite to stamp a shape of some kind onto the background. They are all affected by the state of the sprite's pen. That is, if the pen is up, a stamp command will not result in anything appearing in the graphics box. None of the stamp commands change orientation when the sprite is tilted. E.g., you can't have slanted typing in a graphics box.

3.5 CLEARING GRAPHICS

clearscreen (cs)
clean

These commands clear out a graphics box.

3.6 GRAPHICS SIZE & MODE

set-graphics-size
set-graphics-mode
graphics-mode

These commands change basic state of the graphics box (and allow you to inspect that state). In general they may be addressed to a graphics box or to a sprite inside it. They may be executed directly inside a graphics box or "near" a turtle box.

3.7 SNAPPING & CHANGING

snap
snip
change-graphics

These commands either grab the graphics contents of a graphics box, a part of it, or change it.

3.8 COLOR

make-color
color=
color-under
color-under=
color-at
color-at=
update-color-box
set-color-at
set-background
clear-background (cb)
freeze
bg-color
bg-color-at=
bg-color-at?
without-recording

These commands allow you to use color with sprites and graphics boxes. You can create colors of any specification in Boxer (e.g., to assign to a sprite's pen for drawing, or to change the color in the background of a graphics box). You can ask what the color of the box is under a sprite (or at a particular x, y point). Similarly, you can ask for the color of the background underneath any sprite drawing. Note that Pen color is irrelevant for pictures and shapes drawn with **penreverse**. **penreverse** simply reverses the color of whatever it draws over. Sprite graphics boxes contain a separate background that may be assigned a color, or into which the current sprite picture may be "frozen." The background will not be cleared on **clearscreen** or **clean**; these clear only the foreground sprite graphics. **clear-background** clears the background. You can ask what color is at a particular place in the background, similar to the way one can ask what color appears in the regular sprite picture. Background. Colors are represented in Boxer with graphics boxes, the graphics side of which shows the color, the "flip" side of which shows red, green and blue percentages. The built in color names are black, **white**, **gray**, **red**, **green**, **blue**, **orange**, **yellow**, **purple**, **cyan**, and **magenta**.

3.9 SPRITE INFORMATION & PROPERTIES

x-position
y-position
heading
shape
shown?
pen
***pen-width(not on online man)**
***type-font(not on online man)**
***pen-color (not on online man)**
sprite-size
home-position

These are commands that either set properties of sprites or get information about sprite state. Each sprite has a collection of properties that are changed by sprite commands like **forward**, **change shape**, and so on.

3.10 UPDATE PROPERTIES

**Update-
show-sprite-properties**

These commands make Boxer's internal representation of sprite properties correspond to what's in the property variables in a sprite. Thus, they make the actual changes to the visible representation of the sprite corresponding to a change in **shape**, **heading**, and so on. See the section on sprites (subsection "more on sprite properties") in the overview of the graphics section. Most people will never need to use them.

3.11 OTHER INFORMATION

**distance
enclosing rectangle
touching?
towards**

These commands get useful information from a sprite.

3.12 SPRITE SIZE, SHAPE & HOME

**set-sprite-size
setshape
turtle-shape
set-home-position**

A sprite's size, shape and home-position (where it goes when you tell it **home** or **clearscreen**) may be change with these commands. See also **Sprite-properties** under **Sprite-information**.

3.13 MOUSE INPUT & CLICKS

**-click-on-graphics
-click-on-sprite**

These provide methods to get input from mouse clicks and mouse positioning over a graphics box. For other input methods, see **input-output** chapter.

3.14 MOUSE POSITION

**mouse-position
mouse-position-on-
mouse-x-position
mouse-x-position-on-
mouse-y-position
mouse-y-position-on-**

These commands provide mouse-positioning information for graphics.

3.1 MOVES AND TURNS

forward

Syntax

forward <steps>

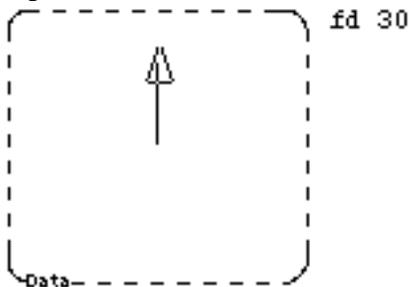
fd abbreviation for **forward**

Description

This command tells a sprite to move forward the specified number of steps.

A negative argument causes the sprite to move backward rather than forward. The direction of movement is determined by the value in the sprite's heading, and the sprite's **x-position** and **y-position** variables are updated appropriately.

Example



back

Syntax

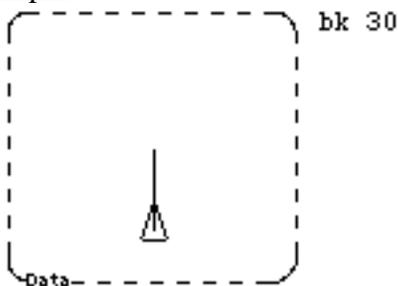
back <steps>

bk abbreviation for **back**

Description

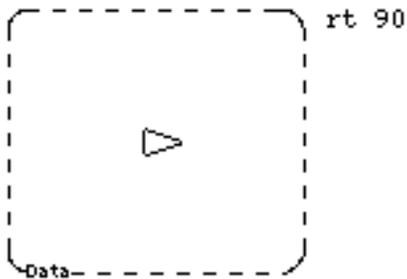
This command tells a sprite to move backward the specified number of steps. A negative argument causes the sprite to move forward rather than backward. The direction of movement is determined by the value in the sprite's heading, and the sprite's **x-position** and **y-position** variables are updated appropriately.

Example

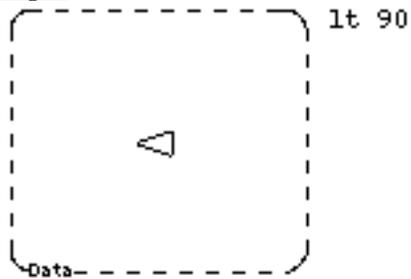


rightSyntax**right** <degrees>**rt** abbreviation for **right**Description

This command tells a sprite to turn right the given number of degrees. If a negative argument is given then the sprite turns left.

Example**left**Syntax**left** <degrees>**lt** abbreviation for **left**Description

This command tells a sprite to turn left the given number of degrees. If a negative argument is given then the sprite turns right.

Example

3.2 SETTING POSITION AND HEADING

home

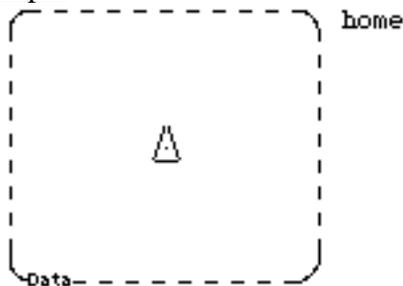
Syntax

home

Description

This command tells a sprite to move to its home position. A sprite's home position is specified inside the sprite in a property variable named **home-position**, inside the sprite's closet. The **home-position** variable will not generally appear inside the sprite until this variable has been modified by a **change** instruction. See graphics overview, under sprites (explanation of properties). The default value of **home-position** is 0,0. See also **clearscreen**.

Example



setheading

Syntax

setheading <heading>

seth abbreviation for **setheading**

Description

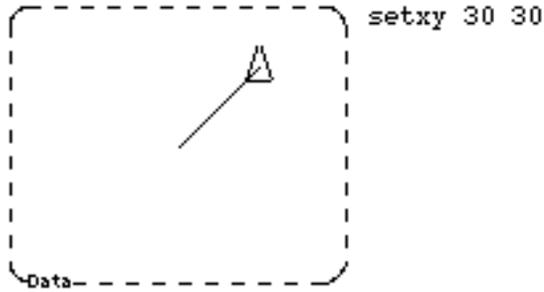
This command tells a sprite to set its heading so that it points in the direction specified by the given angle. The zero of angle is straight up in the graphics box. Positive is to the right (compass heading). Angles greater than 360 degrees and negative angles are interpreted appropriately. Boxer always enters a value between 0 and 360 in the sprite's heading variable.

Example



setxySyntax**setxy** <x> <y>Description

This command tells a sprite to move to the specified x and y coordinates relative to the origin of the graphics box. On graphics boxes that are wrapped, the x and y coordinates will wrap also, so that they will always appear as number within the size constraints of the box. See **graphics-mode**.

Example**setposition**Syntax**setposition** <position>**setpos** abbreviation for **setposition**Description

This command tells a sprite to move to the position specified by its input. The form of the input is a box containing two numbers, the x and y coordinates relative to the origin of the graphics box. **setposition** differs from **setxy** only in the form of the input.

Example

follow-mouse

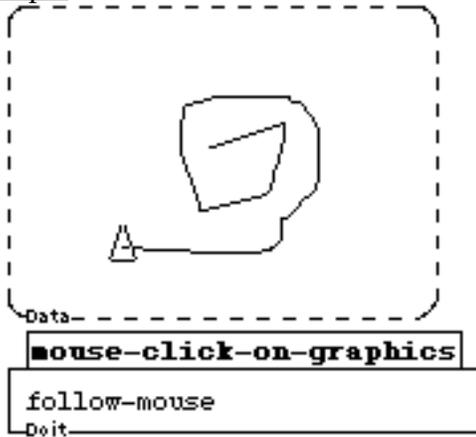
Syntax

follow-mouse

Description

This command causes the sprite to follow the mouse cursor wherever it goes, until you let up on the mouse buttons. It is just a shortcut for writing your own command to fetch the coordinates of the mouse and set the sprite to them.

Example



3.3 HIDE & SHOW

hideturtle

Syntax

hideturtle

ht or **hide** are abbreviations for **hideturtle**

Description

This command causes a sprite to become invisible. See also **shown?** under **Sprite-information**. To make a sprite visible use the **showturtle** command.

Example



showturtle

Syntax

showturtle

st or **show** are abbreviations for **showturtle**

Description

This command causes a sprite to become visible. To make a sprite invisible, use the **hideturtle** command.

Example



hide-sprites

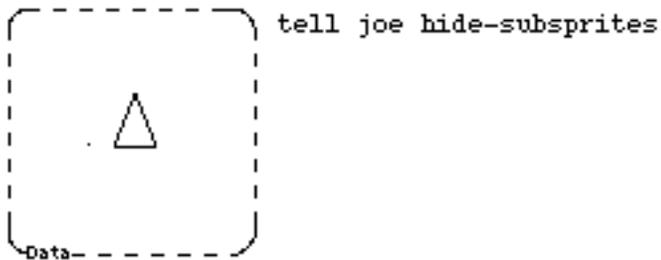
Syntax

hide-sprites

Description

This command tells a sprite to hide all of its subsprites. See **show-sprites**. To make the subsprites visible again use **show-sprites**.

Example



show-sprites

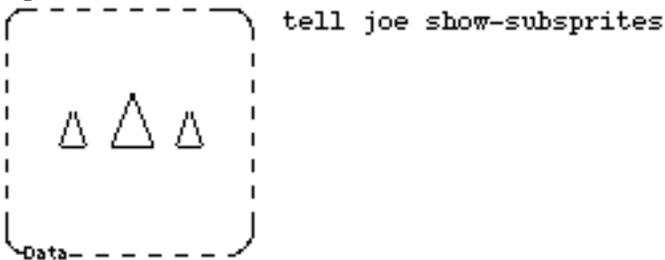
Syntax

show-sprites

Description

This command tells a sprite to make all of its subsprites visible. To make the subsprites invisible use **hide-sprites**.

Example



3.4 PEN

pendown

Syntax

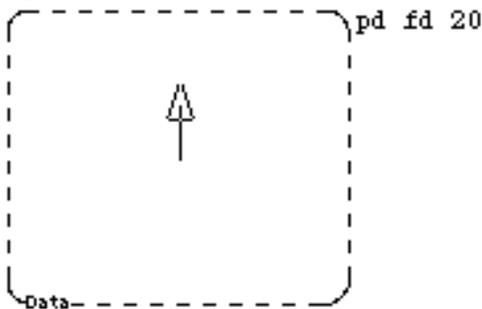
pendown

pd abbreviation for **pendown**

Description

This command tells a sprite to put its pen down. After this command is executed any place that the sprite draws will become black (or whatever color the pen has been set to), regardless of its previous color. Use the **pu (penup)** command to lift the sprite's pen, preventing drawing and stamping. Compare the effects of **pendown (pd)** with the **penreverse (pr)** and **penerase (pe)** commands. See **pen** under **sprite-information**.

Example



penup

Syntax

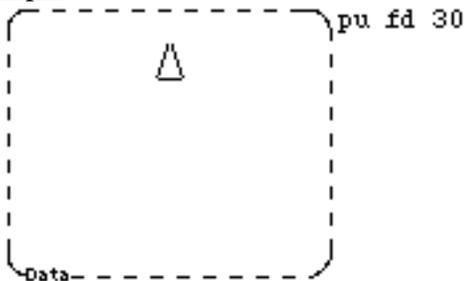
penup

pu abbreviation for **penup**

Description

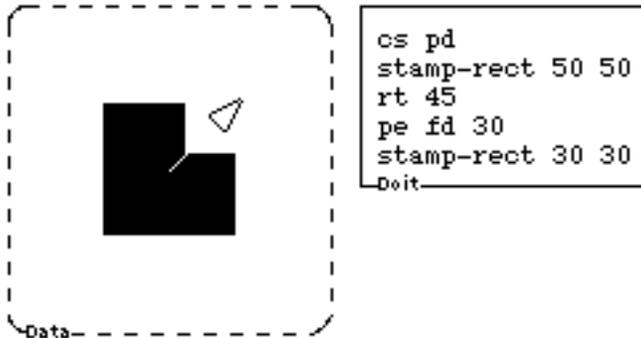
This command tells a sprite to pick up its pen. After this command is executed, no movement or stamping by the sprite will cause any drawing in the graphics box. After this command is executed, no drawing by the sprite will have any effect on the graphics box. Use the **pendown** command to put the sprite's pen back down. See also: **penreverse**, **penerase**.

Example

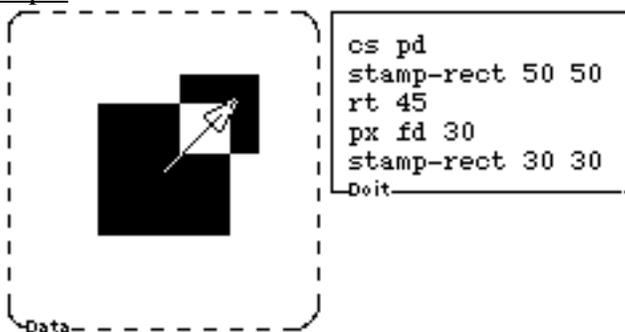


peneraseSyntax**penerase****pe** abbreviation for **penerase**Description

This command tells a sprite to erase instead of drawing. After this command is executed any place that the sprite draws will become **white**, (or whatever is the background color) regardless of its previous color.

Example**penreverse**Syntax**penreverse****pr** abbreviation for **penreverse**Description

After this command is executed any place that the sprite draws will reverse its color. For example, black becomes white and white becomes black. Note: Sprites whose shapes are drawn with **penreverse** will move much faster than with **pendown**. Colors also change (in fairly complex ways) when drawn in **penreverse**.

Example

set-pen-width

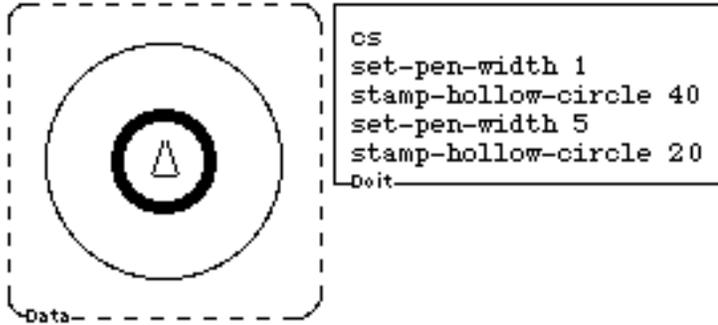
Syntax

set-pen-width <width>

Description

This command sets the width of lines drawn by the particular sprite. The size is in screen pixels. This affects any future drawing in which the result is a line, including, for example, **stamp-hollow-circle**. The size of dots produced by the **dot** primitive are also affected. See **pen-width** under **sprite-information**.

Example



set-type-font

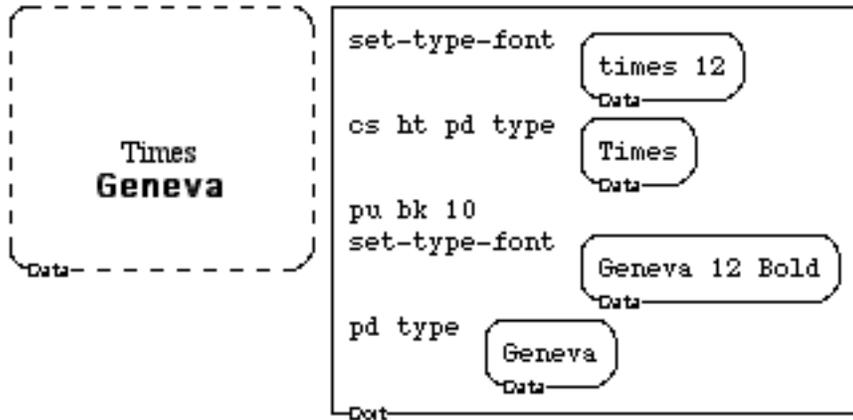
Syntax

set-type-font

Description

This command sets the font that will be used for any text typed by the sprite. Fonts may be specified by <size> <optional style> in a data box, or with a number. Setting a font to a number higher than the number of available fonts is not illegal but will result in repeated use of lower numbered fonts. See **type-font** under **sprite-information**.

Example



set-pen-color

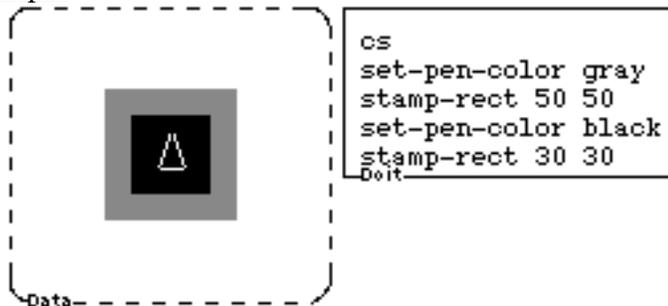
Syntax

set-pen-color <a-color>

Description

This command sets the color for all drawing, stamping or typing done by a sprite. This affects any future drawing, including, for example, **stamp-rect**, **stamp-hollow-circle** and **dot**.

Example

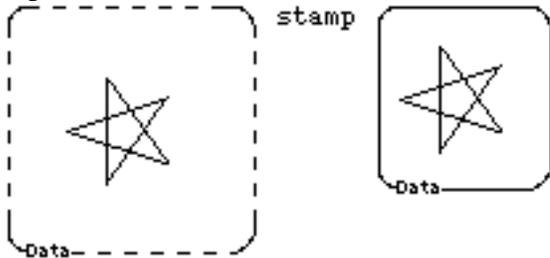


. (dot)Syntax**dot**Description

This command tells a sprite to make a dot centered on the sprite's current location. The size of the dot is affected by the sprite's current **pen-width**. Orientation is not affected by sprite heading.

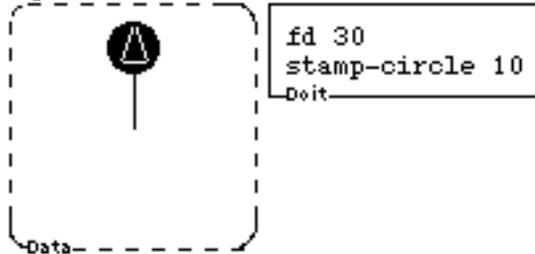
Example**stamp**Syntax**stamp** <graphics>Description

Stamps the contents of a graphics box at the position of a sprite. Must be addressed to a sprite, like **forward**. Orientation is not affected by sprite heading.

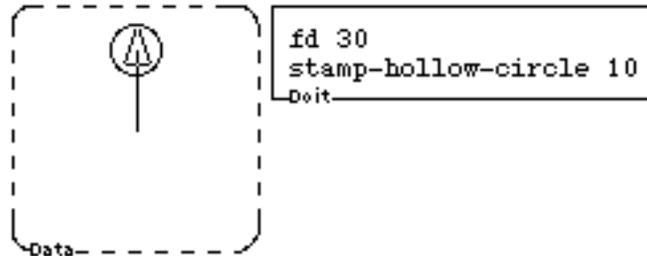
Example

stamp-circleSyntax**stamp-circle** <radius>Description

This command tells a sprite to draw a filled circle of the specified radius centered on the sprite's current location.

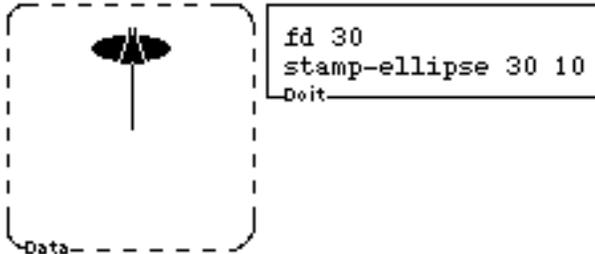
Example**stamp-hollow-circle**Syntax**stamp-hollow-circle** <radius>Description

This command tells a sprite to draw a hollow circle of the specified radius centered on its current location. The thickness and color of the line is determined by the sprite's current **pen-width** and **pen-color**.

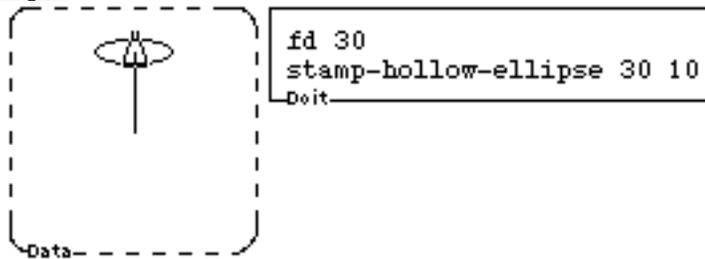
Example

stamp-ellipseSyntax**stamp-ellipse** <width> <height>Description

This command tells the sprite to draw an ellipse with the given width and height centered on the sprite's current location. Orientation is not affected by sprite heading.

Example**stamp-hollow-ellipse**Syntax**stamp-hollow-ellipse** <width> <height>Description

This command tells the sprite to draw a hollow ellipse with the given width and height centered on the sprite's current location. The thickness and color of the line is determined by the sprite's current **pen-width** and **pen-color**. Orientation is not affected by sprite heading.

Example

stamp-rectangle

Syntax

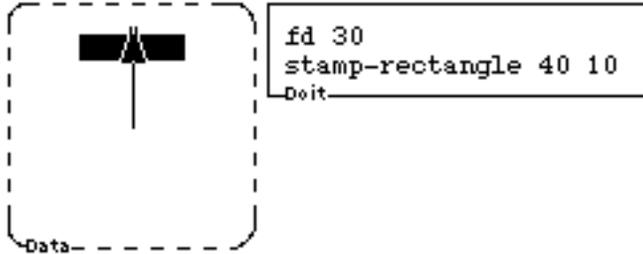
stamp-rectangle <width> <height>

stamp-rect abbreviation for **stamp-rectangle**

Description

This command tells the sprite to draw a rectangle with the given width and height centered on the sprite's current location. Orientation is not affected by sprite heading.

Example



stamp-hollow-rectangle

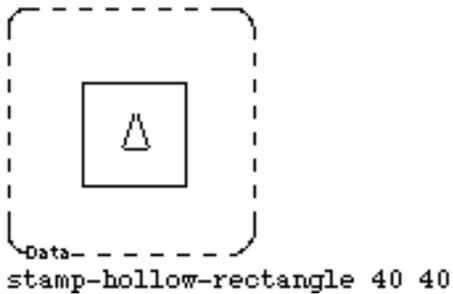
Syntax

stamp-hollow-rectangle <width> <height>

Description

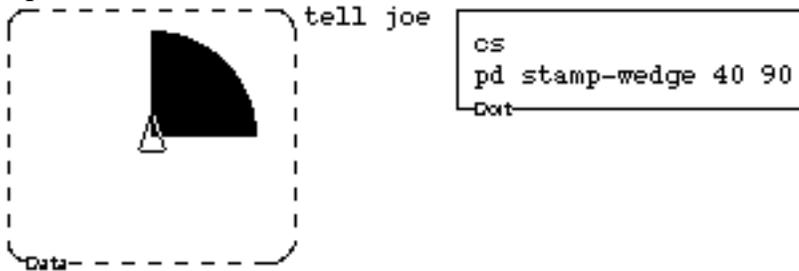
This command tells the sprite to draw a hollow rectangle with the given width and height centered on the sprite's current location. Orientation is not affected by sprite heading.

Example

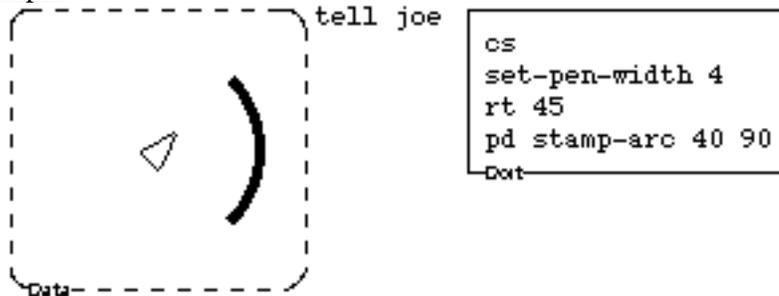


stamp-wedgeSyntax**stamp-wedge** <radius> <angle>Description

This command tells the sprite to stamp a pie wedge of radius <radius> and angular size <angle>, starting at the sprite's position and heading. Orientation IS affected by sprite heading.

Example**stamp-arc**Syntax**stamp-arc** <width> <height>Description

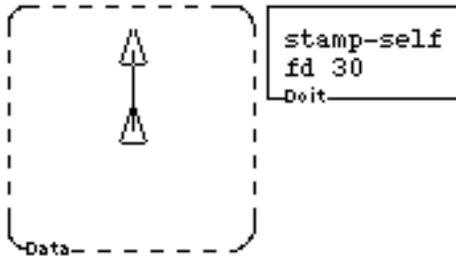
This command tells the sprite to stamp an arc of radius <radius> and angular size <angle>, centered at the sprite's position and heading. Orientation IS affected by sprite heading. Width is affected by pen-width.

Example

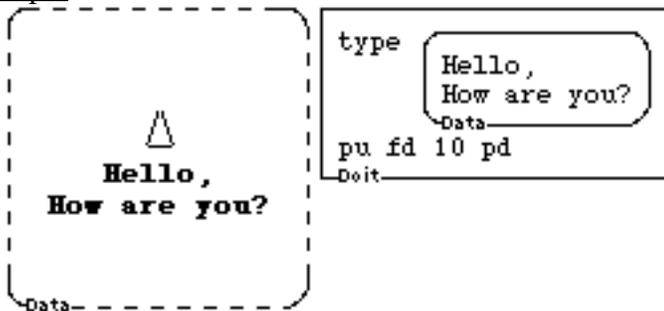
stamp-selfSyntax**stamp-self**Description

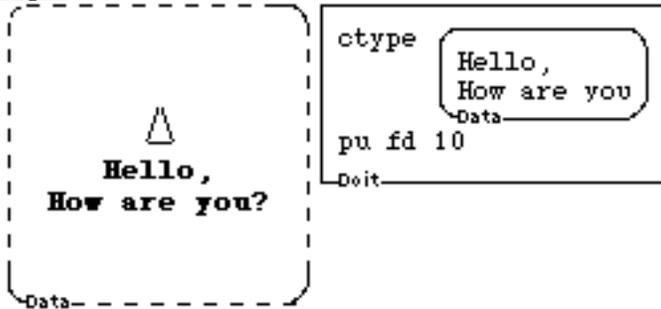
This command tells a sprite to draw its shape in the graphics box at its current location.

Unless the shape of the sprite is a bitmap, the stamp will be affected by the orientation of the sprite.

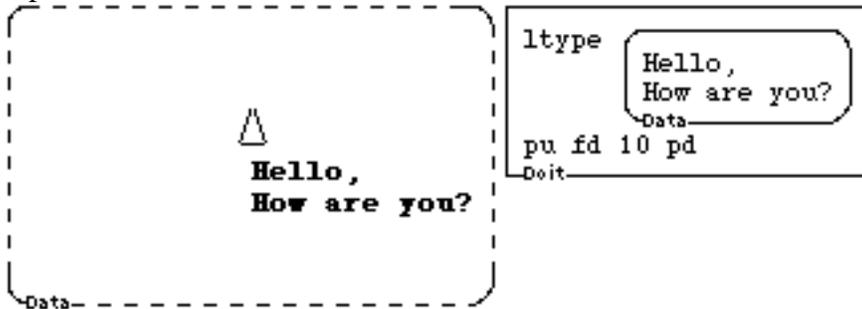
Example**type**Syntax**type <box>**Description

This command tells a sprite to draw the text specified in the argument centered at the sprite's current location. Orientation is not affected by sprite heading. Typing is horizontal only. Multiple lines are acceptable. However sub-boxes will not be printed except as data box. The font used is controlled by the current value of the sprite's **type-font** property. A more complete description of the use of fonts can be found with the **set-type-font** command.

Example

ctypeSyntax**ctype** <box>Description"Center type." Same as **type**.Example**ltype**Syntax**ltype** <box>Description

"Left type." Typing is left justified--aligned on the left, starting from the center of the sprite.

Example

rtypeSyntax**trype** <box>Description

"Right type." Typing is right justified--aligned on the right, starting from the center of the sprite.

Example

3.5 CLEARING GRAPHICS

clearscreen

cleargraphics

Syntax

clearscreen

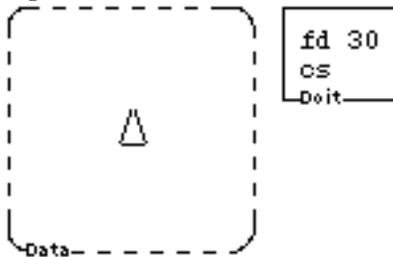
cleargraphics

cs or **cg** abbreviations for **clearscreen** or **cleargraphics**

Description

This command clears all sprite drawing in a graphics box and (if it is addressed to a sprite) moves the sprite to its home position. You may **tell** either a sprite or graphics box to **clearscreen**. Or you may execute it inside a sprite or graphics box, or in the presence of a turtle box.

Example



clean

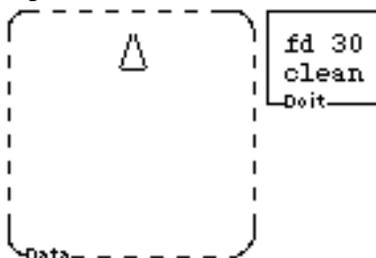
Syntax

clean

Description

This command tells a sprite to clear all drawing in the graphics box without moving to its home position. Like **clearscreen**, it may also be executed in a graphics box directly or by **telling** the graphics box.

Example



3.6 GRAPHICS SIZE & MODE

set-graphics-size

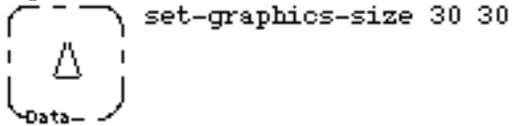
Syntax

set-graphics-size <width> <length>

Description

Changes graphics box size to <width> pixels wide and <length> pixels high. It should be executed inside a graphics box, in the vicinity of a transparent graphics box, or using **tell** to address either a graphics box, or a box (sprite) inside the graphics box.

Example



```
set-graphics-size 30 30
```

graphics-size

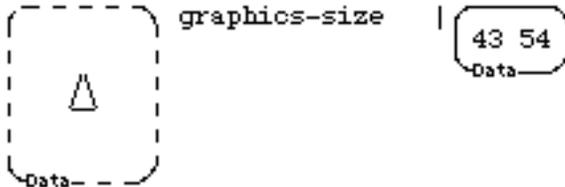
Syntax

graphics-size

Description

Returns the width and height of a graphics box in pixels. It should be executed inside a graphics box, in the vicinity of a transparent graphics box, or using **tell** to address either a graphics box, or a box (sprite) inside the graphics box.

Example



```
graphics-size | 43 54
```

set-graphics-mode

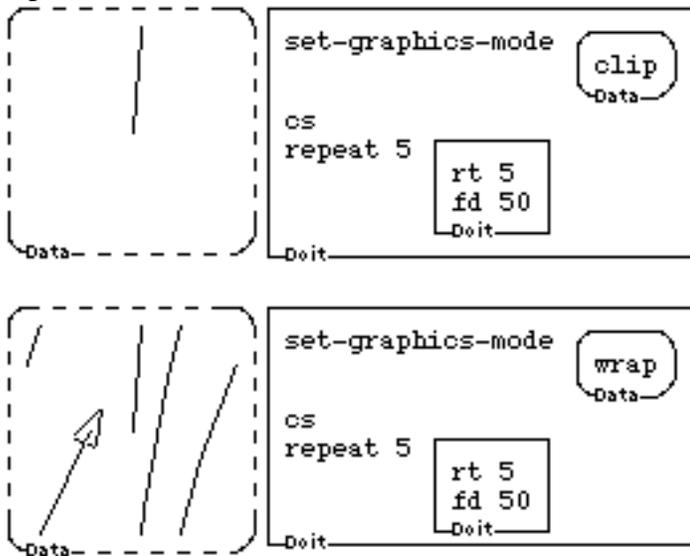
Syntax

set-graphics-mode <clip or wrap>

Description

This command sets a graphics box to clip or wrap modes. Executing this command puts the graphics box in "clip" or "wrap" mode. If the graphics box is in "clip" mode then, when the sprite goes off an edge of the graphics box, the sprite will become invisible and any drawing that the sprite does will not be seen. If you expand the graphics box by grabbing its lower-right corner, you may be able to bring the drawing on screen. If the graphics box is in "wrap" mode then, when a sprite goes off the edge, it will reappear at the opposite side, directly across from where it goes "out of bounds." Use **tell** to the graphics box, or a to a sprite in it, or just in the presence of a turtle box.

Examples



graphics-mode

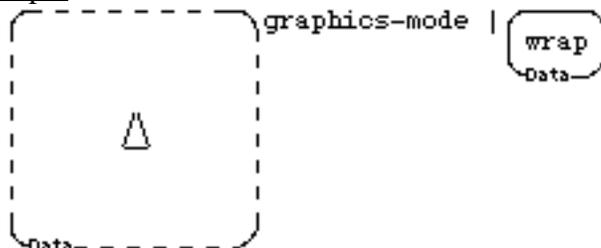
Syntax

graphics-mode

Description

This command returns a data box with either "clip" or "wrap" depending on the mode of the graphics box to which it is addressed. It should be executed inside a graphics box, or using **tell** to address either a graphics box, or a box (sprite) inside the graphics box.

Example



3.7 SNAPPING & CHANGING

snap

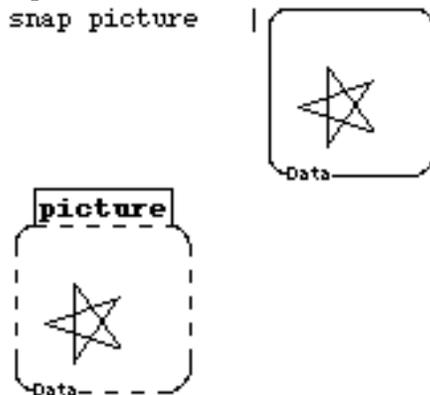
Syntax

snap <graphics box>

Description

This command takes a graphics box as input and returns a graphics box containing the picture that is currently in that graphics box, minus any sprite shapes that might also appear there.

Example



snip

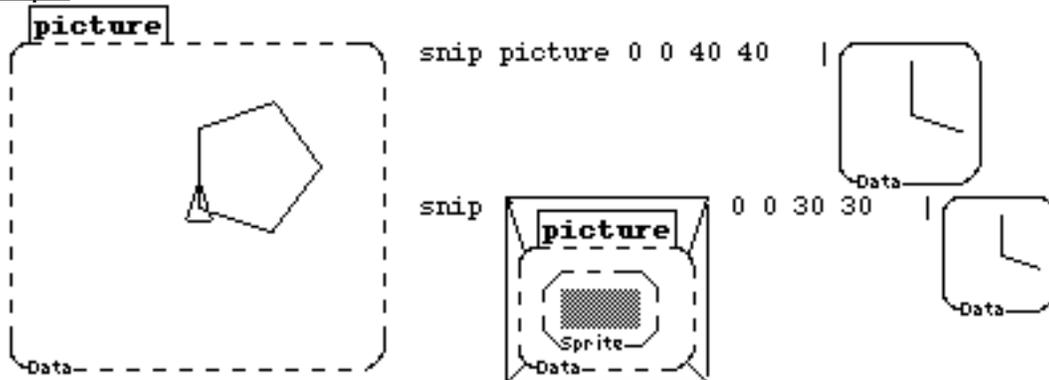
Syntax

snip <graphics box> <x> <y> <width> <height>

Description

This command takes <graphics box>, <x> and <y> position coordinates, and <width> and <height> as inputs, and returns a rectangular piece of the graphics.

Example



change-graphics

Syntax

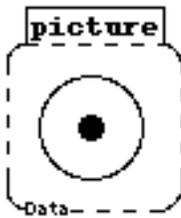
change-graphics <graphics-box> <new-graphics>

Description

This command will change the graphics that is shown in a graphics without changing the sprites that might be there. During the execution of a program, the graphics box will not change size without a **redisplay** command.

Example

```
change-graphics picture
```



3.8 COLOR

make-color

Syntax

make-color <red> <blue> <green>

Description

This command creates a color specified by the percentage of red, blue and green which are given as inputs.

Example

`make-color 50 50 50` | 

color=

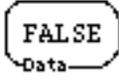
Syntax

color= <color 1> <color 2>

Description

Returns **true** if its inputs are the same screen color, and **false** otherwise. Similar but not identical to =. It differs from = in that = checks only the **Red-Green-Blue** percentages in the color boxes, and **color=** checks the actual displayed color. These may differ in that hardware limitations may force different R-G-B percentages to be displayed identically.

Example

`color=`   | 

color-under

Syntax

color-under

Description

color-under returns the color in a graphics box at the position of a sprite. You must use **tell (ask)** <a sprite> or be in the presence of a turtle box. If the sprite is clipped, i.e., out of the visible portion of the box, **color-under** will not be able to determine the color at that position.

Example

```

cs pd
set-pen-color black stamp-circle 60
set-pen-color white stamp-circle 40
set-pen-color black stamp-circle 20
pu
Dot

```

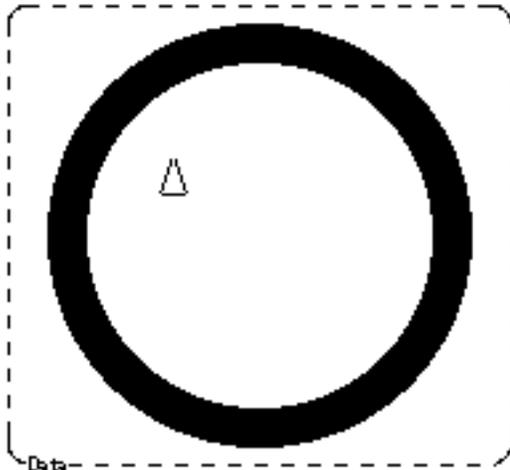
```

Bull's-Eye
setxy 30 30
color-under

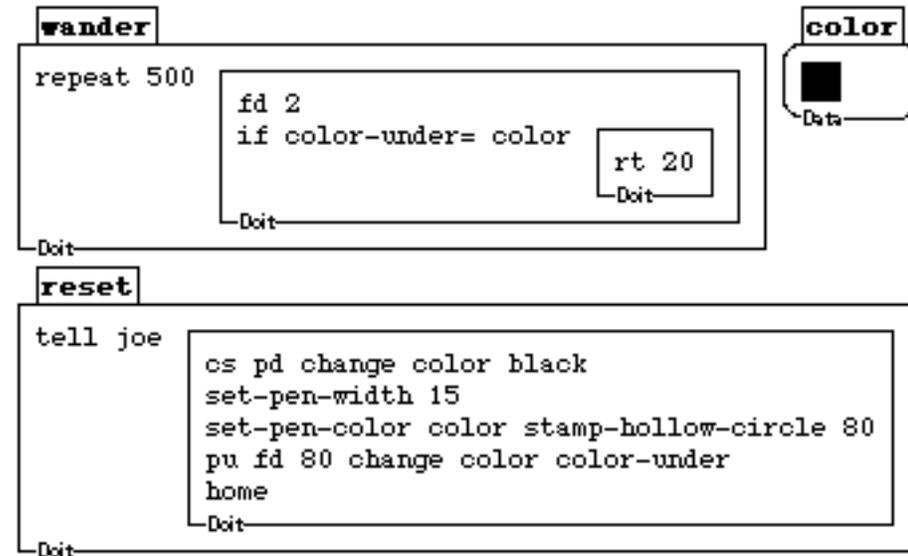
```

color-under=Syntax**color-under=** <color>Description

color-under = checks that the color at a sprite's position is the same as the color given as an input to **color-under=**. **color-under = <color>** is essentially the same as **color = color-under <color>**, only much faster to execute. As with **color-under**, you must use **tell (ask) <a sprite>** or a turtle box.

ExampleDoit
reset

tell joe wander

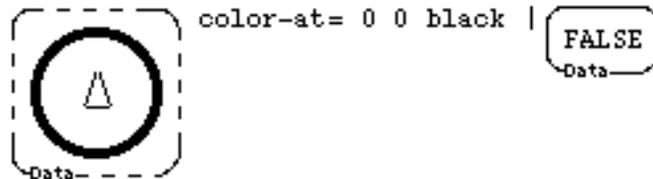


color-atSyntax**color-at** <x> <y>Description

Returns the color in graphics box at the specified coordinates. You must use **tell**, **ask**, or a turtle box. **color-at** returns the actual color you see at a particular point, even if that is due to a sprite shape. If the coordinates specified are out of the visible portion of the box, **color-at** will not be able to determine the color at that position.

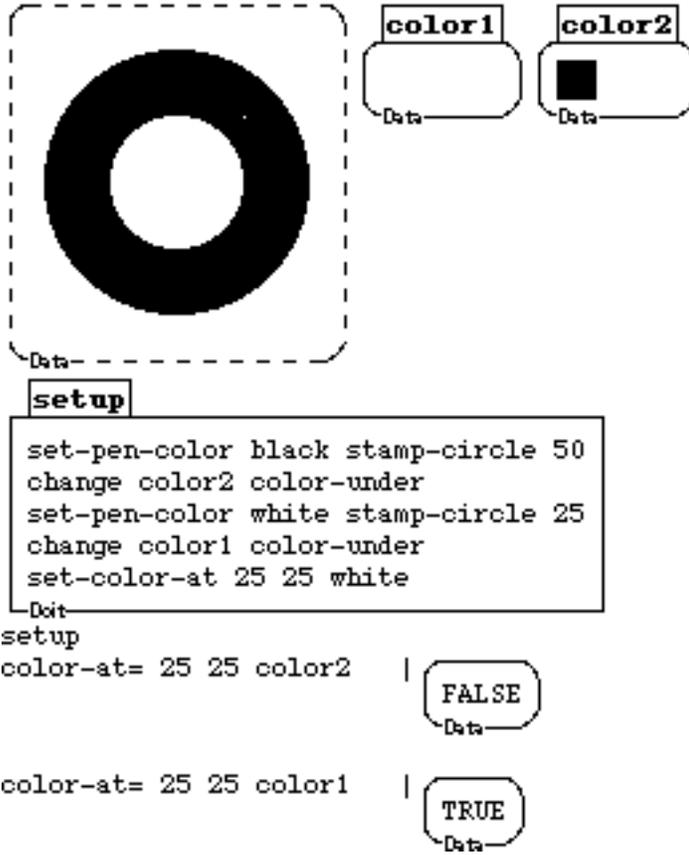
Example**color-at=**Syntax**color-at=** <x> <y> <color>Description

Checks that the color at the specified position is the same as the color given as an input. In other words, it asks the question “Is the color in this <x> <y> position the same as the <color> given as an input.” Use **tell**, **ask**, or a turtle box. **color-at = <x> <y> <color>** is same but faster than **color = <color-at <x> <y>> <color>**.

Example

set-color-atSyntax**set-color-at** <x> <y> <color>Description

This command sets the position <x> <y> to the specified <color>. Use **tell**, **ask**, or a turtle box.

Example**update-color-box**Syntax**update-color-box**Description

This command is used by Boxer to keep the color shown in a color box the same as its red-blue-green percentages. It appears in a modified trigger in the closet of color boxes. Most users are unlikely to use this command. See also the explanation at the beginning of this chapter or in the Boxer Structures document of how sprite properties work.

set-background

Syntax

set-background <color>

Description

This command will set the background of a graphics box to <color>. You must address **set-background** to a graphics box, to a sprite in a graphics box, or execute it inside a graphics box (or near a transparent graphics box). Turtle boxes, as usual, need no **tell**.

Example



clear-background

Syntax

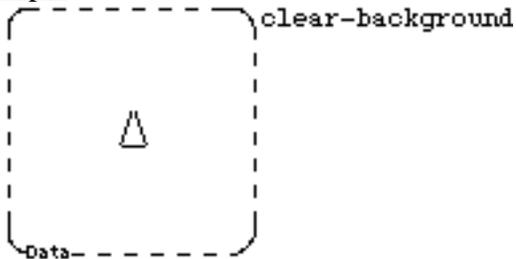
clear-background

cb abbreviation for **clear-background**

Description

This command erases an existing background color or "frozen" background (below) in a graphics box. Compare with **set-background**.

Example



freeze

Syntax

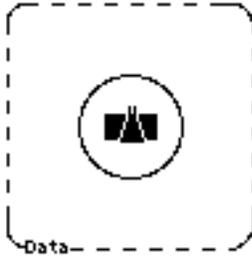
freeze

Description

This command puts the current picture drawing into the background. Then **cs** will not clear it, and the picture could serve as a backdrop for any drawing. **clear-background** will erase what has been frozen. **freeze** is cumulative; it adds any sprite drawing done on top of an existing background to that background. **freeze** makes a good way to produce a background for a video game. Then you can use **bg-color-at=** to check the background to see, for example, if you (a sprite) have run into a (black) mountain and should explode.

Example

freeze



bg-color-under

Syntax

bg-color-under

Description

bg-color-under returns the color of the background in a graphics box at the position of a sprite. It ignores any drawing done on top of the background. You must **tell (ask) <a sprite>** or be in the presence of a turtle box. If the sprite is clipped, i.e., out of the visible portion of the box, **bg-color-under** will not be able to determine the color at that position. In the following example note that the black circle is stamped OVER the background, hence it is not the color seen by **bg-color-under**.

Example



setup

```
ask joe
cb cs pd
set-pen-color black stamp-circle 60
set-pen-color white stamp-circle 40
set-pen-color gray stamp-circle 20
freeze
pu bk 30 set-pen-color black pd
stamp-circle 15
pu home
```

Doit

```
setup
ask joe setxy 0 0
ask joe bg-color-under
```

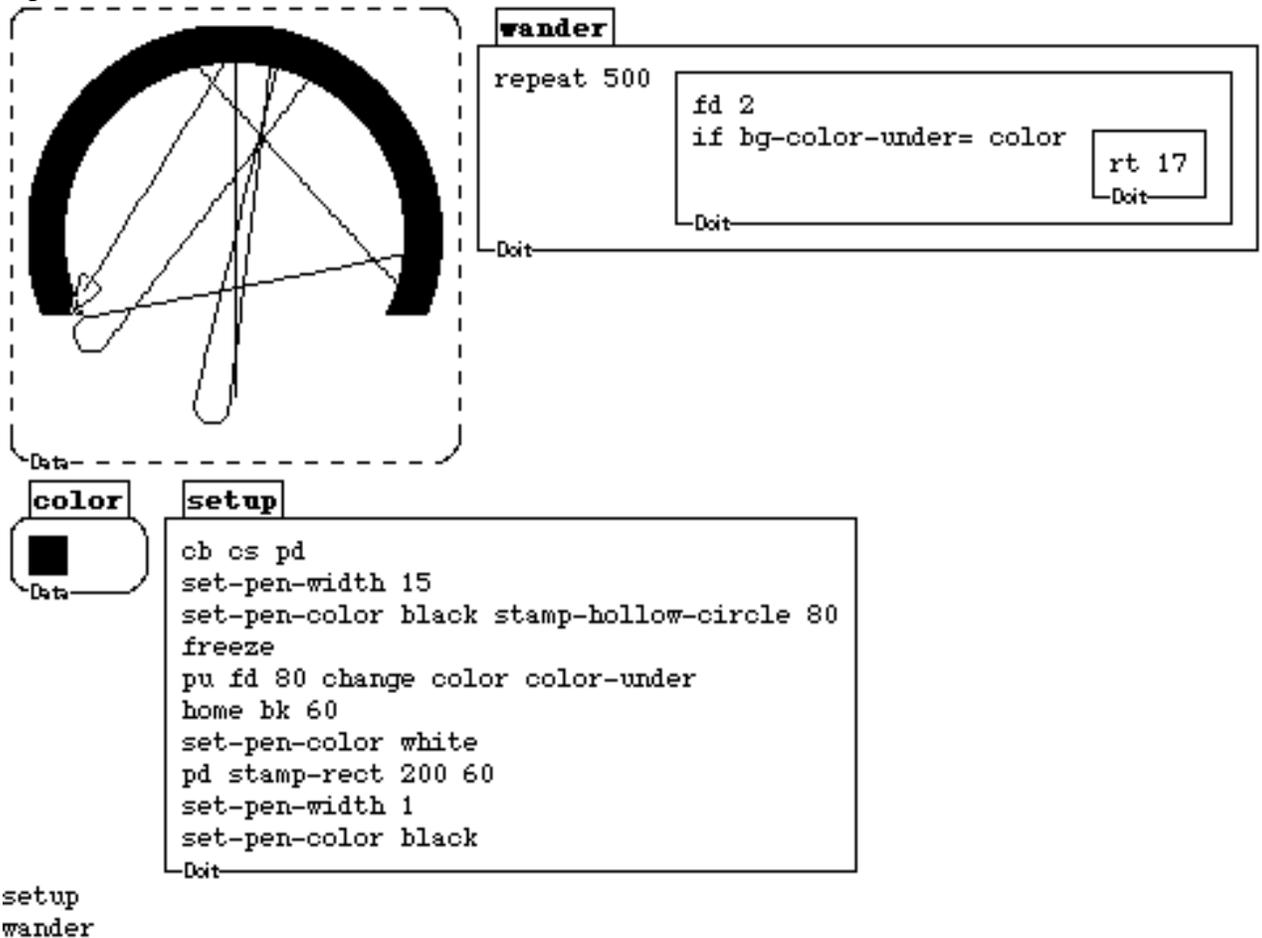


```
ask joe setxy 0 -25
ask joe bg-color-under
```



bg-color-under=Syntax**bg-color-under=** <x> <y>Description

bg-color-under= checks that the color in the background at a sprite's position is the same as the color given as an input to **bg-color-under=**. **bg-color-under=** ignores any colors that have been stamped or drawn over the background. **bg-color-under=** <color> is essentially the same as **color = bg-color-under** <color>, only much faster to execute. As with **bg-color-under**, you must use **tell (ask)** <a sprite> or a turtle box. Note that in the example below the invisible part of the black circle, which has been placed in the background of this box, was merely stamped over with white. **bg-color-over=** still sees the background color.

Example


```

wander
repeat 500
  fd 2
  if bg-color-under= color
    rt 17
Doit

color
[black square]
Doit

setup
cb cs pd
set-pen-width 15
set-pen-color black stamp-hollow-circle 80
freeze
pu fd 80 change color color-under
home bk 60
set-pen-color white
pd stamp-rect 200 60
set-pen-width 1
set-pen-color black
Doit

setup
wander

```

bg-color-under?

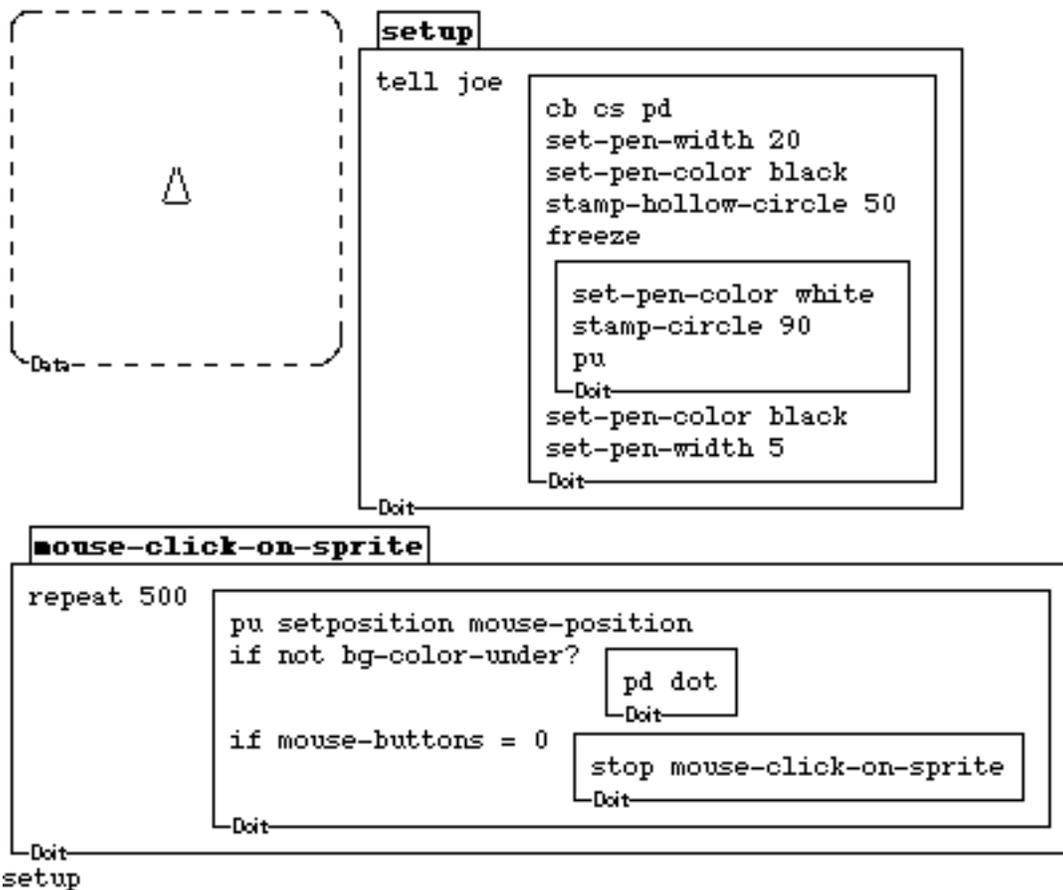
Syntax

bg-color-under?

Description

bg-color-under? checks that the color at a sprite's position is the same as the color of the background at that point. Essentially, it checks whether the position of the sprite has been drawn over with a different color. As with **bg-color-under**, you must use **tell (ask) <a sprite>** or a turtle box. Note that in the example below the sprite will stamp exactly where the color is not the same as the background color (which has been colored over with white).

Example



setup

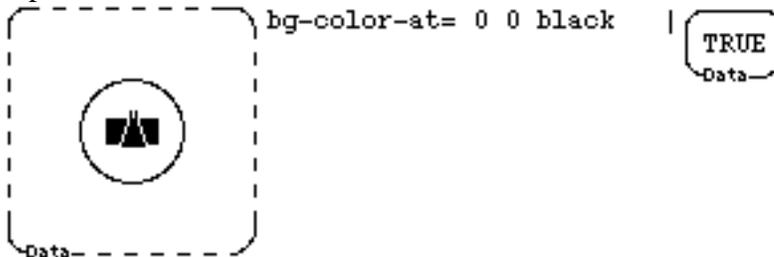
Try: press mouse middle over the sprite and drag it around the graphics box.

bg-color-atSyntax**bg-color-at** <x> <y>Description

Returns the background color at the specified coordinates in a graphics box. You must use **tell**, **ask**, or a turtle box. If the sprite is clipped, i.e., out of the visible portion of the box, **bg-color-under** will not be able to determine the color at that position. If the coordinates specified are out of the visible portion of the box, **bg-color-at** will not be able to determine the color at that position. Note that the background may be a different color than what shows, i.e., you may cover a background with sprite graphics and **bg-color-at** will still return the background.

Example**bg-color-at=**Syntax**bg-color-at=** <x> <y> <color>Description

Checks the background color at the specified position and returns **true** or **false** if it matches <color>. That is, is the color in this position the same as the color given as an input to **bg-color-at=**. As with **color-at**, you must use **tell**, **ask**, or a turtle box. If the coordinates specified are out of the visible portion of the box, **bg-color-at=** will not be able to determine the color at that position. Note that the background may be a different color than what shows, i.e., you may cover a background with sprite graphics and **bg-color-at=** will still check the background.

Example

bg-color-at?

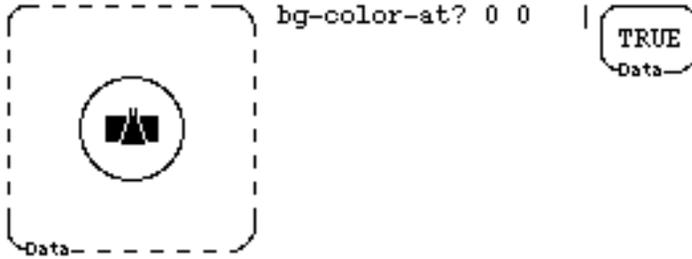
Syntax

bg-color-at? <x> <y>

Description

Returns **true** or **false** depending on whether the color showing at <x> <y> is the background color in a graphics box. You must use **tell**, **ask**, or a turtle box. It tells you if the background (or "frozen" graphics) has been drawn over. If the coordinates specified are out of the visible portion of the box, **bg-color-at?** will not be able to determine the color at that position.

Example



without-recording

Syntax

without-recording <commands>

Description

This is an efficiency hack, not for general use. **without-recording** <commands> will execute the drawing commands that follow it, but no record will remain in the graphics box. Thus a redisplay, or shrink and expand, will cause the drawing to disappear.

Example

reset

```
cs rt 90
bk 100
pu
Doit
```

Instructions

1. Execute Reset: then Try1: or Try2:.
2. Select Redisplay from the Help menu (or shrink and expand the graphics box).
3. Notice that all those stamps are recorded and replayed, unless `without-recording` is used. This is boring if you really mean for them to cancel each other out.

Data

```
reset
Try1: repeat 200
```

```
pr stamp-rect 30 30
stamp-rect 30 30
pu fd 1
Doit
```

```
Try2: without-recording repeat 200
```

```
pr stamp-rect 30 30
stamp-rect 30 30
pu fd 1
Doit
```

3.9 SPRITE INFORMATION & PROPERTIES

x-position

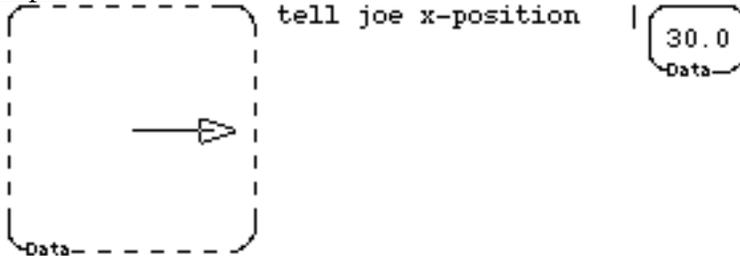
Syntax

x-position

Description

This is the x coordinate of the sprite in its current graphics box, or, if it is a subsprite, its x coordinate relative to its supersprite. **forward**, **back**, **setxy**, and **setpos**, affect **x-position**. All sprite properties work similarly.

Example



y-position

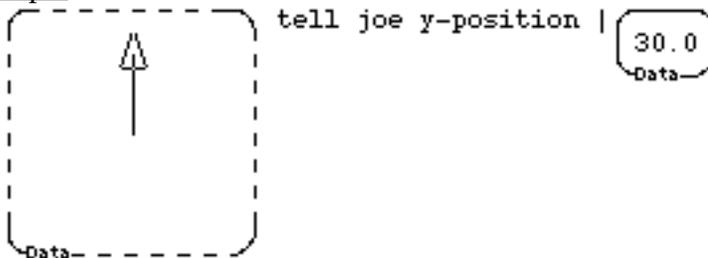
Syntax

y-position

Description

This is the y coordinate of the sprite in its current graphics box, or, if it is a subsprite, its y coordinate relative to its supersprite.

Example



position

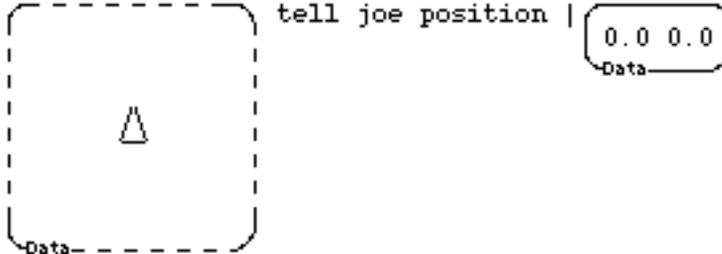
Syntax

position

Description

This returns a box containing the x and y coordinates of the sprite in its current graphics box. If it is a subsprite, the coordinates are relative to its supersprite.

Example



heading

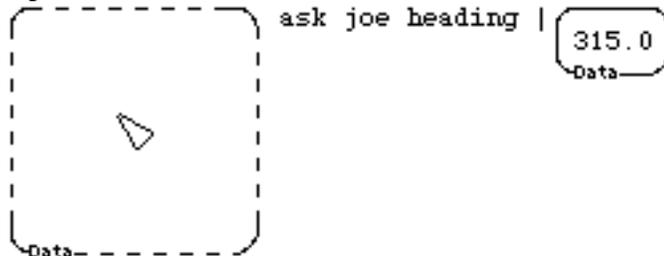
Syntax

heading

Description

Heading is the orientation of a sprite. It is compass heading (zero is up, right is positive) for a regular sprite, and compass heading relative to its supersprite for a subsprite. **right**, **left** and **setheading** affect heading.

Example



shape

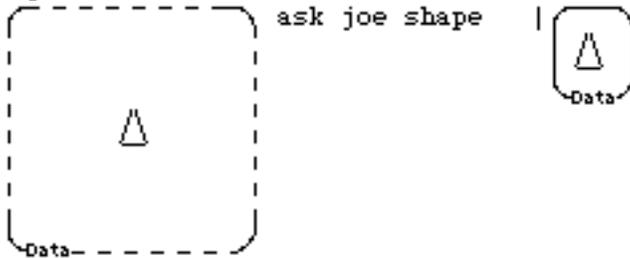
Syntax

shape

Description

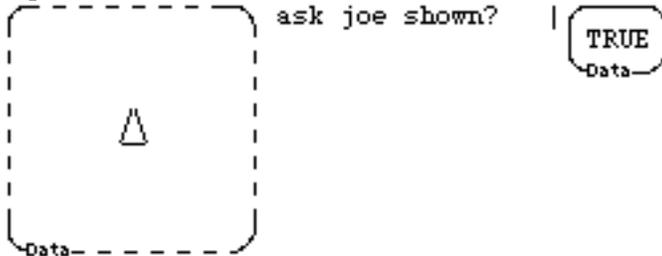
A sprite's **shape** property controls the visual presentation of the sprite. Use **setshape**<graphics box> (or **change shape** ...) to change the shape of a sprite. Normally you will want to have the shape drawn in **reverse** mode, as it will allow moving turtles faster than with pen in **down** mode. The sprite's pen is located at the origin of its shape, the (0,0) home position in the graphics box that defines the shape. The normal turtle shape can be accessed by executing **turtle-shape**. So you can return to the turtle shape with **setshape turtle-shape**

Example

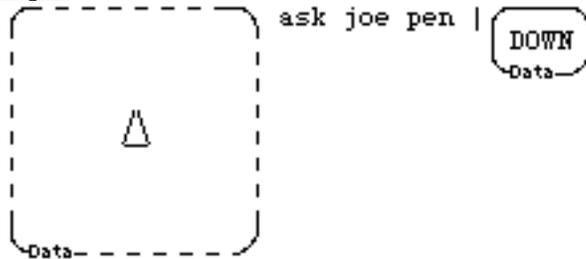


shown?Syntax**shown?**Description

This command tells whether the sprite is seen in the graphics presentation of the enclosing graphics box. It should be **true** or **false**. See **showturtle** and **hideturtle**.

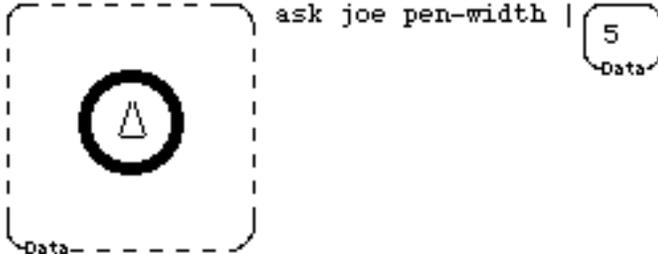
Example**pen**Syntax**pen**Description

This property tells how the sprite's pen draws. It should be *up*, *down*, *reverse* or *erase*. (*reverse* causes the pen to "flip" black to white, and vice-versa, when drawing. *erase* causes the pen to draw in the background color, usually white.). See **penup**, **pendown**, **penreverse**, **penerase**.

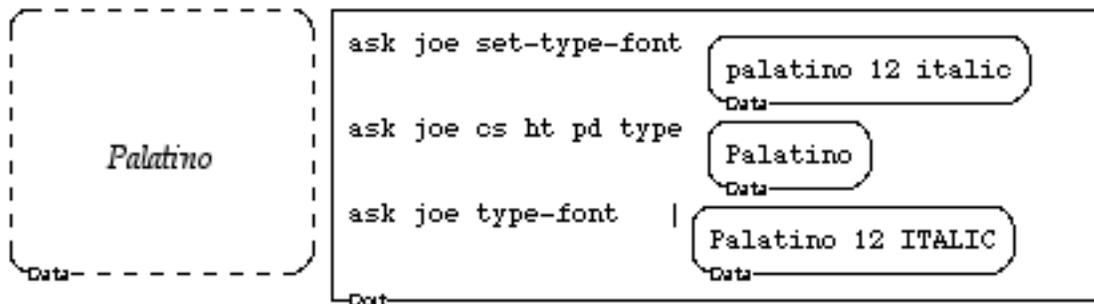
Example

pen-widthsyntax**pen-width**description

This property determines the width of a sprite pen's drawing in pixels. **set-pen-width** changes **pen-width**.

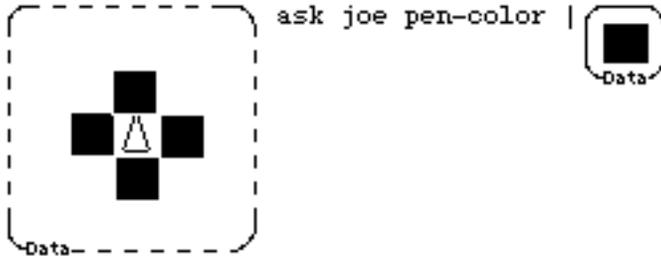
example**type-font**syntax**type-font**description

This property determines the size, and bold and italics properties of the font the sprite types in. This should be font specification (Helvetica 12 bold) or an integer. **set-type-font** changes the font.

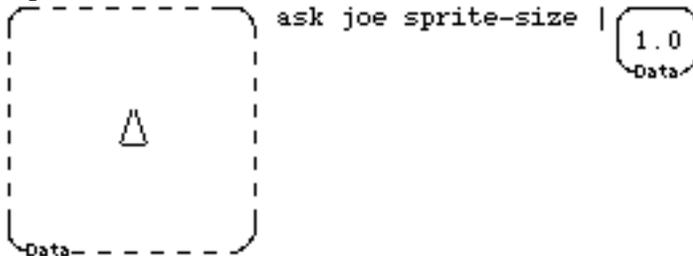
example

pen-colorsyntax**pen-color**description

This property determines the color that the sprite draws in, and the color it will stamp and type with. **color** should be a color box (see **make-color**). **set-pen-color** changes the color.

example**sprite-size**Syntax**sprite-size**Description

You can change the size of a sprite simply by changing this attribute. 1.0 is the standard size. Some parts of a sprites' shape (such as typed text or a bitmap graphic) will not change size. **tell joe set-sprite-size 2.0** will double the size of the sprite.

Example

home-position

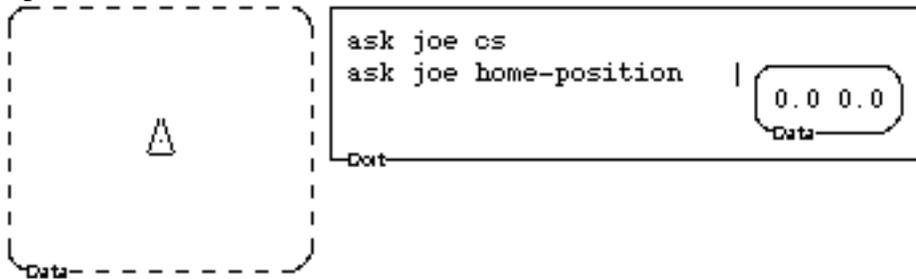
Syntax

home-position

Description

This command shows where the sprite goes when you execute **clearscreen** or **home**. It is a pair of numbers, x and y coordinates. **tell joe change home-position** will change Joe's home position.

Example



3.10 UPDATE PROPERTIES

update-

Syntax

update-heading
update-home-position
update-pen
update-pen-width
update-type-font
update-pen-color
update-shape
update-shown?
update-sprite-size
update-x-position
update-y-position

Description

All update commands are mainly for Boxer's use. They make the visual presentation correspond to what you have changed the sprite property (e.g., X-POSITION, SHAPE) to.

show-sprite-properties

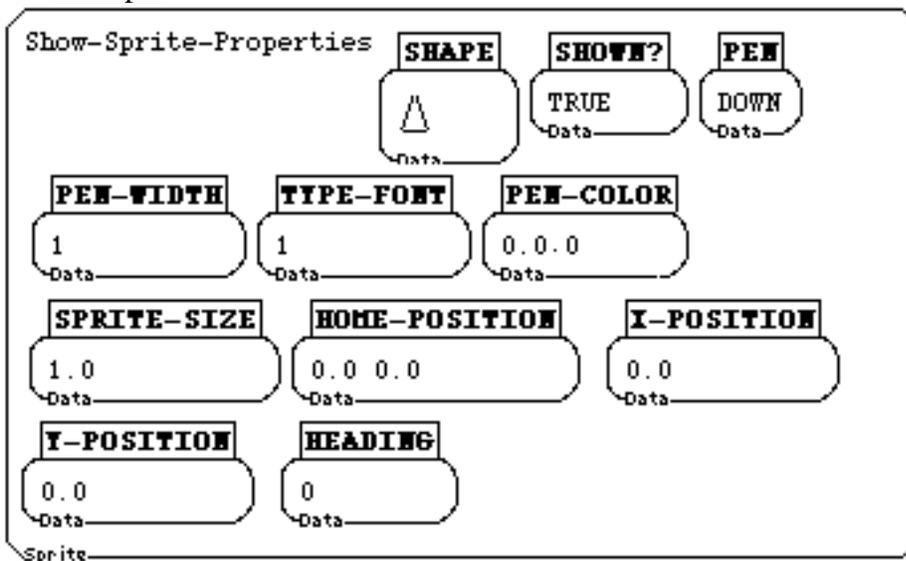
Syntax

show-sprite-properties

Description

When this command is executed in a sprite it will cause all properties to appear (in the closet). Normally only position and heading properties appear in a sprite.

Example



3.11 OTHER INFORMATION

distance

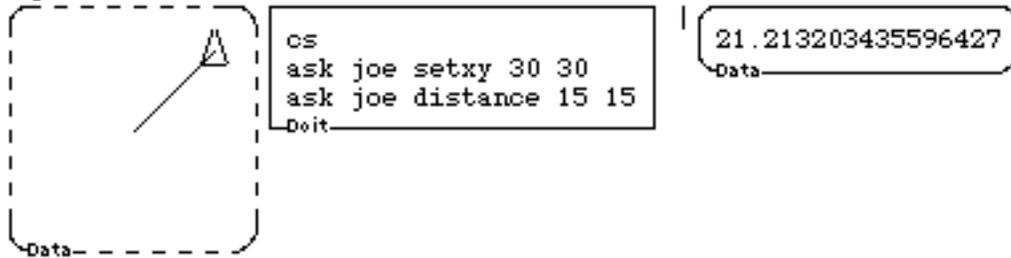
Syntax

distance <x> <y>

Description

This command computes the distance in turtle steps (pixels) between the sprite's current position and the given (x, y) coordinates. You can use this command to find the distance between two sprites.

Example



enclosing-rectangle

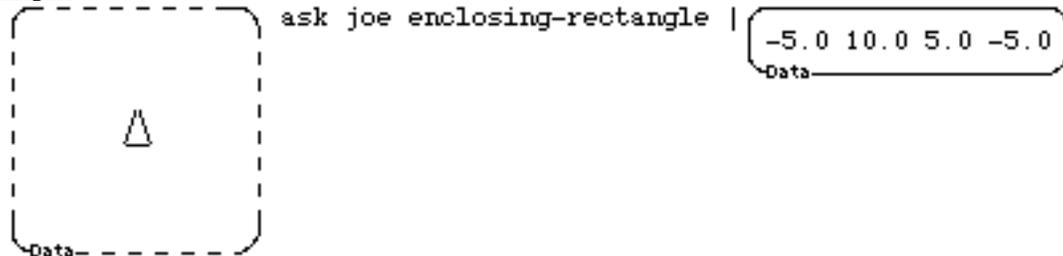
Syntax

enclosing-rectangle

Description

This command returns the sprite's enclosing rectangle (the rectangle that encloses the sprite's shape). The result is a single box in the form <x1 y1 x2 y2> where (x1,y1) is the upper left corner of the rectangle and (x2, y2) is the lower right corner. The enclosing rectangle always includes the origin in the imaginary drawing space of the sprite's shape.

Example



touching?

Syntax

touching? <other-sprite>

Description

This command tells you if the sprite you're talking to is touching another sprite. The command works by checking if the rectangle enclosing one sprite overlaps with the rectangle overlapping the other. Thus, sprites whose shapes are horizontal or vertical lines seem *much* smaller than sprites whose shapes are diagonal lines. The latter kind of sprite may surprise you when you find it is touching another sprite whose shape doesn't *appear* to overlap. See **enclosing-rectangle**.

Example



towards

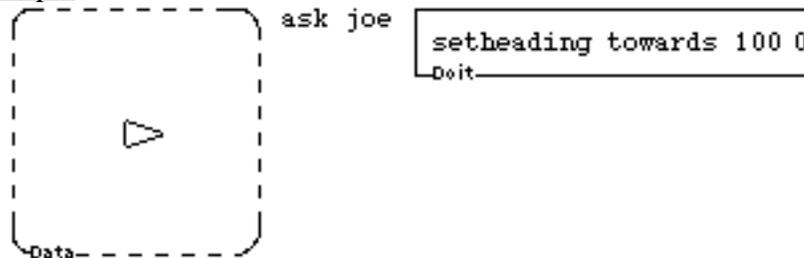
Syntax

towards <x> <y>

Description

This command computes the heading in degrees for the sprite to point in the direction of the given x, y coordinates. You can use this command to make one sprite point towards another sprite.

Example



3.12 SPRITE SIZE, SHAPE & HOME

set-sprite-size

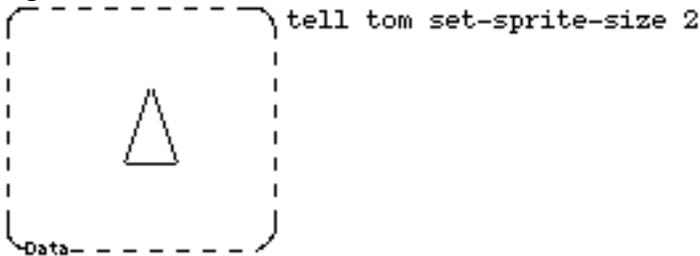
Syntax

set-sprite-size <size>

Description

This command adjusts the graphical size of a sprite. 1.0 is normal and 2.0, for example, means the sprite will appear at twice the size of its assigned shape. See **sprite-size** under *sprite-information*, *sprite-properties*. Note that bitmap shapes or typing will not change in size. However, **stamp-circle**, **stamp-rect**, etc., *will* be affected. Subsprites are also affected by changing a supersprite's size.

Example



setshape

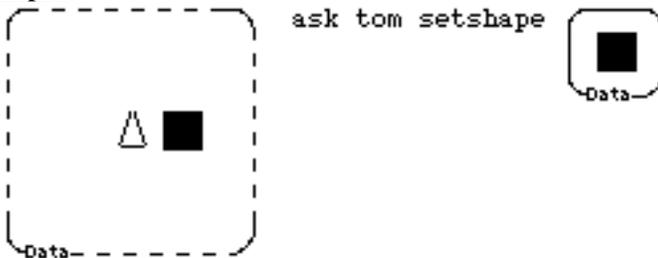
Syntax

setshape <new shape>

Description

This command changes the shape property of a sprite. The input must be a graphics box. It must be addressed to a sprite (with **tell**), executed inside a sprite, or executed in the presence of a turtle. Alternatively you may **change shape** <data box containing turtle commands>. See **shape** under *Sprite Information and Properties*.

Example



turtle-shape

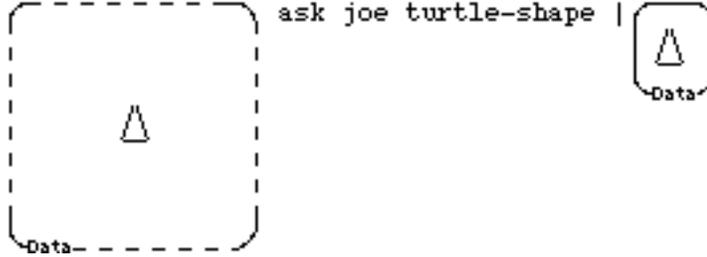
Syntax

turtle-shape

Description

Returns the default shape of sprites. **setshape turtle-shape** returns a sprite to its usual shape. The flip side of the returned graphics box contains the "code" that will draw the shape, if you want to modify it. This is just information, however, and is not linked to the picture in the turtle-shape graphics box. You can execute the code and use **snap** and **set-shape**, or use **change shape** <code>.

Example



set-home-position

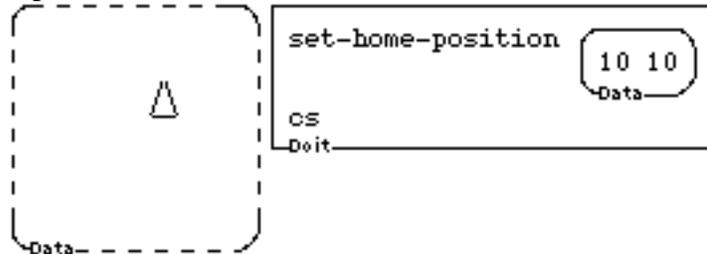
Syntax

set-home-position <new home>

Description

This command determines where a sprite goes when it is issued a **home** or a **cs** command. It must be addressed to a sprite (with **tell**), executed inside a sprite, or executed in the presence of a turtle box. See **home-position** under **sprite-properties** in **sprite-information**.

Example



3.13 MOUSE INPUT & CLICKS

-click-on-graphics

Syntax

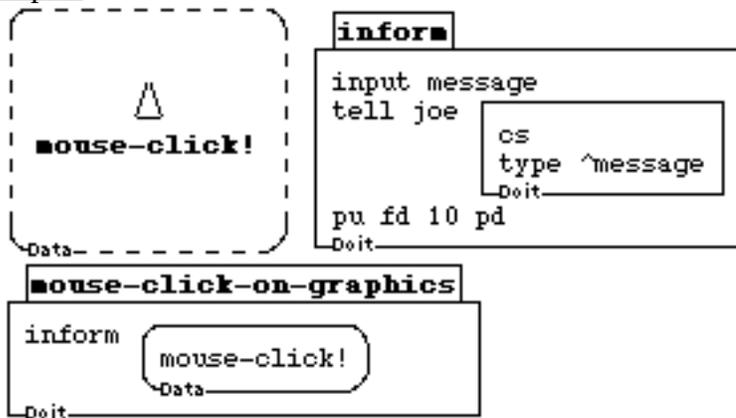
mouse-click-on-graphics

mouse-double-click-on-graphics

Description

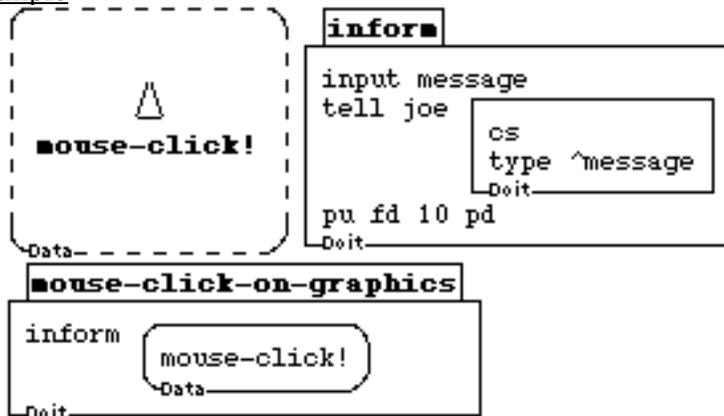
These commands are executed automatically when you click a mouse button over a graphics box in its graphics presentation. They are executed inside the graphics box. The prefixes **command-** and **option-** may also be used. Note that if **mouse-click-on-graphics** is a data box, that box will be returned inside the graphics box, which you will not see until you flip the box to see its box-contents. **mouse-click-on-graphics** and **mouse-click-on-sprite** mouse commands only work when the graphics box is showing the graphics presentation. Regular mouse clicks are activated when showing the regular box presentation side. However, **option-mouse-click-on-graphics** is initially defined to shrink a graphics box. If you want a mouse click command to do the same thing whether it is over a sprite or not, define a **mouse-click-on-sprite** command to do the same thing as any **mouse-click-on-graphics** command you have defined. The obvious place to put these commands is in the graphics box, or in its closet. However, they can be inherited by multiple graphics boxes if they are placed in a containing box.

Examples



-click-on-spriteSyntax**mouse-click-on-sprite****mouse-double-click-on-sprite**Description

these commands are executed automatically when you click the appropriate mouse button over a sprite in a graphics box, when the graphics box is in its graphics presentation. They are executed inside the sprite you clicked on. The prefixes `command-` and `option-` may also be used. Note that if **mouse-click-on-sprite** is a data box, that box will be returned inside the sprite, which you will not see until you flip the box to see its box-contents. **-click-on-sprite** and **-click-on-graphics** mouse commands only work when the graphics box is showing the graphics presentation. Regular mouse clicks are activated showing the box presentation side. The obvious place to put these commands is in the sprite you want them to work with. However, they can be inherited by multiple sprites if they are placed in a containing box, such as the graphics box that contains all the sprites. (As a default, `mouse-click-on-graphics` executes `follow-mouse`, so you can drag a sprite around.)

Example

3.14 MOUSE POSITION

mouse-position

Syntax

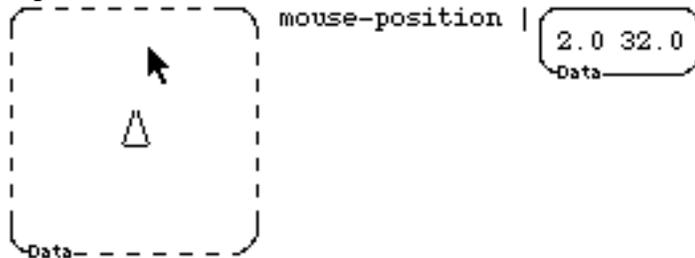
mouse-position

Description

Returns a box containing the x and y coordinates of the mouse cursor. The coordinate system is the same as for sprite positions; (0,0) is at the center of the graphics box. Note: **mouse-position** is a graphics command, and must be used in the presence of a turtle box, or addressed to a sprite or graphics box. **mouse-position** returns coordinates that may be beyond the size of the graphics box if the mouse cursor is outside the graphics box.

Beware of setting the position of a sprite beyond the borders of its graphics box with pen down in wrap mode. Very long lines that endanger the health of Boxer can result. Finally, even when the graphics box is shrunken, the coordinates are relative to (0,0) at where the center of the graphics box would be, if expanded.

Example



mouse-position-on-

Syntax

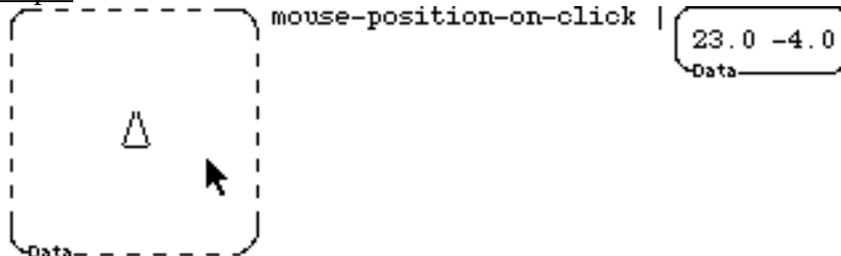
mouse-position-on-click

mouse-position-on-release

Description

These are the same as **mouse-position**, except they wait for either a mouse click or the release of a mouse button before returning the location of the mouse. See **mouse-position** for details. Note a **mouse-position-on-release** waits for the release of a mouse button. If a button is not pressed when it is executed, you (obviously) must press first before releasing.

Example



mouse-x-position

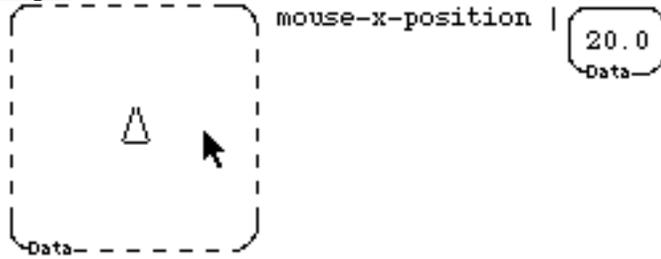
Syntax

mouse-x-position

Description

Returns a box containing the x coordinate of the mouse cursor. The coordinate system is the same as for sprite positions; (0,0) is the center of the graphics box. Note: **mouse-x-position** is a graphics command, and must be used in the presence of a turtle box, or addressed to a sprite or graphics box. **mouse-x-position** is the same as **item 1 mouse-position**. See **mouse-position** for details.

Example



mouse-x-position-on-

Syntax

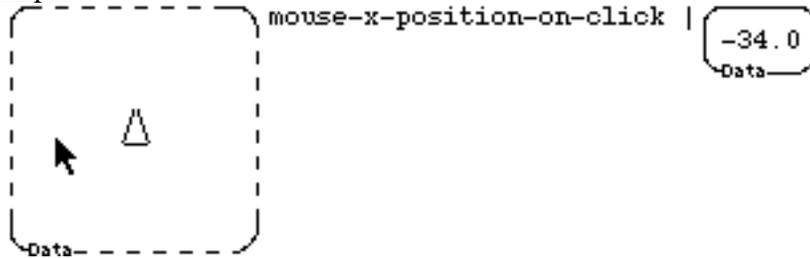
mouse-x-position-on-click

mouse-x-position-on-release

Description

These are the same as **mouse-x-position**, except they wait for either a mouse click or the release of a mouse button before returning the location of the mouse. Note a **mouse-x-position-on-release** waits for the release of a mouse button. If a button is not pressed when it is executed, you (obviously) must press first before releasing. See **mouse-position** for details.

Example



mouse-y-position

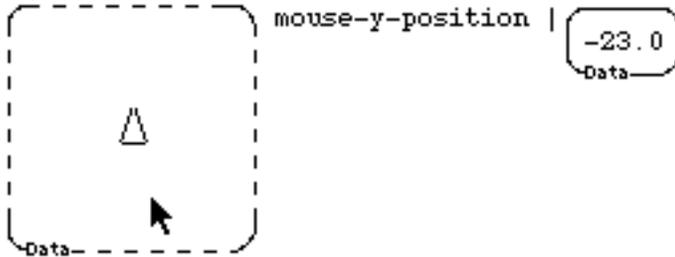
Syntax

mouse-y-position

Description

Returns a box containing the Y coordinate of the mouse cursor. The coordinate system is the same as for sprite positions; (0,0) is the center of the graphics box. Note: **mouse-y-position** is a graphics command, and must be used in the presence of a turtle box, or addressed to a sprite or graphics box. **mouse-y-position** is the same as **item 2 mouse-position**. See **mouse-position** for details.

Example



mouse-y-position-on-

Syntax

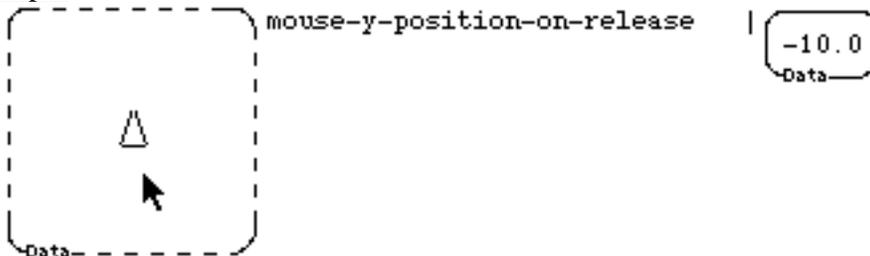
mouse-y-position-on-click

mouse-y-position-on-release

Description

These are the same as **mouse-y-position**, except they wait for either a mouse click or the release of a mouse button before returning the location of the mouse. Note **mouse-y-position-on-release** waits for the release of a mouse button. If a button is not pressed when it is executed, you (obviously) must press first before releasing. See **mouse-position** for details.

Example



CHAPTER 4

Arithmetic & Logic

Boxer has a standard set of number types and operations. There are:

1. integers (e.g., 1, 2, -37)
2. rational numbers (e.g., $2/3$, $29/21$, $195/7$).
3. floating point numbers (e.g., 2.65, 1.0, -73.94827).

Note that $2/3$ denotes a rational number (fraction), NOT a divide; see Arithmetic. Note also that commas are not allowed, even to make big numbers more readable.

There are predicates that tell you what type you have (see *Number-Information*). You can convert from one type to another (See *Number-conversion*).

Numbers are the only kind of data that can be typed outside of a data box. If you execute a number, you get a *datafied* number. And note that rational numbers are reduced to lowest terms, whenever possible.

Automatic type conversion

Arithmetic with integers is not complete in the sense that some operations result in non-integers. E.g., $1 / 2$ is not an integer, but it is a rational number. Boxer comes set up to present calculations that might represent rational numbers as decimals. E.g.: $1/2$ executes to return 0.5.

Any calculation that involves a floating point number forces Boxer to convert from integer to floating point numbers. $1.0 * 2$ returns 2.0.

In addition, you can force Boxer to use fractions to print out the results of rational arithmetic with "**print-fractions true**". (Also Boxer preferences in the Edit menu.) When print-fractions is true, calculations that involve only integers or rational numbers are done perfectly, with no roundoff. (The default setting in Boxer is **print-fractions false**.)

Controlling the number of decimals shown

You can change the number of decimal places used by Boxer with the command **printing-precision**. Printing-precision 2 shows two digits after the decimal place. In Boxer, what you see is what you have, so your calculations are in danger of losing accuracy when you change printing precision to a small number. Boxer does, however, maintain full precision until it has to show you values; so calculations with small precisions lose accuracy only at the end of the calculation, when results are shown to you. To preserve accuracy, you might want to keep a high value of high printing-precision, and then **round** for display separately.

Scientific Notation

Very large or very small numbers may be shown in scientific notation. Thus, 2.01E6 means "2.01 times 10 to the sixth power", which is 2010000. (E stands for Exponent.) You can type in numbers in this format also.

4.1 ARITHMETIC OPERATORS & FUNCTIONS

+ - * / plus minus times divide	Arithmetic in Boxer is fairly straightforward. Arithmetic is one of the few places Boxer allows infix operations, that is, placing the operator between two inputs. Pretty much only +, -, *, / and ** (power) are infix. Even these have prefix forms, if you prefer to use those. Most people write out arithmetic expressions in full detail, using boxes to show subexpressions. Note that parentheses are not understood by Boxer. Also, boxes are better than parentheses because: (1) you can't have unmatched parentheses; (2) you can shrink and expand parts of expressions, or execute them to see their value (just place cursor in box and hit doit); (3) they provide a visually clear parsing of the expression.
--	---

4.2 COMPARISON OPERATORS & FUNCTIONS

= < > <= >= less? greater? less-or-equal? greater-or-equal?	These functions compare the size of two numbers. They return true or false , and are useful for conditionals like if . For each infix command, there is a written out prefix version.: =, <, >, <=, >=; equal? , less? , greater? , less-or-equal? , greater-or-equal?
---	--

4.3 NUMBER TYPE

number? float? integer? rational?	These functions return true or false depending on which type of number is given as input.
--	---

4.4 VALUE INFORMATION

zero? plus? minus? even? odd?	These predicates return true or false depending on whether their numerical inputs are as indicated by the name.
--	---

4.5 OTHER NUMERIC FUNCTIONS

mod remainder abs signum sqrt random max min	These are various useful numeric functions. See also number-conversion for other functions, for example, to round off numbers.
---	--

4.6 EXPONENTIAL & LOG FUNCTIONS

****** Boxer has exponential and logarithmic functions.
power
exp
log
ln

4.7 TRIGONOMETRIC FUNCTIONS

sin These are trigonometric and inverse trigonometric functions. Angles are
cos degrees, not radians.
tan
asin
acos
atan

4.8 NUMBER CONVERSION TO INTEGERS

round These functions convert from various kinds of numbers to others. There are
ceiling several ways to get integers from floating point numbers. **Numerator** and
floor **denominator** give you the specified parts of fractions. **Rationalize** converts
truncate floating point numbers to fractions (rational numbers). These are all ways of
numerator converting rational or decimal (floating-point) numbers to integers that are
denominator "close" to the given number.
rationalize

4.9 NUMBER PRINTING CONTROL

printing-precision You can control how many decimal places Boxer shows or whether a divide of
print-fractions integers shows up as a rational number (1/2) or as a decimal (.5). These
 commands appear in the boxer preferences (Edit menu, preferences option; or
 top-level closet). See also the Overview of Arithmetic and logic, under number
 types (*auto conversion*, and *precision sections*).

4.10 LOGIC

true These are the logical types and operators in Boxer. **true** and **false** are the
false (Boolean) values of all predicates. **not**, **and** and **or** are functions on Boolean
and inputs. These are useful to construct your own predicates (things that return
or **true** or **false**), and to construct inputs to flow-of-control primitives like **if**, **ifs**,
not **unless**, and **when**. **not**, **and** and **or** signal an error if their inputs are not **true** or
false. Note: By convention, Boxer uses a ? at the end of any command that
 returns **true** or **false**. These commands are questions that have **true** or **false**
 answers (aka predicates).

/

Syntax

<number1> / <number2>

DescriptionReturns the ratio of two numbers. **divide** is the prefix version.Examples

```

2/3 / 1/4 | 8/3
              Data
2 / 2/1 | 1
              Data
2/3 / .33333 | 2.0000200002
              Data

```

plusSyntax**plus** <number1> <number2>Description

Prefix version of +.

Examples

```

plus 2 2 | 4
              Data
plus 2/3 1/4 | 11/12
              Data
plus 2/3 .3333 | 0.9999666667
              Data

```

minusSyntax**minus** <number1> <number2>Description

Prefix version of -.

Examples

```

minus 2 2 | 0
              Data
minus 2/3 1/4 | 5/12
              Data
minus 2/3 .33333 | 0.33333666666
              Data

```

times

Syntax

times <number1> <number2>

Description

Prefix version of *.

Examples

```
times 2 2 | 4
           | Data
times 2/3 1/4 | 1/6
              | Data
times 2/3 .33333 | 0.22222
                  | Data
```

divide

Syntax

divide <number1> <number2>

Description

Prefix version of /.

Examples

```
divide 2 2 | 1
           | Data
divide 2 2.0 | 1.0
             | Data
divide 2/3 .33333 | 2.0
                  | Data
```

4.2 COMPARISON OPERATORS & FUNCTIONS

=

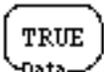
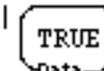
Syntax

<number1> = <number2>

Description

Returns **true** or **false** depending on whether inputs are numerically equal. **equal?** is the prefix version. Both these commands work to compare any box structure, not just numbers. See data manipulation.

Examples

```
1 = 1.0 | 
1/2 = .5 | 
```

<

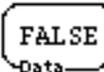
Syntax

<number1> < <number2>

Description

Returns **true** or **false** depending on whether the first input is numerically less than the second. **less?** is the prefix version. Use **alpha<** to compare text alphabetically.

Examples

```
1 < 1.0 | 
2 < 3 | 
```

>

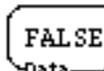
Syntax

<number1> > <number2>

Description

Returns **true** or **false** depending on whether the first input is numerically greater than the second. **greater?** is the prefix version. Use **alpha>** to compare text alphabetically.

Examples

```
1 > 1.0 | 
5 > 4 | 
```

<=

Syntax

<number1> <= <number2>

Description

Returns **true** or **false** depending on whether the first input is numerically less than or equal to the second. **less-or-equal?** is the prefix version.

Examples

```
1 <= 1.0 | (TRUE)
          | (Data)
3 <= 4   | (TRUE)
          | (Data)
```

>=

Syntax

<number1> >= <number2>

Description

Returns **true** or **false** depending on whether the first input is numerically greater than or equal to the second. **greater-or-equal?** is the prefix version.

Examples

```
-2 >= 1.0 | (FALSE)
          | (Data)
2 >= -10  | (TRUE)
          | (Data)
```

less?

Syntax**less?** <number1> <number2>Description

Prefix version of <.

Examples

```
less? 1 1.0 | (FALSE)
            | (Data)
less? 2 3   | (TRUE)
            | (Data)
```

greater?Syntax**greater?** <number1> <number2>Description

Prefix version of >.

Examples

```
greater? 1 1.0 | FALSE
                Data
greater? 5 4   | TRUE
                Data
```

less-or-equal?Syntax**less-or-equal?** <number1> <number2>Description

Prefix version of <=.

Examples

```
less-or-equal? 1 1.0 | TRUE
                   Data
less-or-equal? 3 4   | TRUE
                   Data
```

greater-or-equal?Syntax**greater-or-equal?** <number1> <number2>Description

Prefix version of >=.

Examples

```
greater-or-equal? -2 1.0 | FALSE
                        Data
greater-or-equal? 2 -10  | TRUE
                        Data
```

4.3 NUMBER TYPE

number?

Syntax

number? <number>

Description

Returns true or false depending on whether its input is a number of any sort. **number?** gives an error if it is not given a number as input.

Examples

```
number? pi | FALSE
           |
           |
number? sqrt 2 | TRUE
                |
                |
number? sin 30 | TRUE
                |
                |
```

float?

Syntax

float? <number>

Description

Returns true or false depending on whether its input is a floating point number (i.e., one with a decimal point in it). **float?** gives an error if it is not given a number as input.

Examples

```
float? 2/3 | FALSE
            |
            |
float? sqrt 2 | TRUE
               |
               |
float? sin 30 | TRUE
               |
               |
```

integer?Syntax**integer?** <number>Description

Returns true or false depending on whether its input is a positive or negative integer.

integer? gives an error if it is not given a number as input.Examples

```
integer? -2 | TRUE
              | Data
integer? sqrt 2 | FALSE
                 | Data
integer? cos 0 | FALSE
                 | Data
```

rational?Syntax**rational?** <number>Description

Returns true or false depending on whether its input is a rational number (i.e., a fraction).

rational? gives an error if it is not given a number as input.Examples

```
rational? 2/3 | TRUE
                | Data
rational? sqrt 2 | FALSE
                  | Data
rational? cos 0 | FALSE
                  | Data
```

4.4 VALUE INFORMATION

zero?

Syntax

zero? <number>

Description

Returns **true** or **false** depending on whether its input is equal to zero. As usual, finite precision of computers can occasionally give some surprising results.

Examples

```
zero? .333 - 1/3 | FALSE
                  | Data
zero? -.0001     | FALSE
                  | Data
```

plus?

Syntax

plus? <number>

Description

Returns **true** or **false** depending on whether its input is greater than zero. As usual, finite precision of computers can give some surprising results.

Examples

```
plus? .00001 | TRUE
             | Data
plus? 1/3 - .333 | TRUE
                 | Data
plus? 0 | FALSE
         | Data
```

minus?

Syntax

minus? <number>

Description

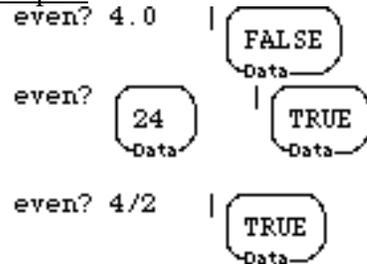
Returns **true** or **false** depending on whether its input is less than zero. As usual, finite precision of computers can give some surprising results.

Examples

```
minus? -.00001 | TRUE
               | Data
minus? .333 - 1/3 | TRUE
                  | Data
minus? 0 | FALSE
          | Data
```

even?Syntax**even?** <number>Description

Returns **true** or **false** depending on whether its input is an even integer. That is, if it is divisible by two evenly.

Examples**odd?**Syntax**odd?** <number>Description

Returns **true** or **false** depending on whether its input is an odd integer.

Examples

4.5 OTHER NUMERIC FUNCTIONS

mod

Syntax

mod <number> <base>

Description

This command does "clock arithmetic" and returns the number a clock with <base> hours on the face (starting at 0 and running to <base> - 1) would read after <number> hours.

For negative numbers, run the clock backwards, but still read the (positive) result off the face. **mod** is similar to remainder, except remainder returns negative numbers for <numbers> greater than 0. **mod** works for fractional and floating point bases too.

Examples

```
mod 0 3 | 0
          | Data
mod 3 3 | 0
          | Data
mod 4 2.5 | 1.5
           | Data
mod -3 2 | 1
           | Data
```

remainder

Syntax

remainder <number> <divisor>

Description

Returns the remainder of dividing <number> by <divisor>. That is, if $R = \mathbf{remainder}$ of <number> <divisor>, then $\text{<number>} = m * \text{<divisor>} + R$, where m is the biggest integer that leaves R positive. **remainder** is similar to **mod**, except **mod** returns a positive number always if the base is positive. **remainder** works for fractional and floating point divisors too.

Examples

```
remainder 0 3 | 0
               | Data
remainder 3 3 | 0
               | Data
remainder 4 2.5 | 1.5
                 | Data
remainder -3 2 | -1
                | Data
```

absSyntax**abs** <number>Description

Returns the absolute value of its input; i.e., it returns x if x is positive, and $-x$ if x is negative.

Examples

```
abs -10.9 | 10.9
           | Data
abs -9/2  | 9/2
           | Data
abs -24   | 24
           | Data
```

signumSyntax**signum** <number>Description

Return +1 if its input is positive and -1 if its input is negative. It's a way to get the "sign" of a number, hence the name.

Examples

```
signum -3 | -1
           | Data
signum 144.8 | 1.0
              | Data
signum 0    | 0
           | Data
```

sqrtSyntax**sqrt** <number>Description

Returns the square root of its input.

Examples

```
sqrt 4 | 2
        | Data
sqrt 2 | 1.4142135623730951
        | Data
sqrt 2/3 | 0.816496580927726
         | Data
sqrt 2.22 | 1.489966442575134
          | Data
```

randomSyntax**random** <number>Description

Returns a number between 0 and one less than the number specified. With floating point numbers, **random** returns a number ≥ 0 and $< n$. Random gives an error on negative inputs.

Examples

```
random 8 | 4
          | Data
random 100 | 76
           | Data
```

maxSyntax**max** <number1> <number2>Description

Returns the greater of its two arguments.

Examples

```
max -3 -7 | -3
          | Data
max 1.345 1.3456 | 1.3456
   Data Data Data
```

minSyntax**min** <number1> <number2>Description

Returns the lesser of its two inputs.

Examples

```
min -3 -7 | -7
      | Data
min 1.4567 1.3 | 1.3
   Data Data Data
```

4.6 EXPONENTIAL & LOG FUNCTIONS

**

Syntax

<number> ** <exponent>

Description

Returns number to the exponent power.

Examples

```
10 ** 2 | 100
          | Data
2/3 ** 2 | 4/9
          | Data
10 **    | log 5 | 4.9999
          | Doit  | Data
```

power

Syntax

power <number> <exponent>

Description

Prefix version of **.

Examples

```
power 10 2 | 100
            | Data
power 2/3 2 | 4/9
            | Data
power 10    | log 5 | 4.99999
            | Doit  | Data
```

exp

Syntax

exp <number>

Description

Returns e (2.718281828459045) to the <number> power. **exp** is the inverse to **ln** (log to the base e).

Examples

```
exp 1 | 2.718281828459045
      | Data
exp 1/100 | 1.010050167084168
          | Data
exp      | ln 5 | 4.999999999999999
          | Doit  | Data
```

logSyntax**log** <number>Description

Returns the logarithm to the base 10 of its input. Note that **log** is the inverse of **power 10** x.

Examples

```
log 1 | 0.0
      | Data
log 4 | 0.6020599913279623
      | Data
log   | power 10 5 | 5.0
      | Doit      | Data
```

lnSyntax**ln** <number>Description

Returns the natural logarithm of its input. That is, the logarithm to the base $e = 2.7182818284590456$. Note that **ln** is the inverse of **exp**.

Examples

```
ln 2 | 0.6931471805599453
     | Data
ln 10 | 2.302585092994046
     | Data
ln    | exp 5 | 5.0
     | Doit  | Data
```

4.7 TRIGONOMETRIC FUNCTIONS

sin

Syntax

sin <angle>

Description

Returns the sine of its input. The inverse function is **asin** (arc sine).

Examples

```
sin 45 | 0.7071067811865475
      | Data
sin 90 | 1.0
      | Data
sin asin .707 | 0.707
      | Data
```

cos

Syntax

cos <angle>

Description

Returns the cosine of its input. The inverse function is **acos** (arc cosine).

Examples

```
cos 45 | 0.7071067811865476
      | Data
cos 90 | 0
      | Data
cos acos .707 | 0.7069999999999999
      | Data
```

tan

Syntax

tan <angle>

Description

Returns the tangent of its input. See **atan** entry concerning inverse.

Examples

```
tan 45 | 0.9999999999999999
      | Data
tan 90 | 1.6324552277619072E+16
      | Data
tan atan 30 1 | 29.999999999999925
      | Data
```

asinSyntax**asin** <number>Description

Returns the arc sine of its input. It's the inverse function of sine. **asin** returns an angle between -90 and 90 as its input varies from -1 to 1.

Examples

```
asin 1 | 90.0
        | Data
asin .707 | 44.99134833716201
          | Data
asin -1 | -90.0
         | Data
```

acosSyntax**acos** <number>Description

Returns the arc cosine of its input. It's the inverse function of cosine. **acos** returns an angle between 0 and 180 as its input varies from 1 to -1.

Examples

```
acos 1 | 0.0
        | Data
acos .707 | 45.008651662838
          | Data
acos -1 | 180.0
         | Data
```

atanSyntax**atan** <y> <x>Description

This command is the arc tangent function. Because the arctangent of an angle does not contain enough information to determine the quadrant of the angle, **atan** takes two inputs. These are the separate y and x components of the tangent, rather than just the ratio, y / x . Multiplying both inputs by a constant does not change their **atan**.

Example

```
atan 0 1 | 0.0
          | Data
atan 1 1 | 45.0
          | Data
atan -1 1 | -45.0
          | Data
```

4.8 NUMBER CONVERSION TO INTEGERS

round

Syntax

round <number>

Description

This command is the traditional rounding to the nearest integer. It rounds toward zero. That is, .5 gets rounded to 0, and so does -.5.

Examples

```
round 4.49 | (4)
             | Data
round 4.5   | (4)
             | Data
round 4.51  | (5)
             | Data
```

ceiling

Syntax

ceiling <number>

Description

This command takes as input any number and returns the smallest integer greater than or equal to that number. If you think of a floor in a building at each integer, **ceiling** returns the integer that is your ceiling if you are at <number>.

Examples

```
ceiling 4.49 | (5)
              | Data
ceiling 4.5   | (5)
              | Data
ceiling -4.9  | (-4)
              | Data
```

floorSyntax**floor** <number>Description

This command takes as input any number and returns the largest integer less than or equal to that number. If you think of a floor in a building at each integer, **floor** returns the integer that is your floor if you are at <number>.

Examples

```

floor 4.49 | 4
             | Data
floor 4.5   | 4
             | Data
floor -4.9  | -5
             | Data

```

truncateSyntax**truncate** <number>Description

This command takes as input any number and returns the integer portion. It simply lops off any digits after the decimal place. Note: **floor** and **truncate** give the same result for positive numbers, but different results for negative numbers.

Examples

```

truncate 4.6 | 4
              | Data
truncate 4.25 | 4
              | Data
truncate -5.99 | -5
              | Data

```

numeratorSyntax**numerator** <number>Description

Returns the **numerator** of the fraction (rational number) given it as input. It first reduces the fraction to lowest terms so that **numerator** \times n / x \times d is the same as **numerator** n / d .

Examples

```

numerator 2/3 | 2
              | Data
numerator 71/31 | 71
               | Data
numerator 5 | 5
            | Data

```

denominatorSyntax**denominator** <number>Description

Returns the denominator of the fraction (rational number) given it as input. It first reduces the fraction to lowest terms so that **denominator** \times $n / x \times d$ is the same as **denominator** n / d .

Examples

```
denominator 2/3 | 3
                  | Data
denominator 71/31 | 31
                   | Data
denominator 5 | 1
               | Data
```

rationalizeSyntax**rationalize** <number>Description

This command produces the rational number, in lowest terms, equal to its input. It is mostly used to force floating point number into rational (fraction) format.

Examples

```
rationalize 37.5 | 75/2
                  | Data
rationalize 2.72828 | 68207/25000
                    | Data
rationalize 100 | 100
                | Data
```

4.9 NUMBER PRINTING CONTROL

printing-precision

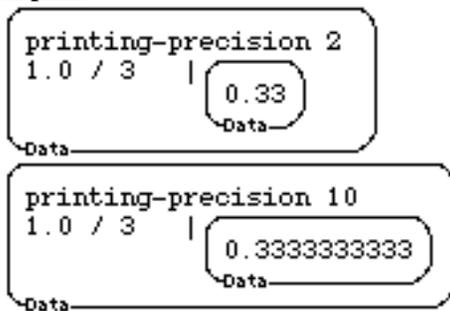
Syntax

printing-precision <new-precision>

Description

This command specifies how many decimal places to show. It does not effect internal calculations until a number is shown on the screen. However, from then on, the accuracy of the number will be reduced. The input, naturally, must be an integer.

Examples



print-fractions

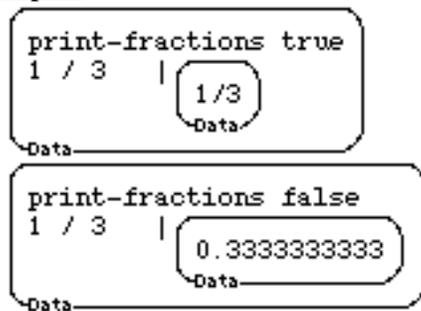
Syntax

print-fractions <true or false>

Description

This command controls whether a divide of two integers (a fraction) will be a fraction or a floating point number. An input of **false** forces printing results as decimals. As with **printing-precision**, this only effects numbers when they get shown on the screen. But once that has occurred, there's no turning back.

Examples



4.10 LOGIC

true

Syntax

true

Description

Returns standard-form Boolean value, the word **true** in a data box. In this way, it is an "automatic" data object that does not need to be placed in a data box, just like a number.

"**IF** <predicate> <consequent> <alternative>" will do the <consequent> if <predicate> returns **true** and the <alternative> if the predicate returns **false**.

Example

```
if true
```

if you get me
if was true
<small>Data</small>

if you get me
if was false
<small>Data</small>

if you get me
if was true
<small>Data</small>

false

Syntax

false

Description

Returns standard-form Boolean value, the word **false** in a data box. In this way, it is an "automatic" data object that does not need to be placed in a data box, just like a number.

"**if** <predicate> <consequent> <alternative>" will do the <consequent> if <predicate> returns **true** and the <alternative> if the predicate returns **false**.

Example

```
if false
```

if you get me
if was true
<small>Data</small>

if you get me
if was false
<small>Data</small>

if you get me
if was false
<small>Data</small>

and

Syntax

and <predicate1> <predicate2>

Description

This command is the logical and of its inputs: **and** returns **true** if both its inputs are **true**. It returns **false** otherwise.

Example

```
between?
```

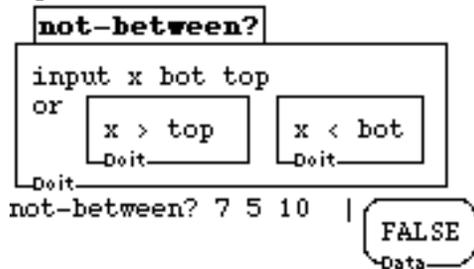
input x bot top
and
x > bot
x < top
<small>Data</small>

```
between? 7 5 10 |
```

TRUE
<small>Data</small>

orSyntax**or** <predicate1> <predicate2>Description

This command is the logical or of its inputs: **or** returns **true** if either or both its inputs are **true**; It returns **false** if neither are.

Example**not**Syntax**not** <predicate>Description

This command returns the negative of its input. It returns **true** if its input is false and **false** if its input is true. Gives an error if its input is not true or false.

Examples

CHAPTER 5

Triggers

Boxer has a set of structures called triggers that allow you to program in an "activation-oriented" style. This means things can be triggered to happen when particular other things occur.

5.1 TRIGGERS

entry-trigger
exit-trigger
modified-trigger

These commands can be set to run when a box is entered, when a box is exited, and when a box is changed (either with the editor or with a **change** command).

5.1 TRIGGERS

entry-trigger

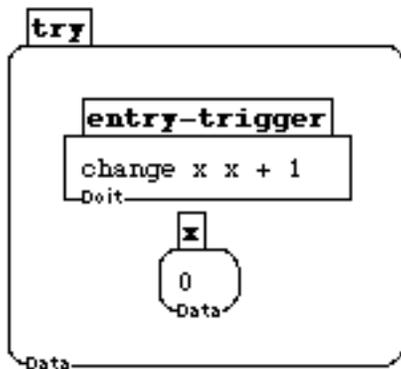
Syntax

entry-trigger

Description

The command **entry-trigger** is executed when you enter the box in which an **entry-trigger** is defined. Boxes inferior to the box in which you define an **entry-trigger** do not inherit the trigger. Entering a subbox of one that has an **entry-trigger** does not count as entering the box, but exiting a subbox (into the interior proper of the box) does count as entering it. Entering a port to a box that contains an **entry-trigger** will also trigger it.

Example



Try: Click middle at various places inside the above box. Notice the trigger works only the first time you enter the box, and only when you enter the box itself -- not a subbox. However if you click in a subbox and then on the interior proper of a box, that counts.

exit-trigger

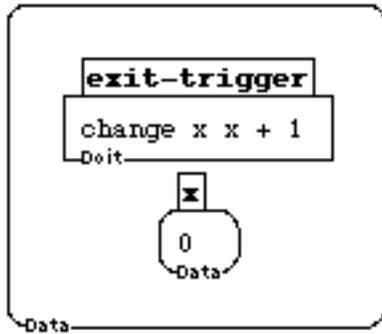
Syntax

exit-trigger

Description

The command **exit-trigger** is executed when you leave the box in which an **exit-trigger** is defined. Boxes inferior to the box in which you define an **entry-trigger** do not inherit the trigger. Exiting a box directly from an inferior box (with the mouse) also triggers the **exit-trigger**. Exiting a port to a box that contains an **entry-trigger** will also trigger it.

Example



Try: Click middle inside the above box, and then exit it. Notice the trigger works whenever you exit the box, even if you exit from a subbox.

modified-trigger

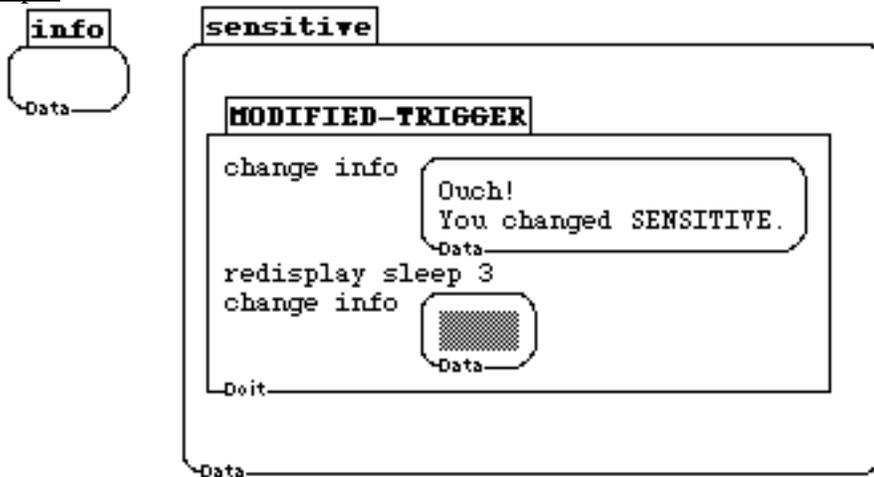
Syntax

modified-trigger

Description

The command **modified-trigger** is executed when you change a box in which a **modified-trigger** is defined. This happens when you change it with the editor, or when you use **change**. If you change a box with the editor, the **modified-trigger** is triggered when you exit the box. Boxes inferior to the box in which you define a **modified-trigger** do not inherit the trigger. Changing a port to a box will also result in a **modified-trigger** being executed.

Example



Try: Click middle inside the above box, make some small change (e.g., type a space, or even just shrink or expand a box) and then exit it.

Notice the trigger works whenever you exit the changed box, even if you exit from a subbox.

CHAPTER 6

Miscellaneous Commands

Commands not covered in the other categories.

6.1 MISCELLANEOUS

beep
click-sound
sleep
unique-symbol

Commands not cover in other categories

6.1 MISCELLANEOUS COMMANDS

beep

Syntax

beep

Description

beep emits a short sound.

Example

```
beep
repeat 5
```

```
beep
sleep 1
Dot
```

click-sound

Syntax

click-sound

Description

Click-sound emits a short click..

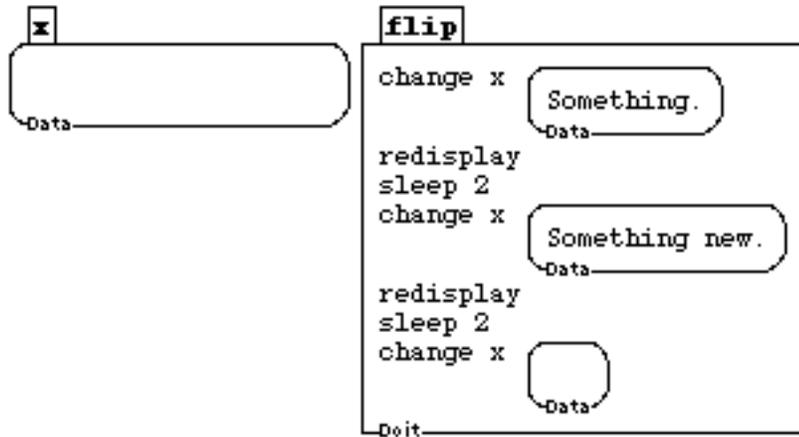
Example

```
click-sound
repeat 5
```

```
click-sound
sleep .2
Dot
```

sleepSyntax**sleep** <seconds>Description

sleep causes Boxer to pause for the number of <seconds> specified. It is useful to delay or slow down Boxer processing so that things can be comfortably viewed. The input may be a fraction or decimal less than one. Note that for the example below, if you remove the sleep command you will not be able to read it.

Example

flip

unique-symbolSyntax**unique-symbol**Description

unique-symbol returns a word that is guaranteed to be unique during the current Boxer session.

Example

```
unique-symbol | UNIQUE-SYMBOL-1
```

The output 'UNIQUE-SYMBOL-1' is shown in a box with a 'Data' field.

CHAPTER 7

Environment: Input & Output

This chapter explains how a program can provide output to the user, and how you can arrange for the user of a program to supply input to the program.

Output of information to the user of a program in Boxer is done simply by making part of Boxer visible, say, a variable or graphics box, and changing that variable or executing graphics commands. The only complication is that, for speed of execution, Boxer does not ordinarily attend to its own display during the running of a program (in contrast to while one is directly editing Boxer, or after some execution has stopped). So you must sometimes tell Boxer when you want changes to be made visible using the command **redisplay**.

Input comes in two classes: keystrokes and input from the mouse. Boxer can provide for the following kinds of input from the user:

4. Essentially any keystroke in Boxer can be rebound (reconnected) to an arbitrary Boxer action by defining a box whose name contains the suffix **-key**. E.g., **a-key**, **command-f-key**, **F1-key**.
5. As with **redisplay**, Boxer does not ordinarily attend to keyboard or mouse input when a program is running, but you can request it to do so with **handle-input**. This can allow users of a program the full editing capabilities of Boxer to provide input.
6. Any Boxer program can "*poll*" (request information about) the state of the mouse buttons, and get information about where the mouse is pointing.

Rebinding keys is a very powerful device, and should be used cautiously. For example, generally it is far better to redefine function keys (F1, etc.) than any keys that would be used for something else. Next in line are the control, command, or option keys; but beware redefining those you use for other things. Finally, it's neat and powerful to define regular letter and number keys, but frequently you will have difficulty continuing to work in your own environment. Anybody else using your environment will have trouble working or changing things if you change frequently used keys, especially things like "doit" and "stop"! Actions on various kinds of mouse clicks can similarly be redefined (see Mouse Input below). Note: Mouse input specific to graphics boxes is described in the graphics section. See **request-for-input-handling** to make key bindings and mouse clicks active during execution of another program.

Mouse Input

Using the mouse as an input device can be done in two ways (in addition to using it to control editing and activation of operations, as usual in controlling Boxer).

1. You may redefine what Boxer does when you click mouse buttons in various places.
2. You may "poll" the mouse to find out various things about its state; such as where it is pointing and which buttons are currently being pressed.

Here are some ideas for coping with input and output in Boxer.

Idea 1

Work as much as possible from Boxer top level. Many times it is unnecessary to program any input or output at all because Boxer shows you the state of your world directly, and has access to any changes you make. So program output might simply be changing a variable that is shown on the screen (or executing some graphics commands). Program input might be having a user change a variable, or chose a "menu item," which might just be a line of text typed on the screen.

Idea 2

Use simple user-defined extensions of the standard input forms. For example, it is simple to arrange for any Boxer action to take place whenever one clicks mouse buttons on sprites or graphics boxes (see the section on graphics). Also, any key may be used as an instant-action function key.

Idea 3

Boxer has a few simple commands to allow you to use any of the tricks in Key Ideas 1 or 2 while a program is running. See **handle-input**, **input?**, and **edit-box**.

7.1 OUTPUT

redisplay
status-line-message
status-line-y-or-n

The first command makes changes that happen in Boxer visible. The other two request or present information to the user.

7.2 INPUT: KEYSTROKE BINDING

-key
command-option-v-
command-''-
command-'-

This suffix is how you redefine what Boxer should do when you press a key. See *Request For Input Handling* to make key bindings active during execution of another program. Press option-help (**option-?**) and press the key to see its **key** name.

7.3 INPUT: MOUSE BINDING

-click
mouse-click-on-

These suffixes are how you define (or redefine) what Boxer should do when your click the mouse. Mouse click redefinition for graphics boxes and sprites, and graphics-specific polling methods are documented in the graphics chapter. See *Request For Input Handling* to make mouse clicks active during execution of another program. Press option-help (**option-?**) and then perform any mouse action to see the mouse- or -mouse- name for it.

7.4 INPUT: MOUSE POLLING

mouse-buttons
mouse-box
mouse-box-on-
mouse-rc-box
mouse-rc-box-on-
mouse-rc
mouse-rc-on-

These commands return information about the mouse when executed. **mouse-buttons** tells you whether and which buttons on the mouse are pressed. The other commands give you information about where the mouse is pointed: either the box it is in (**-box**), the row and column number of the item it is pointing to (**-rc**), or both (**rc-box**). See Environment, **Cursor-location**, for related commands based on the cursor.

7.5 REQUEST FOR INPUT HANDLING

handle-input
input?
edit-box

These commands allow you to get input from a user while a program is running.

7.1 OUTPUT

redisplay

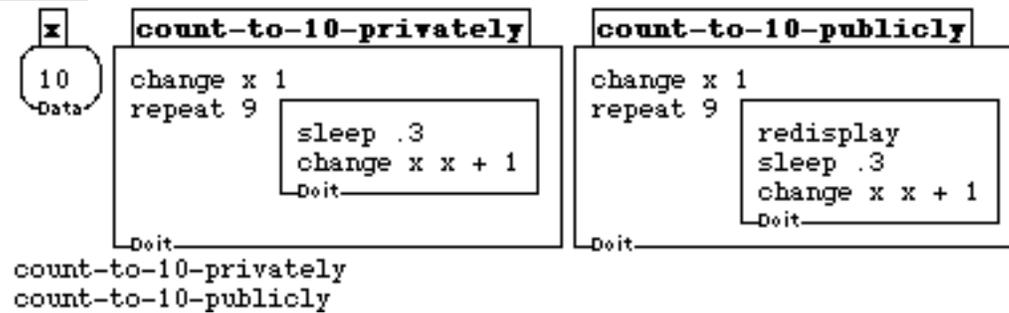
Syntax

redisplay

Description

This command causes Boxer to recompute its presentation to the user so that all changes that have been made by a running program can be seen. Note: You don't need **redisplay** for graphics commands since they cause automatic redisplay of the graphics box.

Examples



status-line-message

Syntax

status-line-message

Description

This command places some text in the Boxer status and information line. It prints only the first line from the input box. Note: The message will disappear whenever Boxer would ordinarily place a new message in the status line.

Examples



7.2 INPUT: KEYSTROKE BINDING

-key

Syntax

-key

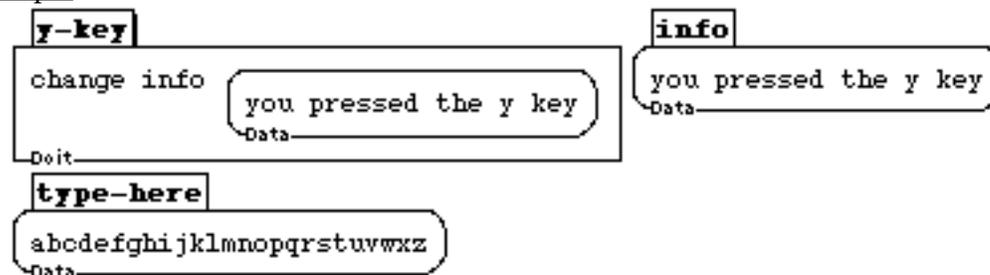
Description

The suffix "-key" added to the name of a key in a box nametab creates a procedure or variable that gets automatically executed when you press the named key. E.g., **a-key** as the name of a procedure will cause that procedure to be run whenever you hit the "a" key. See *Request For Input Handling* to make keybindings active during execution of another program. Press option-help (**option-?**) and press any key to see its **-key** name. Most keys can be rebound simply by adding **-key** to their printed form in the name of a procedure or variable, like **a-key**, **1-key**, **#-key**. However, since Boxer doesn't distinguish upper and lower case under most circumstances, capital letters must use the prefix "**capital**," e.g., **capital-a-key**. Control keys should be prefixed with **command-**, e.g., **command-a-key**, or **option-**, e.g., **option-a-key**. Special keys may have special names, like "**space-key**" for redefining the **space-bar**, **Delete-key** for redefining the delete key, and so on.

In some instances you can avoid problems of redefined keys with the "quote" key. If you press **command-"** (or **command-'**), the next key you press is not run through ordinary Boxer channels, but is inserted directly. You can get keys like **control-a** to print this way. You can also get keys that have been redefined to print.

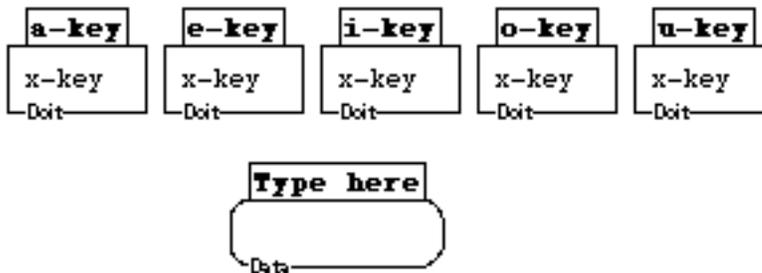
You can temporarily turn off all key redefinitions and mouse redefinitions by entering "Top Level" input mode. Use *Other* pulldown, Top Level (Local) *Key/Mouse* Mode selections to control this. Or **command-option-v** ("v" for "vanilla"--ordinary mode) will enter Top Level mode, during which keys and mouse clicks behave as they do in a "bare" Boxer. Pressing **command-option-v** again, or **stop** exits to "Local" mode.

Example



command-option-v-keySyntax**command-option-v-key**Description

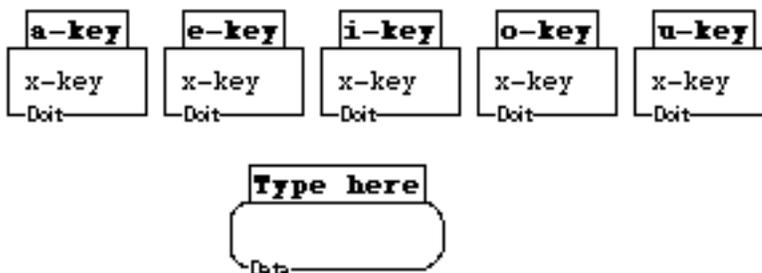
This keystroke causes key and mouse bindings to revert to "vanilla" bindings. That is, it turns off any redefined keys or mouse clicks. It is most useful to type in a box in which many keys have been rebound. The command is usually executed by pressing the **command-option-v** key. Pressing the **stop (command-g)** key or **command-option-v** again turns off its effect. Use also *Other* pulldown menu, *Key/Mouse* selection.

ExamplesNote

In the example above, all vowel keystrokes have been redefined. It is virtually impossible to type anything in the box, including any more key redefinitions. BUT, press the **command-option-v-key** and you get back "vanilla" key bindings until you press **STOP, command-g** or **command-option-v** again.

command-''-key**command-'-key**Syntax**command-''-key****command-'-key**Description

This keystroke causes key bindings to revert to "vanilla" bindings for one keystroke. It is like **command-option-v**, except for only one key press. It is most useful to type in a box in which many keys have been rebound. In the following example, all vowel keystrokes have been redefined. It is virtually impossible to type anything in the box, including any more key redefinitions. But, press the **command-''** (or **command-'**) key and you can type a vowel at a time.

Examples

7.3 INPUT: MOUSE BINDING

-click

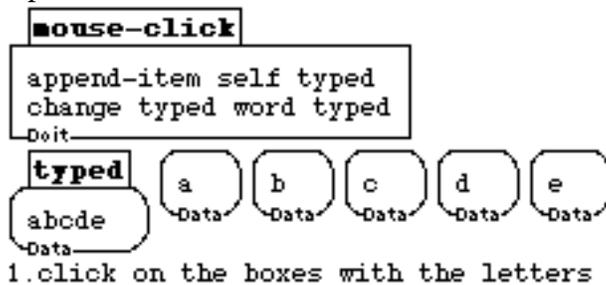
Syntax

mouse-click
option-mouse-click
command-mouse-click
mouse-double-click
option-mouse-double-click
command-mouse-double-click

Description

Boxer executes commands by these names when you click the mouse. Under ordinary circumstances, these do editor operations (expand box, shrink box, etc.), but you can change that by defining new procedures that have these names. Note: You may use **command-** or **option-** for other possibilities. See the graphics section for graphics specific click rebinding. Note: You can turn off mouse key bindings with the "vanilla mode" command, **command-option-v**. In the example that follows, when you click on any of the boxes with the letters, their value will be added to the boxes named "typed."

Example

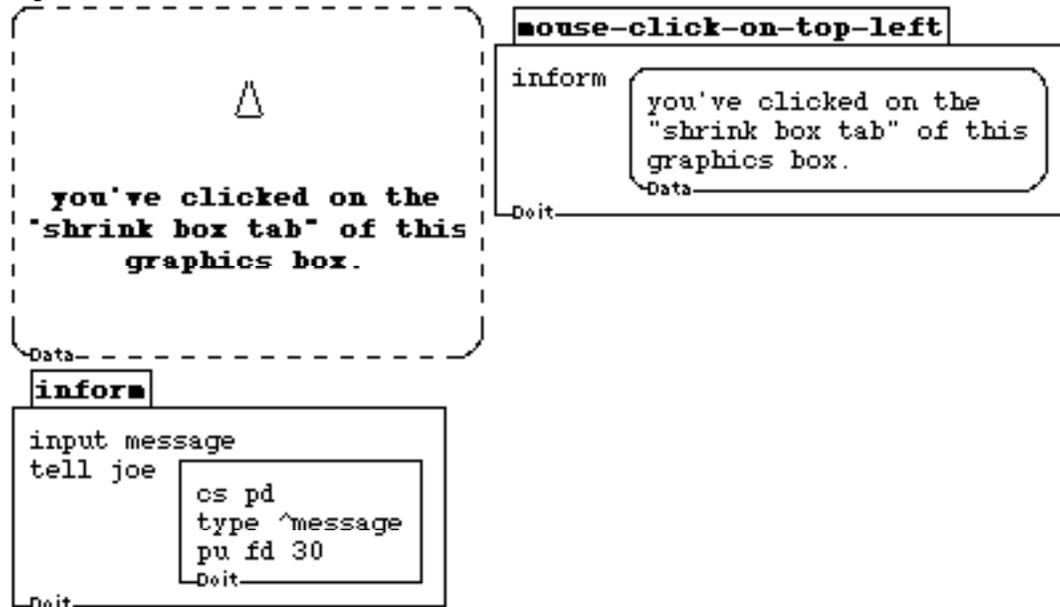


mouse-click-on-Syntax

mouse-click-on-top-right
mouse-click-on-top-left
mouse-click-on-bottom-right
mouse-click-on-bottom-left
mouse-click-on-type
mouse-click-on-name
mouse-click-on-scroll-bar
mouse-double-click-on-

Description

Boxer executes commands by these names (e.g., **mouse-click-on-top-right**) when you click the mouse in the "hot spots" indicated. Under ordinary circumstances, these mouse actions do editor operations (flip the *view*, open/close the *closet*, allow you to grasp the lower right corner of a box to *resize*, etc.), but you can change that by defining new procedures that have these names. **click** may be replaced by **double-click**. **control-** and **option-** prefixes *may* work, depending on window system and implementation. Use vanilla-mode (**command-option-v**) to revert temporarily to standard mouse bindings. In the example below, once the procedure **mouse-click-on-top-right** has been defined, any time you click on any top-right corner of any box the message "you clicked in the top right corner of a box" will be displayed in the X box.

Example

1. Click on the top left corner of the graphics box.

7.4 INPUT: MOUSE POLLING

mouse-buttons

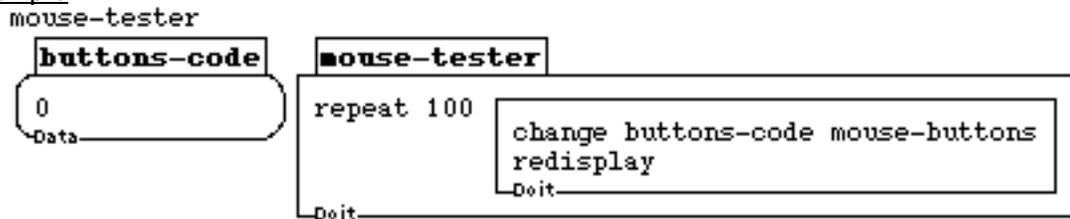
Syntax

mouse-buttons

Description

This command tells you if any buttons are pressed and, if so, which ones. The code is "binary digits": 0 for no buttons, 1 for option-mouse-click (Unix: left), 2 for mouse (Unix: middle), 4 for command-mouse-click (Unix: right), and the sum of these codes for multiple simultaneous presses.

Example



NOTE: Once mouse-tester is executed, the codes 0,1,2, and 4 will be display in buttons-code depending on which mouse button you press.

mouse-box

Syntax

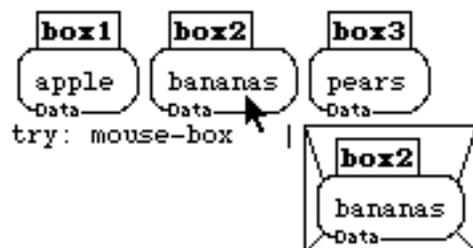
mouse-box

Description

This command returns a port to the box in which the mouse cursor is currently placed. Placing the mouse cursor in a port and executing mouse-box gets a port to the target of the port you pointed to.

Example

1. Put the typing cursor in the "try" line.
2. Place the mouse arrow on one of the boxes but don't click.
3. Press the "doit" key.

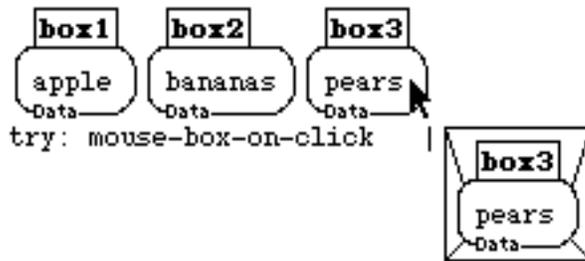


mouse-box-on-Syntax**mouse-box-on-click****mouse-box-on-release**Description

Like **mouse-box**, these commands return a port to the box in which the mouse cursor is placed. **mouse-box-on-click** waits until you click a mouse button before looking where the mouse is, and **mouse-box-on-release** waits until you release a pressed mouse button. If a mouse button is not pressed when you execute **mouse-box-on-release**, it will wait until you press a button, and *then* wait for a release. Placing the mouse cursor in a port and executing either of these gets a port to the target of the port you pointed to.

Example

1. Execute the "try" line. Then click on one of the boxes.



mouse-rc-boxSyntax**mouse-rc-box**Description

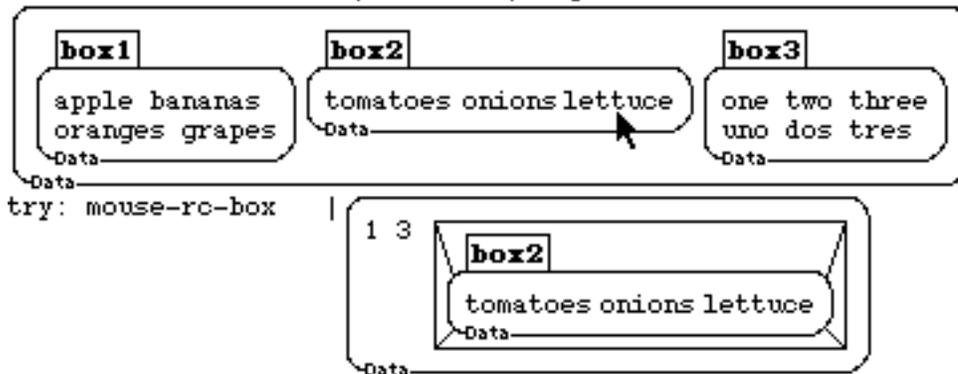
Returns a box containing, in sequence:

1. The row number in that box that the mouse is pointing to.
2. The column number (item number on the pointing-to row) of the nearest item in the box.
3. A port to the box in which the mouse cursor is pointing.

Note: Name is mnemonic for the sequence -- **rc-box** means **row, column, box**, in that order. **mouse-rc-box** uses the same algorithm to decide what item is closest to the mouse that the Boxer editor does. Pointing at white space gets you the item closest to where the cursor would be if you just clicked middle where you were pointing. Placing the mouse cursor in a port and executing either of these commands gets a port to the target of the port you pointed to.

Example

1. Put the typing cursor in the "try" line.
2. Place the mouse arrow on one of the words inside the boxes but don't click.
3. Press the "enter" (SUN: doit) key.



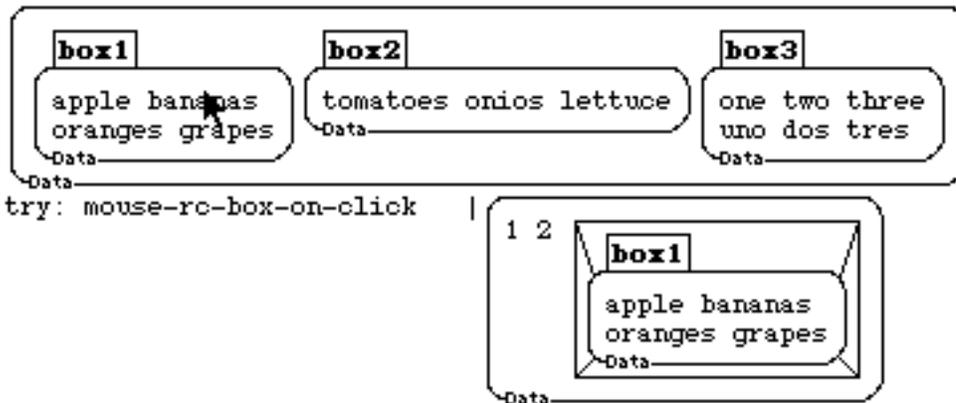
mouse-rc-box-on-Syntax**mouse-rc-box-on-click****mouse-rc-box-on-release**Description

Like MOUSE-RC-BOX, these commands return the row and column numbers of the item pointed to by the mouse, and a port to the box in which the mouse cursor is placed.

mouse-rc-box-on-click waits until you click a mouse button before looking where the mouse is, and **mouse-rc-box-on-release** waits until you release a pressed mouse button.

Example

1. Execute the "try" line. Then click one of the words inside the boxes.

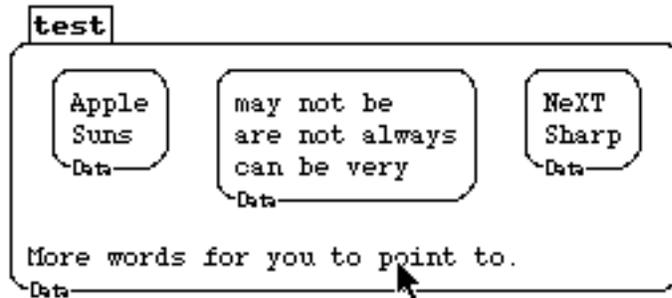


mouse-rcSyntax**mouse-rc**Description

mouse-rc <box> returns a box containing the row and column numbers of the item in <box> that you are pointing to with the mouse. In contrast to **mouse-rc-box**, **mouse-rc** specifies the box that you are to get **rc** numbers with respect to. So pointing anywhere in a subbox of <box> will get you the same **rc** numbers for **mouse-rc** <box> while **mouse-rc-box** gives you numbers with respect to the particular box you happen to point to, and will change if you move around inside a sub box. Placing the mouse cursor in a port and executing either of these commands gets a port to the target of the port you pointed to.

Example

1. Place the mouse typing cursor on the "Try" line below.
2. Move the mouse arrow over one of the words in the following box (but don't click there).
3. Then press "doit" to execute the "Try" line.



Try: mouse-rc test | | 3 6

mouse-rc-on-Syntax

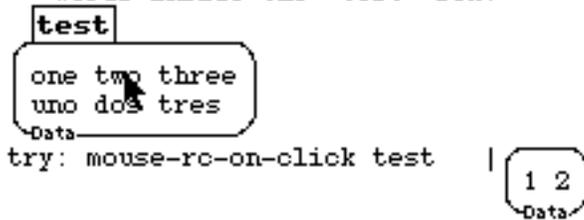
mouse-rc-on-click
mouse-rc-on-release

Description

Like **mouse-rc-box**, these commands, when executed, return the row and column numbers of the item pointed to by the mouse, and a port to the box in which the mouse cursor is placed. Except **mouse-rc-box-on-click** waits until you click a mouse button before looking where the mouse is, and **mouse-rc-box-on-release** waits until you release a pressed mouse button. If a mouse button is not pressed when you execute **-on-release** command, it will wait until you press a button, and *then* wait for a release. Placing the mouse cursor in a port and executing either of these commands gets a port to the target of the port you pointed to.

Example

1. Execute the "try" line. Then, click on one of the words inside the "test" box.



7.5 REQUEST FOR INPUT HANDLING

handle-input

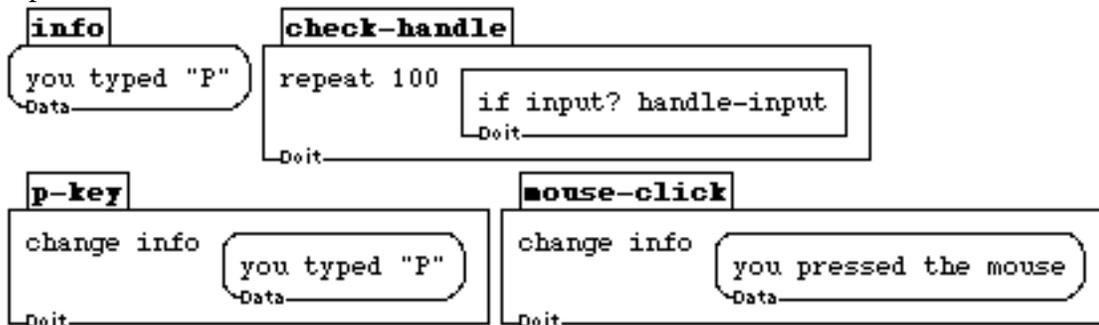
Syntax

handle-input

Description

This command causes Boxer to attend to mouse clicks and keystrokes. It is useful when you want user input while a program is running. **handle-input** will wait for a mouse click or keystroke if one is not already "waiting". Therefore, it is almost always used as in: **if input? handle-input**, which will not "hang" (get stuck and wait) if there is no input. Note: if you're using **Unix** then use the command **mouse-right**. **handle-input** will cause key presses or mouse clicks to be handled, one each time it is executed, in the order in which they came. You are not allowed to do a "doit" (or mouse-middle-twice) during a handle-input. Any input events pending, but not used by **handle-input**, will be saved and executed when a running procedure stops. You can use the "waits for an input if none already there" property of **handle-input** to have a command that is executing wait for a signal (any mouse-click or key-stroke) before continuing.

Example

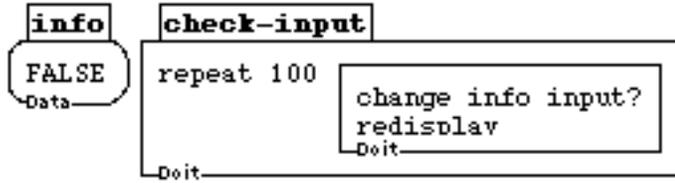


input?Syntax**input?**Description

Returns true or false depending on whether or not the user has generated some input (typed a key or pressed a mouse button) since **input?** was last handled. **handle-input** will attend to any pending input, one item each time it is executed. When all pending inputs are taken care of, **input?** will become false again.

Example

```
check-input
```



edit-box

Syntax

edit-box <box>

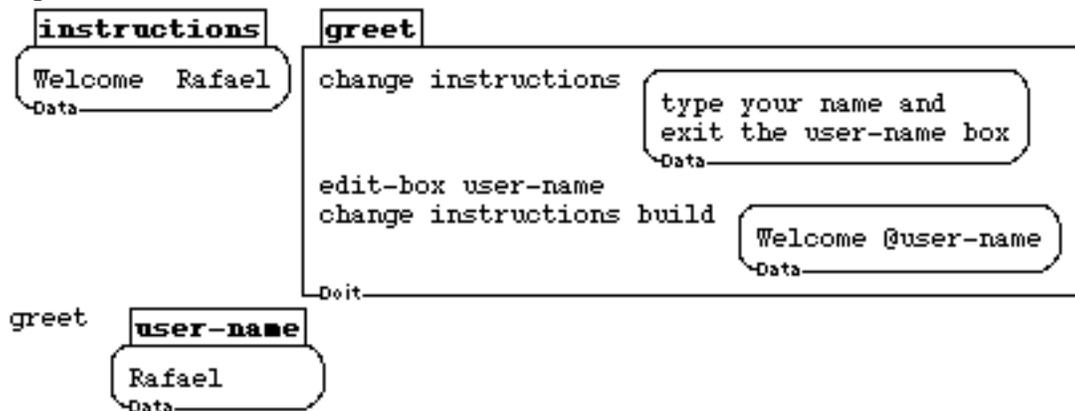
Description

This command places the cursor in <box>, then allows the user to type, until the box is exited. It is used during the running of a program to allow the user to input data that will thereafter be used. You can essentially do the same thing as an **edit-box** by simply stopping your program, letting the user type something in the Boxer world, and have him/her restart the program with some signal (e.g., a menu click, "continue-when-done-with-typing", or binding a key in that box to restart the program).

- If the input box is on-screen, but closed, it will be opened.
- If the input box is off-screen, you will get an error.
- If the input box is a graphics box in graphics presentation, you won't see the typing. (This may actually be handy if you don't want the typing to show, e.g., you have some instant keystrokes in the box.)

Note: You can actually do a lot while **edit-box** is running. You can type, execute procedures, or whatever. **edit-box** will keep running until you exit the edited box.

Example



CHAPTER 8

Environment: File Commands

When you Open a file into Boxer, it appears as a box right where your typing cursor is. So you may need to think a moment about where you want a file to appear. File boxes appear with double-thick borders. When the typing cursor is in a file, the file box's name and on-disk file name appear in the Boxer window title bar.

File menu "Save" saves the first superior file from where your typing cursor is currently located. The same for Save As. Any box in Boxer may be saved as a file. The "Save Box As" menu option saves the box in which your typing cursor currently sits. You may save a plain text box as a text file with the Save As option. Note that Boxer usually saves a backup file with a ~ suffix (adjustable in Preferences) for safety. You may also open plain text files from other applications into Boxer.

Boxer also provides a general set of commands that may be used to save, open and delete files; to read in directories, and to set the current directory (folder) for opening, saving and deleting. Beginning users rarely need these commands, but use File menu versions.

Networking Files

File boxes and network boxes (which read in from remote machines over the network) are almost identical. For example, the command **open** works with both. See Networking Chapter for details.

Subfiles

Subfiles will appear as black boxes (or boxtop icons) when you read the superior file back in. They load automatically when clicked or double clicked. You may wish to use this feature to organize all or most of your Boxer work from within Boxer. If you change the file name or location of a subfile, save the containing file also so that the new location will be available when the containing filebox is read in.

Box Properties

Box Properties ("Other" menu) allows you to adjust some file parameters, such as the file name, whether the file is read only, and whether the file should be automatically read in with its superior file. You can also unlink file boxes, hence turn them into regular boxes. If you try to save a different box to the same filename, Boxer will warn you.

Paths to files

In order to specify the exact location of a file with **save-as** or **open** Boxer commands, use a "path." (A path is a sequence of nested directories (folders.) On the Mac, the path separator is ":" as in "Hard Disk:Desktop Folder:Boxer Release Work:File" Unix notation uses "/" for separator. The latter is useful for network paths (net boxes).

Links to non-Boxer files

The "File" pulldown menu, Link to Mac File selection, will allow you to create a link in Boxer to an external file. A double-click on that link will activate the file, as if you had double clicked on it directly. The link will appear as the normal file's icon and name.

Note

The commands in this chapter will not be needed by most users. Use the File pulldown menu instead.

8.1 STANDARD FILE COMMANDS

open
save
save-as
save-box-as
delete-file
choose-file
directory
current-directory
set-current-directory
save-text-file
read-text-file

Standard command files are programming alternatives to Menu commands in Mac Boxer.

8.1 STANDARD FILE COMMANDS

open

Syntax

open <file-name>

Description

open takes a file name (in a box) as input and returns the Boxer box that has been saved under that name. If the box had a name, Boxer tries to preserve the name. If **open** is given a file name without a path, it looks in the **current-directory**. If it is given a complete path + file name, the current directory is changed to that of the file opened.

open is also used to open in "net boxes" over the Internet. See also **open** in the networking chapter. Note that the third example is opening a directory.

Examples

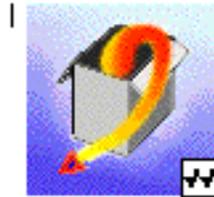
open

a-file
Data

| A-FILE
THIS IS MY FILE
Data

open

ftp://soe.berkeley.edu/pub/boxer/hub.box
Data



open ftp://soe.berkeley.edu
Data

|

- .logout**
Data
- .profile**
Data
- MRC Test**
Data
- bin**
Data
- etc**
Data
- fred.box**
Data
- prorder.wri**
Data
- pub**
Data

Data

saveSyntax**save**Description

save saves the first superior file box to its corresponding file. Ordinarily Boxer is set up to keep one previous file as backup. If the file saved is under the name file, the backup will be called file~. **save** can also work for net boxes, provided you can connect to the appropriate machine and have permissions or passwords. See also **save** in the manual networking section. **save**, in a port, will save the target of the port if it is a file box, or otherwise, the first superior file box from the target. Ports are preserved in files if their target is within the current file. If the target is outside, the connection will be lost.

Examples

Setup: `tell stuff save-box-as` `test-file` ; creates a file from STUFF

```

Data
  tell stuff save-box-as test-file
Data

stuff
Just some junk.
You can edit me before doing any save, below.

save
  Try: save

  Note: This save also saves its first
  superior file box, STUFF. It does NOT
  save the box it is in unless it is a
  file box.
Data

```

Note: You can use TELL STUFF SAVE to save without entering STUFF.

save-asSyntax**save-as** <file-name>Description

save-as saves the first superior file box to a file named by its input. **save-box-as** saves the local box where the command is executed, whether or not that box is already a file box. Ordinarily Boxer is set up to keep one previous file as backup. If the file saved is under the name file, the backup will be called file~. **save-as** can also work for net boxes, provided you can connect to the appropriate machine and have permissions or passwords. **save-as**, in a port, will save the target of the port if it is a file box, or otherwise, the first superior file box from the target. Ports are preserved in files if their target is within the current file. If the target is outside, the connection will be lost. See also **save-as** in the networking section of this manual.

Example

```

stuff
Just some junk. The line below saves this box
as a file.

save-as  (new-junk)
Data_____

```

save-box-asSyntax**save-box-as** <file-name>Description

save-box-as saves the box it is executed in (whether or not it is already a file box) to a file named by its input. It can take a full path specification. Ordinarily Boxer is set up to keep one previous file as backup. If the file saved is under the name file, the backup will be called file~. **save-box-as** can also work for net boxes, provided you can connect to the appropriate machine and have permissions or passwords. See also networking section of this manual. An error results from a non-existent file directory, etc. Network saves are subject to such things as establishing a connection to the appropriate machine.

Example

```

stuff
Just some junk. The line below saves this box
as a file.

save-box-as (junk)
Data_____

```

delete-fileSyntax**delete-file** <file-name>Description

delete-file deletes the specified file from disk. **delete-file**, like **save** and **read**, uses the current directory if no path is specified in <file-name>, but it can take a full path to a file if you wish.

Example

```

stuff
Just some junk. The line below saves this box
as a file.

save-as junk
Data

delete-file JUNK
Data

```

choose-fileSyntax**choose-file** <path>Description

choose-file brings up a dialog to choose a file from disk. It returns the path to that file.

Note: You can directly open a chosen file; see the second example.

Examples

```

choose-file | Medalist:Rafael:BoxerMaterials folder:Command Manual:junk~
Data

open choose-file | stuff
Just some junk. The line below saves this box
as a file.

save-box-as junk
Data

```

directorySyntax**directory** <path>Description

directory takes a directory (or file-specification) as input and returns a list of files in that directory. May be used with URL for network access. Path separator is “:” for MAC.

Example

directory

Macintosh HD:Apps:

Data

Macintosh HD:Apps:Readme

Macintosh HD:Apps:SimpleText

Data

current-directorySyntax**current-directory**Description

current-directory returns your current directory in the form of a path. Path separator is “:” for MAC.

Example

current-directory |

Macintosh HD:Boxer:Command Manual:

Data

set-current-directorySyntax**set-current-directory** <path>Description

set-current-directory changes the current directory for file reading and saving to the one specified in the path. Note that this directory may not exist in your machine. But if you can specify the path to one, you may try it out.

Example

set-current-directory

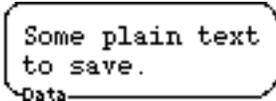
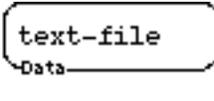
Macintosh HD:Boxer:Command Manual:

Data

save-text-fileSyntax**save-text-file** <box> <file-name>Description

save-text-file saves a Boxer file as plain text, instead of using the Boxer file format. This might be useful if you want to transfer a Boxer file to a text processing program. You can specify a path if you need to in <file-name>. **save-text-file** has a different format. It must be given a box to save as input (rather than saving a superior file box), and it doesn't turn that box into a file box.

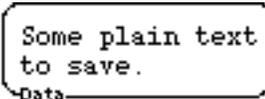
Example

```
save-text-file  
```

read-text-fileSyntax**read-text-file** <box> <file-name>Description

read-text-file reads in a plain text file (rather than one in Boxer format) and places the result in a box. This way, you can import plain text from a text processing program. You can specify a path if you need to in <file-name>.

Example

```
read-text-file  | 
```

CHAPTER 9

Environment: Keystrokes and Editing

This section deals with the Boxer editor. It explains the basic key bindings, Boxer's special characters and how to locate the cursor under program control. Note that keyboards differ, so that some options explained here may not be available with your keyboard.

9.1 MOVING IN BOXER

Most motion is okeys. Of course, there are mouse equivalents to many of these.

Characters

left-arrow	left one character
right-arrow	right one character

Words

command-right-arrow	right one word
command-left-arrow	left one word

Lines

up-arrow	up one line
down-arrow	down one line
option-right-arrow (command-e)	end of line
option left-arrow (command-a)	beginning of line

Box-scroll

PgUp	scroll up one box
PgDn	scroll down one box

Global-box

home (option-up-arrow)	to top of box
end (option-down-arrow)	to bottom of box

Among-Box

command-option-right-arrow	enter next box
command-option-left-arrow	enter previous box
tab	hop to next box

Exit-box

]	exit box
)	exit box and shrink
command-tab	exit, hop into next box
option-tab	exit, hop into previous box

9.2 MAKING BOXES

Regular Boxes

[doit box
{	data box

Graphics

option-t	turtle box
option-s	sprite box
option-g	graphics box

Ports

option-p	port box
command-option-p	set port target (then option-p ports to target)

9.3 CUT AND PASTE

Command-X, command-C and command-V are standard shortcuts for cut, copy and paste. There are mouse equivalents to these actions.

Boxer saves up prior cuts or deletes you make (usually up to 8 of them). "Paste" ordinarily retrieves a copy of the last one. You may paste any number of copies with multiple pastes.

"Yank" (command-y) does not fetch a copy of the last delete, but places the original at the point of the cursor. (In Boxer you can tell the difference between a copy and the original if there are ports to any part of the original. Those ports will not target a copy.)

Pressing Yank several times in succession cycles through saved cuts/copies or deletes. It leaves the items highlighted so you can press "Copy" if you see something you want a copy of, you can press "Delete" if you don't see what you want, or just click the mouse to leave the currently yanked stuff. Yanked stuff, of course, no longer is saved.

Boxer tries to combine several presses of the delete key, and similar actions, into a single cut, which is saved.

Cut & Paste Keys

command-x	cut region (also Delete key)
command-c	copy region
command-v	paste last cut or copy
command-y	yank last item cut, copied or deleted; or if pressed more than once, yank cycles through prior cuts.

Other Cut & Paste Actions

Delete (Backspace)	delete previous character
Ins	delete one character forward
option-delete (option-backspace)	delete one word backward
command-delete (command-backspace)	delete one word forward
command-k	delete complete line
command-option-delete (command-option-backspace)	delete to end of line

9.4 OTHER KEYSTROKES

General Keystrokes

enter (command-Return on keyboards without Enter.)	doit (execute current line)
command-LineFeed (command-Enter)	step execution
command-. (command-g)	stop execution
command-f	find (search)
option-f	reverse find
up-arrow	name this box (only from top line of box)
	name this box
command-@ (may be control-@)	unbox this box (remove box boundary)
command-r	refresh display
command-t	toggle transparency (see Boxer Structures)
command-z	zoom to target port the cursor is in
PrSc (command-p)	print screen
command-' (command-")	quote next character (to print non-printing keys)

Places

command-space	mark this place
option-space	jump to last place (may be used in succession)
command-/	name this place
option-/	jump to named place
option-x	jump to last place and mark the current one

HELP

Help (command-h)	Help
command-Help (command-?)	Prompt inputs (show input names for command near cursor)
option-Help (option-?)	Provides help on key bindings or mouse presses.

Capitalization

option-c	capitalize next word
option-u	upper case next word
option-l	lower case next word

Miscellaneous

option-return	open a line (insert carriage return, but leave cursor here)
command-<number>	numeric repeat for editor keys (Hold the command key and press a number to repeat the editor command that follows)
command-option-v	"vanilla" mode for keybindings and mouse actions (turn off key and mouse redefinitions). Also available in the Other pulldown menu, Top Level (Local) Mouse/Key Mode. (See Input-Output on re-defining keys.)

9.5 SPECIAL-CHARACTERS: KEYS-THAT-PRINT

@
!
.(dot)
^
;
:

These are characters that Boxer uses for special purposes, i.e., they have special meanings. Most are also documented elsewhere.

9.6 CURSOR LOCATION

cursor-column-number
cursor-row-number
move-cursor

These commands tell you where the cursor is placed in a box, and allow you to reposition it. These may be useful with edit-box or other programs that monitor asking a user to edit. See Input-output, Mouse-polling, for related commands.

9.7 BOX-SIZING

expand-box
shrink-box
fullscreen-box
supershrink-box

These commands control the display size of boxes (fullscreen, expanded, shrunk, or supershrunk). They may be used to replace "by hand" (mouse) expanding and shrinking operations. They may be used with triggers to make boxes that "know" what size they want to be displayed at. You cannot control the sizes of ports this way, only their targets.

9.5 KEYS THAT PRINT

@

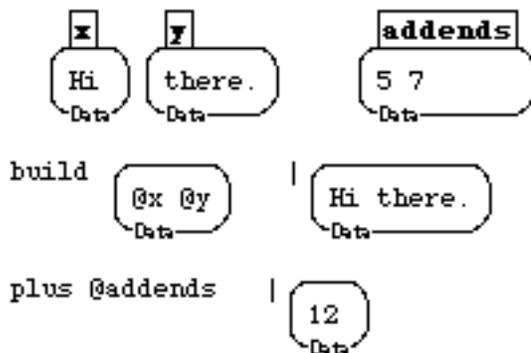
Syntax

@

Description

@ is used inside a **build** template to mean "evaluate and unbox." See Data-manipulation Section, **build**. @ also may be used outside a **build** template. In this case it means "evaluate this part of a command line first, then unbox in place, and finish by executing the line as modified." See Boxer Structures and Control-structure, Evaluation.

Example



!

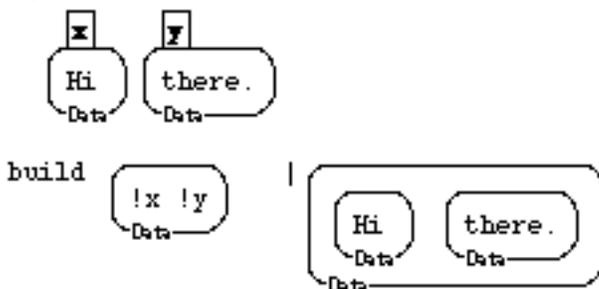
Syntax

!

Description

! is used in a **build** template to mean "evaluate" (and leave boxed) this expression. See **build** in Data-manipulation.

Example



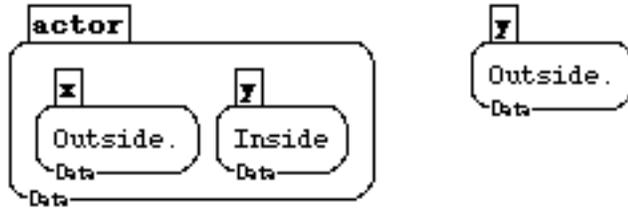
^

Syntax

^

Description

^ is used with **tell** (**ask**) to cause a name refer to a box accessible from the place **tell** is executed. Ordinarily, any name in the message for **tell** will refer to the boxes accessible in the object told. See Evaluation in Control-structure, and also Boxer Structures.

Example

```
tell actor change x ^y
```

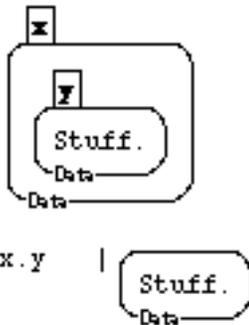
. (dot)

Syntax

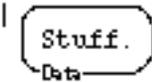
<name1>.<name2>

Description

dot is used to "chain" box names into a path. x.y.z refers to the z box inside y, inside x. See Data-manipulation, Data-access by name.

Example

x.y



;

Syntax

;

Description

; is a comment character. Boxer ignores anything following it on a line when executing

Example

```

info
I will be executed.
Data

```

```

change info
I will be executed.
Data
; change info
I will not be executed.
Data
Data

```

:

Syntax

:

Description

Words that end with : are not executed They are "in-line comments".

Example

```

try: Plus 34 89 | 123
Data

```

9.6 CURSOR LOCATION

cursor-column-number

Syntax

cursor-column-number

Description

Returns the character number at which the cursor is located in a box (i.e., which character in its current row it is placed before.) It works in name tabs, too.

Example

```
cursor-column-number | (10)
                    Data
```

cursor-row-number

Syntax

cursor-row-number

Description

Returns the row number at which the cursor is located in a box.

Example

```
Row 1
Row 2
cursor-row-number | (3)
                  Data
Row 4
```

move-cursor

Syntax

move-cursor <box> <row> <character>

Description

Moves the cursor, and animates its motion, to the box, row, and character number specified. Note: You can only move to boxes, not ports. Move-cursor moves you to the target of any port passed as first input.

Example

```

one | two
line one | line one
line two | line two
Data | Data

move-cursor two 1 6
```

9.7 BOX-SIZING

expand-box

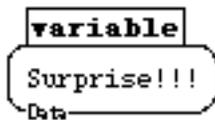
Syntax

expand-box <box>

Description

This command is used to set a box to normal expanded size under program control, instead of having to do it "by hand" using the mouse. It will actually shrink a fullscreen box to merely expanded size. **expand-box** will expand the target of a port passed to it as input.

Example



`expand-box variable`

shrink-box

Syntax

shrink-box <box>

Description

This command is used to set a box to normal shrunk size under program control, instead of having to do it "by hand" using the mouse. **shrink-box** will actually "expand" a supershrunk box to merely "shrunk" size. Expand-box, shrink-box, etc., will operate on the target of a port passed to it as input.

Example



`shrink-box variable`

CHAPTER 10

Environment: Networking

In order to access remote boxes on the Internet, you need to have a direct or modem (SLIP, PPP) connection to it. (Technically, you need TCP/IP services.) In order to host others by making some of your local boxes available, you need to configure your machine to be an FTP server. If you do not have or use such services, we suggest you consult your local network guru.

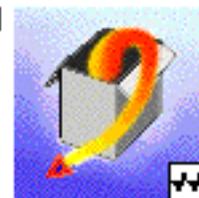
Boxer has a special kind of box, a "net box", to maintain connection to resources over the network, or to browse the known Boxer network universe. Net boxes appear like file boxes, but instead of remembering the file associated with them, they remember the network location (URL) of the information in that box. Like file boxes, net boxes start shrunk and empty when you read them in within a file, but they fetch their internals whenever you click on them or in any other way try to use them. Once read in, they behave like ordinary data boxes.

Net boxes can link to Boxer or text files, or to directories. The contents of directories (which consists of files or more directories) appear as named net boxes.

You can convert net boxes to regular boxes, change them to file boxes, inspect or change their associated URL using the LINKS selection from the Box Properties panel (Box Properties is in the Other menu--or use the box type hotspot popup menu). You can also change a regular box to a net box and assign a URL to save it to in the same way. File/Save will then save it. The OPEN menu command (see documentation below) can also read in and thus create a new net box. We recommend that you attach the suffix .box to Boxer files made available over the network.

To get to the "hub of the Boxer universe" the following command takes you there -- the box on the right is the actual hub box.

```
open ftp://soe.berkeley.edu/pub/boxer/hub.box
```



Remember that the address of the net box must be saved with its superior file (or superior net box) in order to be available from that file (net box) when it is read in later. Hence, if you change a URL, always save the superior file (net box).

See also general information on file/net boxes in the Files section of the manual.

Boxer Site Suggestions

Setting up your own Boxer site:

After you have arranged to have a site FTP accessible, here are some hints: Use the suffix `.box` on Boxer files you make network accessible. It will help force the proper type of file transfer. Transfer files from your Mac to the server using "raw" protocol, i.e., *not* MacBinary or other special codings. FTP won't recognize these. (Or you can use Box Properties (Other menu) to link a local box to its server URL, then use **save**.)

In making boxes for others to link to, make sure you have a "browsable layer": that is, a set of small boxes that do not take long to read in but give a person browsing enough information -- like file size and a brief description -- so that s/he can make an intelligent decision about whether it is worth waiting for the net box to read in. Leave information on what kind of feedback, if any, you are interested in and appropriate e-mail addresses. Keep binary and other non-Boxer useful files out of any directories that you make accessible.

Linking to the Hub of the (Boxer) Universe

As long as it is feasible, we will try to coordinate interesting boxes to browse from our hub. Send URLs and information on how you imagine your box should be categorized to: `boxer-inquiry@soe.berkeley.edu`. We solicit especially: (1) descriptions of Boxer projects around the world, (2) solicitations for Boxer subcommunities and network projects, (3) general tools and utilities you feel people may like to have, (4) ideas on excellent things to do with Boxer, including "great hacks," (5) materials for learning various subject matter with Boxer, (6) examples of especially interesting student work.

10.1 NETBOX FILE/DIRECTORY COMMANDS

open
save
save-as
save-box-as
delete-file

These are programming commands to be use with net boxes. They work essentially identically with their use with files.

10.2 MAIL COMMANDS

mail
getmail

These are commands that allow a Boxer user to send and receive mail over the network. Before using mail, you must initialize the user-mail-address and the mail-relay-host in your Boxer Preferences (Other pulldown menu). We recommend getting a Boxer mailer (e.g., in the Boxer release demo files) to ease mail use.

10.1 NET BOX FILE/DIRECTORY COMMANDS

open

Syntax

open <URL>

Description

Open, in addition to reading in ordinary files, will also read in files and directories over the network, creating "net boxes". See the overview, above, and Boxer Structures document for an explanation of net boxes. **Open** may actually be more convenient in most cases compared to using the File menu. The format for the input (Universal Resource Locator--URL) is as follows: (Place the URL in a data box.)

<protocol>://<login-name>:<password>@<internet-machine-address>/<file path>/<file>

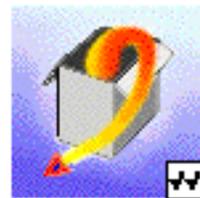
- <protocol> currently is "ftp", probably the most general protocol (using TCP/IP resources). Other protocols may be added.
- <login-name>:<password>@ is optional. The default is "anonymous". This makes connecting to an anonymous FTP site simple.
- /<file-path> optional.
- /<file> is optional. Without it, a directory will be read.

FTP standards specify *text* as the default type of files read. If you save a Boxer file, **open** will recognize that it is NOT text if you put a .box suffix on the file name. Files should be in raw binary (not Mac binary, or other format) on Unix machines.

Examples

```
open ftp://login-name:password@machine-name.site.edu/directory/path/file
Data
```

```
open ftp://soe.berkeley.edu/pub/boxer/hub.box
Data
```



open ftp://soe.berkeley.edu
Data

|

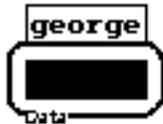
.logout
Data
.profile
Data
HRC Test
Data
bin
Data
etc
Data
fred.box
Data
prorder.wri
Data
pub
Data
Data

saveSyntax**save**Description

save saves the first superior net or file box to its corresponding file. It may be handy to use it with **tell**, as in **tell <box> save**. **save**, in addition to reading in ordinary files, will save files over the network. It may be handy to use it with TELL as in **tell <box> save**.

Examples

These examples are not guaranteed to work because of changing machine and directory access privileges.



```
tell george save
```

save-asSyntax**save-as** <URL>Description

Save-as <URL>, in addition to saving ordinary files, will save files over the network. It saves the first superior net (file) box from where it is executed. The format for the input (Universal Resource Locator--URL) is as follows: (Place the URL in a data box.)
 <protocol>://<login-name>:<password>@<internet-machine-address>/<file path>/<file>

- <protocol> currently is "ftp", probably the most general protocol (using TCP/IP resources). Other protocols may be added at a later date.
- <login-name>:<password>@ is optional. The default is "anonymous". This makes connecting to an anonymous FTP site simple.
- /<file-path> optional.

Examples

```
save-as ftp://machine.site.edu/a-file.box
Data
```

```
tell <a-box> save-as ftp://machine.site.edu/a-file.box
Data
```

```
save-as ftp://login-name:MyPaSsWoRd@machine-name.site.edu/a-file.box
Data
```

```
save-as ftp://login-name:password@machine-name.site.edu/directory/path/file
Data
```

save-box-asSyntax**save-box-as** <URL>Description

Save-box-as <URL>, in addition to saving ordinary files, will save files over the network. It saves the box where it is executed.

The format for the input (Universal Resource Locator--URL) is as follows: (Place the URL in a data box.)

```
<protocol>://<login-name>:<password>@<internet-machine-address>/<file path>/<file>
```

- <protocol> currently is "ftp", probably the most general protocol (using TCP/IP resources). Other protocols may be added at a later date.
- <login-name>:<password>@ is optional. The default is "anonymous". This makes connecting to an anonymous FTP site simple.
- /<file-path> optional.

Example

```
save-box-as ftp://machine.site.edu/a-file.box
Data

tell <a-box> save-box-as ftp://machine.site.edu/a-file.box
Data

save-box-as ftp://login-name:MyPaSsWoRd@machine-name.site.edu/a-file.box
Data

save-box-as ftp://login-name:password@machine-name.site.edu/directory/path/fil
Data
```


get-mailSyntax

get-mail <mailbox> <delete-messages?>

Description

get-mail <mailbox> <deleted?> fetches mail from your mail host machine. It returns a box containing a series of messages, each in a box. The second input is true or false and specifies whether the messages should also be deleted from host machine mailbox. For safety, consider using FALSE as second input. Enclosed files will appear as "links to Mac

files", which you can double-click to open. The closet of each message contains several variables that can be used in sorting or other processing of messages: To, From, Subject (optional), Date, Header. **get-mail** uses the stadard POP protocol, like such mail readers as Eudora. Currently (9-99), **get-mail** recognizes base64 and binhex mime protocols. Check the boxer hub or newer demos for a full-featured mail reader that uses get-mail.

Examples

```
get-mail  
```

Note: You will probably be prompted for a password. False specifies messages will NOT be deleted from host mailer.

```
get-mail  
```

Note: "True" specifies messages will be deleted from host after receipt.

```
get-mail  
```

```
From: nose@clowncollege.edu
To: bozo@clowncollege.edu
Date: Sun, 7 Dec 97 12:37:26
```

```
:-)
```

```
Date: Sun, 7 Dec 97
From: bigfeet (Blue nose)
To: bozo
Subject: testing, testing
```

```
I got the joke
```

```
--Bigfeet
```

sendSyntax**send** <address> <text>Description

Sends a box to networked colleagues for communication or real-time interaction. **send** is not currently (9-99) configured to work on non-Unix machine.

Examples**No examples YET**

CHAPTER 11

Environment: Miscellaneous

The commands in this chapter have to do with various aspects of the Boxer environment, such as setting preferences and handling errors when they cannot be returned to the screen for various reasons. Other facilities are for help, defining icons, saving a user's interaction ("dribble"), Boxer extensions and serial port connections. Use Edit menu, Preferences selection as a "by-hand" alternative.

11.1 SYSTEM-PREFERENCES

system-preferences

You may change settings by editing and executing the lines in the system preference box that have preferences commands in them. This command appears in the closet of the top level box of your world. A Mac interface to preferences is available in the Edit menu, Preferences... option.

11.2 RESULT APPEARANCE

printing-precision
print-fractions
preserve-empty-lines-in-build

These commands affect how various sorts of returned values appear: i.e., the number of decimal places that are printed in a number, whether rational numbers should appear as decimals or not, and whether **build** should preserve or delete empty lines. See *Arithmetic-and-logic, under Printing-control*.

11.3 EVALUATOR SETTINGS

step-wait-for-key-press
step-time
evaluator-help
primitive-shadow-warnings

These control settings on various forms of help Boxer can supply, or adjustments to the way Boxer runs.

11.4 GRAPHICS SETTINGS

make-transparent-graphics-boxes
include-sprite-in-new-graphics
name-new-sprites
make-diet-sprites

These adjust details about what happens when you create a new turtle, graphics or sprite box.

11.5 EDITOR SETTINGS

zoom-pause
show-border-type-labels
smooth-scrolling
global-hotspot-controls
input-device-names
fullscreen-window

These control some options in the way the Boxer editor works. Most inputs are true or false; some are numbers.

11.6 FILE SYSTEM SETTINGS

terse-file-status
backup-file-suffix
warn-about-outlink-ports

Sets parameters relevant to files.

11.7 NETWORK SETTINGS

user-mail-address
mail-relay-host
max-viewable-message-size

Sets parameters relevant to network operations.

11.8 COMMUNICATIONS SETTINGS

newline-after-serial-writes
serial-read-base

These preferences determine how Boxer reads and writes to (from) external devices over the serial line.

11.9 MISCELLANEOUS SYSTEM COMMANDS

name-help	This command gives some brief information about all primitive commands in Boxer that contain the sequence of letters in the input.
invisible-error	If Boxer cannot find a place to print an error message, it notifies you. Then, invisible-error , when executed, will return the error. A typical place where you get such errors is from interface messages, like <code>sprite-</code> or <code>graphics-mouse-</code> commands.
invisible-value	If Boxer cannot find a place to print a returned value it notifies you. Then, invisible-value , when executed, will return the value. A typical place where you get such errors is from clicks in nametabs that execute something that returns a value. (You can't put a box in a name-tab.).
date-and-time	This command returns the data and time according to the systems clock
boxtop	If Boxer finds a graphics box called <code>boxtop</code> in a box, the graphics in that box become the box's shrunken shape, in place of the usual small gray box. The size of the graphics box will be the size of the <code>boxtop</code> . The best place for a <code>boxtop</code> box is usually in the box's closet. Sprites don't show up in the <code>boxtop</code> .
dribble-on dribble-off playback-dribble-file	Boxer can create and run "dribble files", which are records in a file of every mouse click and keystroke done by a user. Then those files may be played back at another time. These are useful to take data from subjects doing an experiment in Boxer, or for storing an active demonstration, or for automating a test procedure, etc.
mark-for-saving	Mark-for-saving forces the File pulldown menu to show that saving is allowed.
choose-file	This command pops up a Mac dialog box to allow you to select a file. Then, choose-file returns the complete file path to that file. Typical use is with <code>read</code> . That is, you ask for a file to read: <code>read choose-file</code> .

11.10 EXTENSIONS

extension-info load-extension add-extension remove-extension	Boxer extensions are files that add extra features to Boxer. Boxer automatically loads any extensions placed in a folder called "Extensions" when it starts up. (The Extensions folder must be placed in the same folder as the Boxer application.) Extensions may be placed in a folder "Extensions(off)" when you don't want them loaded at startup. Extension-info allows you to see (1) what extensions are available, (2) which ones are loaded automatically at startup, and (3) which ones are currently loaded. In addition, commands allow you to move extensions between Extensions and Extensions(off).
---	---

11.11 SERIAL PORT: SETUP *NOTE: Serial Port commands are minimally documented*

open-serial-line close-serial-line configure-serial-line set-default-line-parameters serial-line-parameters	These are commands to configure and open a serial port for communications with external devices (such as computer-controlled video machines, or science laboratory sensing devices). They provide many relatively standard ways of reading from and writing to a serial line. These are designed for "hackers" so our documentation is very brief.
--	--

11.12 SERIAL PORT: READING

serial-listen
serial-read-line
serial-read-line-with-timeout
serial-read-line-no-hang
serial-read-char
serial-read-char-with-timeout
serial-read-char-no-hang
serial-read-byte
11.13 SERIAL PORT: WRITING

serial-write
serial-write-byte
11.4 SERIAL PORT: PREFERENCES

serial-read-base
newline-after-serial-writes

11.1 SYSTEM-PREFERENCES

system-preferences

Syntax

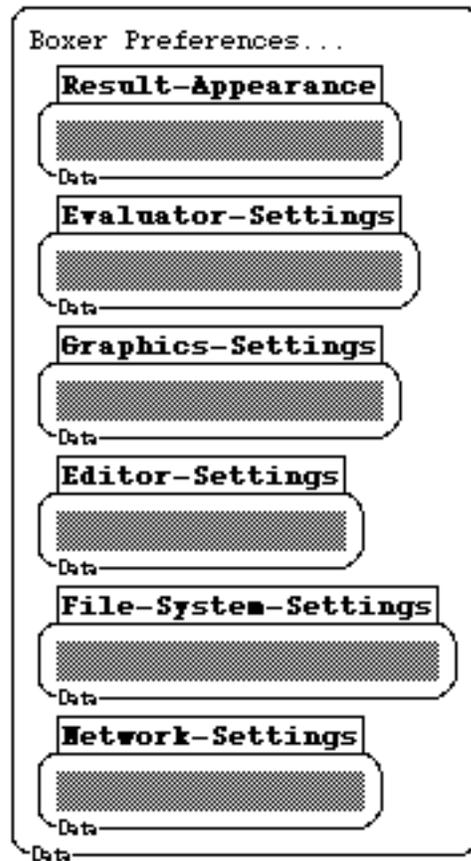
system-preferences

Description

System-preferences causes a box to appear that has all Boxer's adjustable system parameters in it. You may change settings by editing and executing the lines in the system preferences box that have preferences commands in them. This command appears in the closet of the top level box of your world. Alternatively, use Edit menu, Preferences selection.

Examples

`system-preferences`



11.2 RESULT APPEARANCE

preserve-empty-lines-in-build

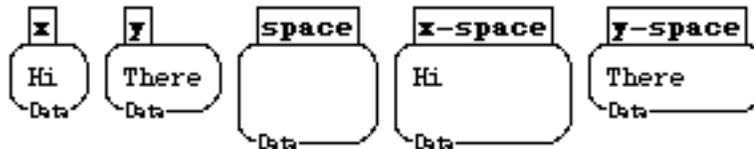
Syntax

preserve-empty-lines-in-build <true-or-false>

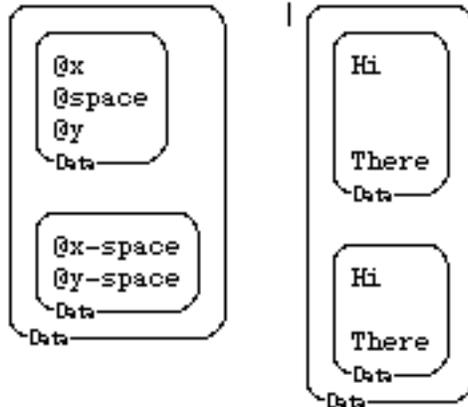
Description

This tells Boxer whether or not to preserve empty lines in **builds**. These refer only to empty lines in evaluated parts of **build** (@ or ! parts). Empty lines typed into the **build** template directly are always preserved.

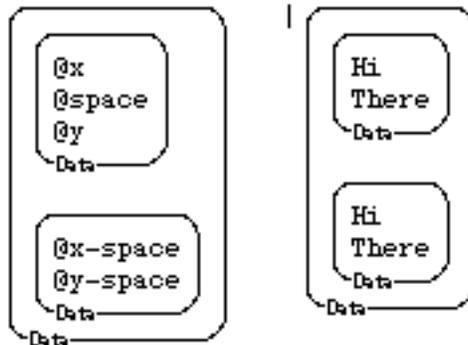
Example



```
preserve-empty-lines-in-build true
build
```



```
preserve-empty-lines-in-build false
build
```



11. 3 EVALUATOR-SETTINGS

step-wait-for-key-press

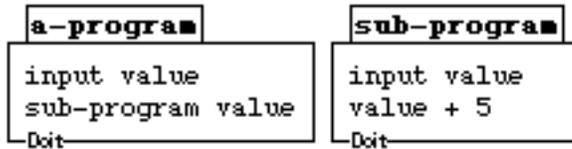
Syntax

step-wait-for-key-press <true-or-false>

Description

This determines whether Boxer waits for a keypress when showing the "stepper" execution before going from step to the next step. (As of 9-99, the stepper is disabled.)

Examples



Place your cursor at the end of the Try&Look and press the STEP key (ctrl-linefeed on Sun, command-enter on Macintosh -- or see interface and keystroke documentation.). Do this with step-... turned on and off.

Try: step-wait-for-key-press true

Try: step-wait-for-key-press false

Try&Look: a-program 3
Doit

Note

The stepper shows Boxer's copy and execute model in action. See documentation of Boxer Structures.

Doit

step-timeSyntax**step-time** <seconds>Description

Step-time <seconds> determines how long the Boxer stepper waits before moving to the next step. It is in effect only when **step-wait-for-key-press** is set to false. (As of 9-99, the stepper is disabled.)

Examples

a-program	sub-program
input value sub-program value	input value value + 5
Exit	Exit

Place your cursor at the end of the Try&Look and press the STEP key (ctrl-linefeed on Sun, command-enter on Macintosh -- or see interface and keystroke documentation.). Adjust the step-time in Setup: , and try again.

Setup:

step-wait-for-key-press false step-time 1
Exit

Try&Look: a-program

3
Exit

Note

The stepper shows Boxer's copy and execute model in action. See documentation of Boxer Structures.

Exit

evaluator-help

Syntax

evaluator-help <True or False>

Description

Evaluator-help determines whether Boxer supplies "helpful" messages when it detects stylistic oddities. For example, making a port to the result of a Boxer primitive is an odd thing to do. **Evaluator-help** set to false also suppresses some other information that Boxer otherwise prints out about its internal state (e.g., when it is expanding internal space reserved for various activities).

Examples

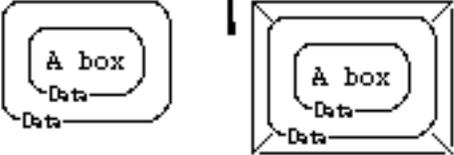
```

+ File Box: env-miscellaneous • |
Made a PORT to a copy of the result of a primitive.
examples
Try the below port-to command with evaluator
help set to true and then false.

Try: evaluator-help true

Try: port-to first

```



primitive-shadow-warning

Syntax

primitive-shadow-warning <True or False>

Description

Primitive-shadow-warning controls whether Boxer should print out a warning when you create a box that has the same name as a primitive, and will "shadow" the primitive so that it cannot be used in the current or inferior boxes. See Boxer Structures documentation concerning shadowing.

Examples

The screenshot shows a window titled "File Box: env-miscellaneous • From: env-miscellan". At the top, a warning message reads: "Warning: The Primitive FORWARD will no longer be available in this box". Below the warning, there is a section titled "examples" containing the following text:

```
Setup: PRIMITIVE-SHADOW-WARNINGS true
```

Edit the following procedure so that it has the same name as
The primitive FORWARD. With shadow-warnings set to true,
Boxer will print a warning at the top of its window.

```
forward |
  Data
```

11.4 GRAPHICS SETTINGS

make-transparent-graphics-boxes

Syntax

make-transparent-graphics-boxes <true or false>

Description

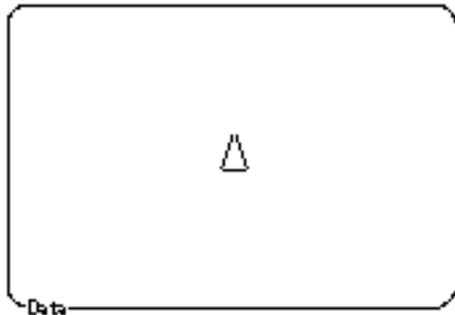
This command controls whether Boxer makes transparent graphics boxes, or non-transparent ones. Graphics boxes that are not transparent will not "export" the names of sprites (or anything else) that are inside them. If you want to address sprites inside, you need to talk first to the graphics box (usually first giving it a name). Make a graphics box below with **make-transparent-...** set to **true**, then **false**.

Examples

```
make-transparent-graphics-boxes true
```



```
make-transparent-graphics-boxes false
```



include-sprite-in-new-graphics

Syntax

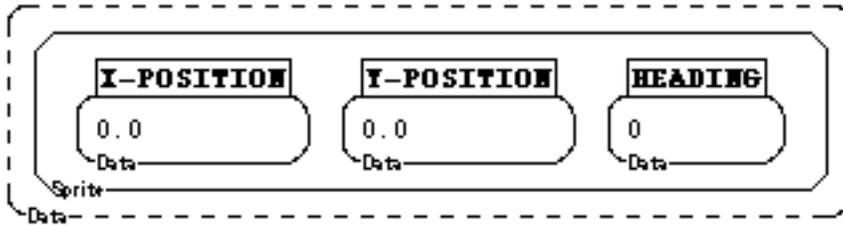
include-sprite-in-new-graphics < true or false >

Description

This command determines whether Boxer includes a sprite when you create a new graphics box.

Examples

```
include-sprite-in-new-graphics true
```



```
include-sprite-in-new-graphics false
```



name-new-sprites

Syntax

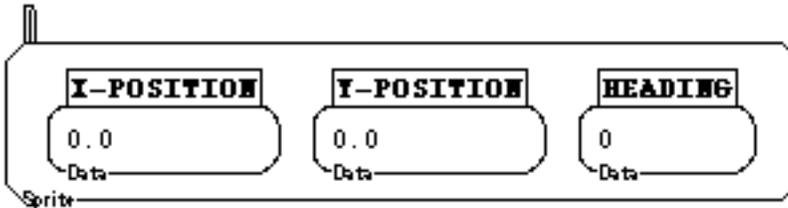
name-new-sprites <true-or-false> >

Description

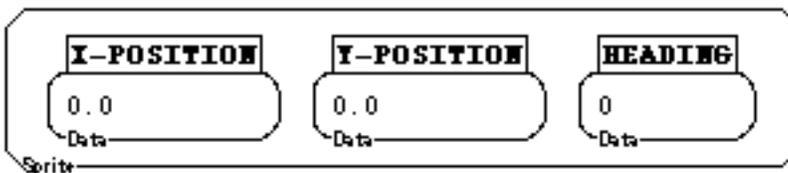
Name-new-sprites determines whether you are automatically left in the name-tab of a sprite when you make one.

Example

```
name-new-sprites true
```



```
name-new-sprites false
```



make-diet-sprites

Syntax

make-diet-sprites < true or false >

Description

This command determines whether sprites are created in their diet configuration (where most sprite attributes are not shown unless needed), or with all attributes included (some in the closet).

Examples

```
make-diet-sprite true
```

Show-Sprite-Properties

X-POSITION	Y-POSITION	HEADING
0.0	0.0	0
Data	Data	Data

Sprite

```
make-diet-sprite false
```

SHAPE	SHOWN?	HOME-POSITION	SPRITE-SIZE
△	TRUE	0.0 0.0	1.0
Data	Data	Data	Data
PEN	PEN-WIDTH	TYPE-FONT	PEN-COLOR
DOWN	1	1442058	0.0 0.0 0.0
Data	Data	Data	Data
X-POSITION	Y-POSITION	HEADING	
0.0	0.0	0	
Data	Data	Data	

Sprite

11.5 EDITOR SETTINGS

zoom-pause

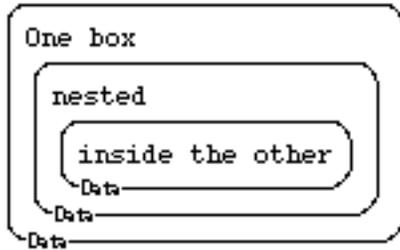
Syntax

zoom-pause <seconds>

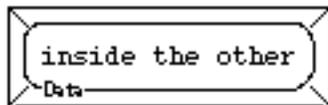
Description

This command controls the wait time between steps of the animation during zooming.

Examples



Place your cursor inside the following port and ZOOM (press cntrl-z, or command-z).



Now change the pause time and try it again (normal is .05).

zoom-pause .1

zoom-pause 0

show-border-type-labels

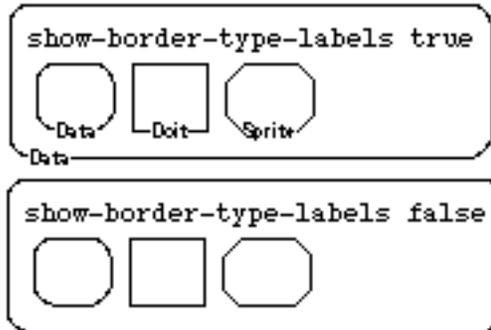
Syntax

show-border-type-labels <true or false>

Description

This command causes type labels to appear or disappear from the bottom line of boxes.

Examples



smooth-scrolling

Syntax

smooth-scrolling <true or false>

Description

When executed with true, scrolls a pixel at a time. When false, scrolls a line at a time.

Examples

```
smooth-scrolling true
```

```
smooth-scrolling false
```

global-hotspot-controls

Syntax

global-hotspot-controls <true or false>

Description

This command causes turning off or on a hotspot (like graphics flip--lower left) to affect all boxes. Otherwise, turning on or off hotspot sensitivity affects only the box changed.

Examples



```
global-hotspot-controls true
```

```
global-hotspot-controls false
```

input-device-names

Syntax

input-device-names <SUN-TYPE-4 or Mac>

Description

Determines the set of names used for keyboard keys, shifts, and mouse clicks interface messages. This command will allow you to run programs written for the Mac or Sun on the other platform without changing all the messages.

DON'T KNOW IF THESE ARE CORRECT

Mac:

Sun:

sprite mouse click

sprite-mouse-middle

mouse-double-click-on-graphics

graphics-mouse-middle-twice

mouse-double-click-on-name

name-mouse-middle

command-mouse-click

mouse-right

option-mouse-click

mouse-left

Examples

```
input-device-names SUN-TYPE-4
```

```
input-device-names Mac
```

Try both of the above then perform some action that results in an interface message. E.g, click on a graphics box or sprite.

fullscreen-windowSyntax**fullscreen-window** <true or false>**Description**

Determines whether Boxer starts up with its window filling the full screen, or a smaller (more typical) size. The former is better if someone is using only Boxer; the latter is better if you are moving back and forth among applications.

Examples

```
fullscreen-window true
```

11.6 FILE SYSTEM SETTINGS

terse-file-status

Syntax

terse-file-status <true or false>

Description

With input true, abbreviates information in the Boxer window title bar.

Examples

```

┌ File Box: env-miscellaneous • From: (disk) env-miscellaneous
└ Purduegold Boxer
  terse-file-status true

```

```

┌ File Box: env-miscellaneous • From: env-miscellaneous {Medalist:R...
└ Purduegold Boxer
  terse-file-status false

```

backup-file-suffix

Syntax

backup-file-suffix <symbol>

Description

Its input determines the suffix that marks backup files. The default symbol is ~.

Examples

```

backup-file-suffix ~

```



warn-about-outlink-ports

Syntax

Warn-about-outlink-ports <true or false>

Description

Should Boxer provide a warning when saving a file that has ports to boxes outside that file? (Such links will NOT be preserved through the file-saving file-reading process.).

Examples

```

warn-about-outlink-ports true

```

11.7 NETWORK SETTINGS

user-mail-address

Syntax

user-mail-address <address>

Description

Sets the "return address" for network operations like sending mail.

Examples

```
user-mail-address nobody@soe.berkeley.edu  
Data
```

mail-relay-host

Syntax

mail-relay-host <machine>

Description

Sets the name of the computer responsible for relaying mail to the Internet.

Examples

```
mail-relay-host dewey.soe.berkeley.edu  
Data
```

max-viewable-message-size

Syntax

max-viewable-message-size<size-in-bytes>

Description

Sets the maximum size of a message that will be read directly into Boxer with read-mail. larger messages will be read to a file..

Examples

```
max-viewable-message-size 64000
```

11.8 COMMUNICATIONS SETTINGS

newline-after-serial-writes

Syntax

newline-after-serial-writes <true-or-false>

Description

This command determines if a *carriage return* should be added at the end of each Serial-Write. This command is only useful if you are communicating with an external device through the serial port on your computer.

Examples

```
newline-after-serial-writes true
```

```
newline-after-serial-writes false
```

serial-read-base

Syntax

serial-read-base <interger>

Description

This command sets the radix that will be used when the serial line reads in numbers.

Examples

```
serial-read-base 10
```

```
serial-read-base 8
```

11.9 MISCELLANEOUS SYSTEM COMMANDS

name-help

Syntax

name-help <word-or-part-of-word>

Description

name-help <word-or-part-of-word> gives some brief information about all primitive commands in Boxer that contain the sequence of letters in the input.

Examples

name-help clear|

```
CLEAR-GRAPHICS is a primitive with no arguments
CLEARSCREEN is a primitive with no arguments
CLEARGRAPHICS is a primitive with no arguments
CLEAR-BACKGROUND is a primitive with no arguments
CLEARBACKGROUND is a primitive with no arguments
```

Da

invisible-error

Syntax

invisible-error

Description

If Boxer cannot find a place to print an error message, it notifies you. Then, **invisible-error**, when executed, will return the error. A typical place where you get such errors is from interface messages, like `sprite-` or `graphics-mouse-` commands.

Examples

File Box: env-miscellaneous • From: (disk)
Graphics Mouse Click Can't print error box, Invisible-Error returns it

examples



Click on the sprite or graphics box above, note the information line at the top of your Boxer window. Then try "invisible-error".

Look:

mouse-click-on-graphics

undefined-procedure

Data

mouse-click-on-sprite

undefined-procedure

Data

Try: invisible-error

Error: Can't find a box named
 UNDEFINED-PROCEDURE
 in line UNDEFINED-PROCEDURE
 of box

Data

invisible-value

Syntax

invisible-value

Description

If Boxer cannot find a place to print a returned value it notifies you. Then **invisible-value**, when executed, will return the value. A typical place where you get such errors is from clicks in nametabs that execute something that returns a value. (You can't put a box in a name-tab.)

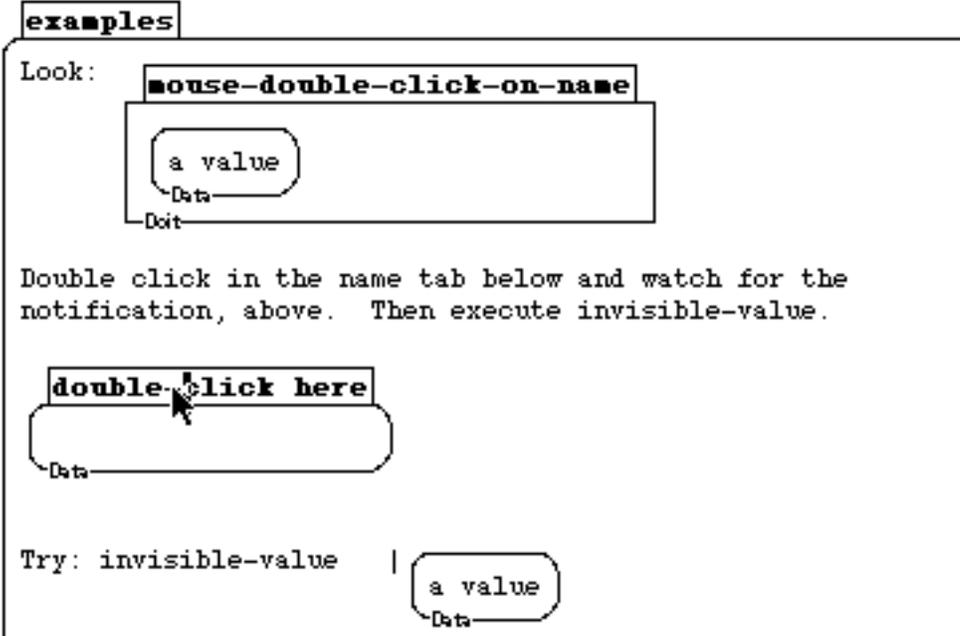
Examples

↑ File Box: env-miscellaneous

Can't print returned value, Invisible-Value returns it

examples

Look: `mouse-double-click-on-name`



Double click in the name tab below and watch for the notification, above. Then execute invisible-value.

Try: `invisible-value` | `a value`

date-and-time

Syntax

date-and-time

Description

Date-and-time returns the date and time according to the system clock.

Examples

`date-and-time` | `Monday May 17 1999 16:55:14 +0000`

`items 1 4 date-and-time` | `Monday May 17 1999`

boxtop

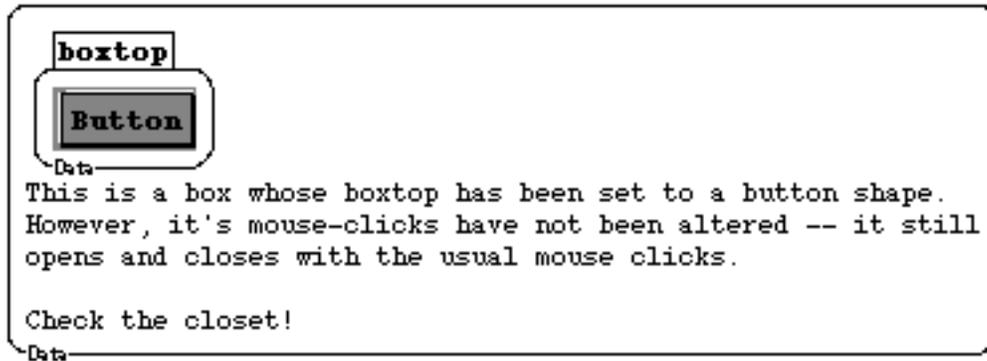
Syntax

boxtop

Description

If Boxer finds a graphics box called **boxtop** in a box, the graphics in that box become the box's shrunken shape, in place of the usual small gray box. The size of the graphics box will be the size of the **boxtop**. The best place for a **boxtop** box is usually in the box's closet. Sprites don't show up in the **boxtop**.

Example



dribble-on

Syntax

dribble-on <file>

Description

Dribble-on <file> causes each mouse click and keystroke to be stored in a file called <file>. As usual, the file name should be put in a box. **dribble-off** halts saving and **playback-dribble-file** <file> replays it. Note: For playback, in order to have click positions and typing placement appear properly, you must have the screen showing exactly what it was when **dribble-on** was executed. A good convention is to start with a blank screen or some fixed file, then start dribble or replay with key bindings (e.g., option-command-s-key (Start) bound to **dribble-on**, option-command-r-key (Replay) bound to **playback-dribble-file**).

Example

Expand this box to full screen. Execute the line below, then perform a few edits in this box, including executing each line in the menu. Then execute the "dribble-off" command, "reset" and "playback-dribble-file". (You will have to locate the dribble-test file, by default at the top level of your hard disk.)

Try: dribble-on

```
dribble-test
Data
```

```

type a little here
la vida loca
Data

menu
2 + 2 | 4
      Data
asfd
Data
```

Stop-dribble: dribble-off

Reset:

```
change examples start-screen
Data
```

Playback:

```
playback-dribble-file choose-file
Data
```

dribble-offSyntax**dribble-off**Description**dribble-off** halts saving of a dribble file. See also **dribble-on**.Example

See dribble-on example

playback-dribble-fileSyntax**playback-dribble-file** <file>DescriptionThis command causes the dribble file called <file> to be run, thus executing all the keystrokes and mouse clicks stored there. See **dribble-on** for another example and other details. <file> must be in a data box. Note: See **dribble-on**.Example

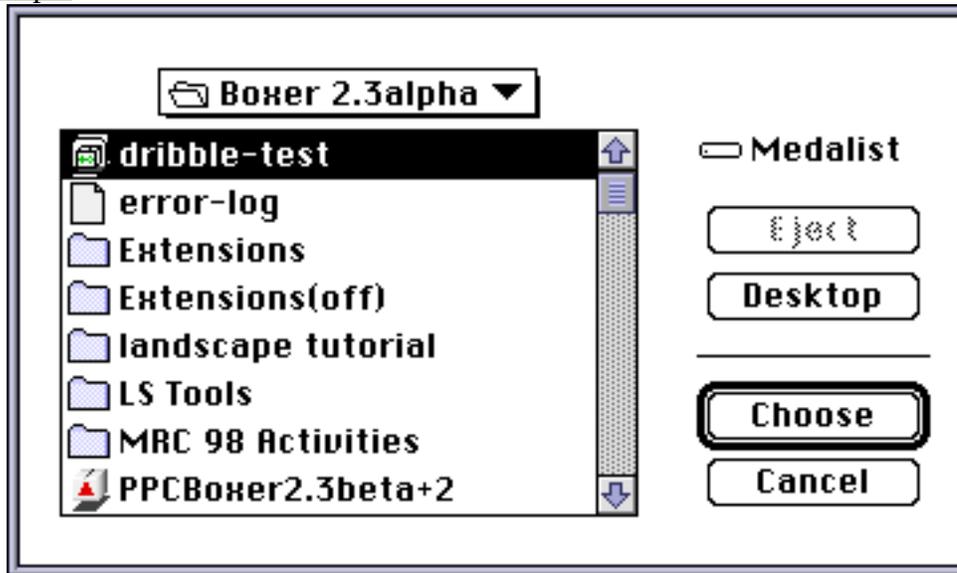
See dribble-on example

mark-for-savingSyntax**mark-for-saving**DescriptionMark-for-saving forces the file pulldown menu to show that saving is allowed. Boxer notes when a box has been changed by the editor, and it then shows this status with an up-arrow in the Boxer window title bar and by the fact that SAVE is an available option in the FILE pulldown. However, Boxer does NOT note if a file box has been changed by the action of a command. **mark-for-saving** lets you compensate in a program if you know the file has been changed or you want saving to be available for some other reasonExample

mark-for-savings

choose-fileSyntax**choose-file**Description

This command pops up a Mac dialog box to allow you to select a file. Then, **choose-file** returns the complete file path to that file. Typical use is with read. That is, you ask for a file to read: read choose-file.

Example

```
choose-file | Macintosh HD:Boxer2.3alpha:dribble-test
Data
```

11.10 EXTENSIONS

extension-info

Syntax

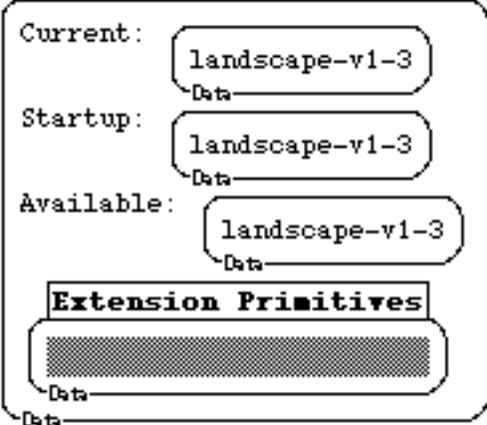
extension-info

Description

This command provides information on which extensions are loaded, which are automatically loaded at startup, and which are available for loading. It also lists commands to load extensions and move them between Extensions folder (load on startup) and Extensions(off) folder (not automatically loaded).

Example

```
extension-info |
```



load-extension

Syntax

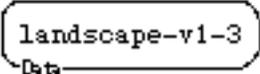
load-extension <box containing an extension name>

Description

Loads into Boxer facilities from the Boxer extension file name in its input. It will almost always load from Extensions(off) folder.

Example

```
load-extension
```



add-extension

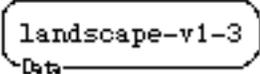
Syntax

add-extension <box containing an extension name>

Description

This command moves a Boxer extension from the folder Extensions(off) to the folder Extensions. (Both folders must be in the same folder as Boxer.) This causes the extension to be loaded automatically the next time you start Boxer. See **load-extension** to load an extension without restarting Boxer.

Example

```
add-extension  landscape-v1-3
```

remove-extension

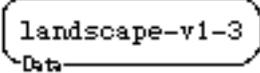
Syntax

remove-extension <box containing an extension name>

Description

This command moves a Boxer extension from the folder Extensions to the folder Extensions(off). Both these folders must be in the same folder as Boxer. This means the extension will *not* be loaded automatically the next time you start Boxer. **Remove-extension** does not remove the extension resources from the running Boxer.

Example

```
remove-extension  landscape-v1-3
```

11.11 SERIAL PORT: SETUP

open-serial-line

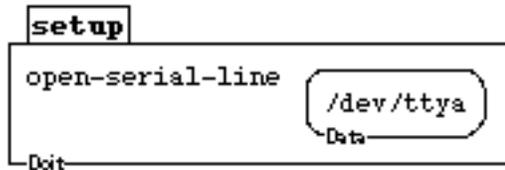
Syntax

open-serial-line <line>

Description

Opens a serial line per input spec.

Example



setup

close-serial-line

Syntax

close-serial-line<line>

Description

Closes a serial line.

configure-serial-line

Syntax

configure-serial-line <baud> <charsize> <stopbits> <parity>
<echo> <canonical> <flowcontrol>

Description

Sets up the parameters of the serial line. See serial-line-parameters and set-default-line-parameters. A typical setup is:

Speed: 1200

Character Size: 8

Stop Bits: 1

Parity: 0 (None)

No Echo

Canonical Processing Off

Flow Control Disabled

set-default-line-parameters

Syntax

set-default-line-parameters <box containing an extension name>

Description

This sets parameters to:

Speed: 9600

Character Size: 7

Stop Bits: 1

Parity: 2 (Even)

Echo On

Canonical Processing On

Flow Control Enabled

serial-line-parameters

Syntax

serial-line-parameters

Description

Returns a list of current settings of the serial line. See **configure-serial-line**.

11.12 SERIAL PORT: READING

serial-listen

Syntax

serial-listen

Description

Returns true if something is in the serial buffer.

serial-read-line

Syntax

serial-read-line

Description

Read a line from serial buffer. Hangs until a full line is there.

serial-read-line-with-timeout

Syntax

serial-read-line-with-timeout <seconds>

Description

Reads a line, waiting <seconds> before returning nothing if a full line is not available.

serial-read-line-no-hang

Syntax

serial-read-line-no-hang

Description

Returns instantly. Returns nothing if a full line is not in the buffer.

serial-read-char

Syntax

serial-read-char

Description

Read a character from serial buffer. Hangs until a character is there.

serial-read-char-with-timeout

Syntax

serial-read-char-with-timeout <seconds>

Description

Reads a character, waiting <seconds> before returning nothing if a character is not available.

serial-read-char-no-hang

Syntax

serial-read-char-no-hang

Description

Returns instantly. Returns nothing if a character is not in the buffer.

serial-read-byteSyntax**serial-read-byte**Description

Reads a byte, waiting <seconds> before returning nothing if a byte is not available.

11.13 SERIAL PORT: WRITING

serial-write

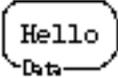
Syntax

serial-write <box>

Description

Sends text to the serial line.

Example

```
serial-write 
```

serial-write-byte

Syntax

serial-write-byte <byte>

Description

Sends one byte of information to the serial line.

11.14 SERIAL PORT: PREFERENCES

serial-read-base

Syntax

serial-read-base <radix>

Description

See System-preferences.

newline-after-serial-writes

Syntax

newline-after-serial-writes <True or False>

Description

See System-preferences

INDEX

A

abs, 134
acos, 139
add-extension, 234
alpha<, 18
alpha>, 17
and, 144
append-column, 50
append-item, 44
append-row, 47
asin, 139
ask, 9, 34
atan, 139

B

back, 65
backup-file-suffix, 223
beep, 154
bg-color-at, 99
bg-color-at?, 100
bg-color-at=, 99
bg-color-under, 96
bg-color-under?, 98
bg-color-under=, 27
boxify, 38, 52
boxtop, 229
build, 37
butcolumn, 32
butfirst, 26
butfirst-column, 32
butfirst-row, 29
butitem, 27
butlast, 27
butlast-column, 32
butlast-row, 30
but-row, 30

C

ceiling, 140
change, 40
change-column, 48
change-graphics, 88
change-item, 42
change-last-column, 48
change-last-item, 43
change-last-row, 45
change-rc, 51
change-row, 45
choose-file, 179, 232

clean, 84
clear-background, 94
cleargraphics, 84
clearscreen, 84
click-sound, 154
close-serial-line, 235
color=, 89
color-at, 92
color-at=, 92
color-under, 90
color-under=, 91
column, 31
column-numbers, 24
columns, 32
command-', 187
command-"-key, 161
command-/, 187
command-@, 187
command-<number, 187
command-c, 186
command-delete, 186
command-f, 187
command-Help, 187
command-k, 186
command-'-key, 161
command-left-arrow, 184
command-mouse-click, 162
command-mouse-double-click, 162
command-option-delete, 186
command-option-left-arrow, 184
command-option-p, 185
command-option-right-arrow, 184
command-option-v, 187
command-option-v-key, 161
command-r, 187
command-space, 187
command-t, 187
command-tab, 184
command-v, 186
command-x, 186
command-y, 186
command-z, 187
configure-serial-line, 235
copy, 53
cos, 138
count, 19
ctype, 82
current-directory, 180
cursor-column-number, 193
cursor-row-number, 193

D

datafy, 39, 52
date-and-time, 228
Delete, 186
delete-column, 49
delete-columns, 49
delete-file, 179
delete-item, 42
delete-items, 43
delete-last-column, 49
delete-last-item, 43
delete-last-row, 46
delete-rc, 51
delete-row, 46
delete-rows, 46
denominator, 142
directory, 180
distance, 110
divide, 125
down-arrow, 184
dribble-off, 231
dribble-on, 230

E

edit-box, 173
empty?, 21
enclosing-rectangle, 110
end, 184
enter, 187
entry-trigger, 148
equal?, 17
evaluator-help, 213
even?, 132
exit-trigger, 149
exp, 136
expand-box, 194
extension-info, 233

F

false, 144
first, 25
first-column, 31
first-row, 28
float?, 129
floor, 141
follow-mouse, 69
for-each-item, 7
for-each-row, 7
forward, 65
freeze, 95
fullscreen-box, 195
fullscreen-window, 222

G

get-mail, 203
global-hotspot-controls, 220
graphics-mode, 86
graphics-size, 85
greater?, 128
greater-or-equal?, 128

H

handle-input, 171
heading, 103
height, 20
Help, 187
hide-subsprites, 71
hideturtle, 70
home, 67, 184
home-position, 108

I

if, 3
ifs, 4
include-sprite-in-new-graphics, 216
input?, 172
input-device-names, 221
insert-column, 50
insert-item, 44
insert-rc, 51
insert-row, 47
integer?, 130
invisible-error, 227
invisible-value, 228
item, 25
item-numbers, 23
items, 27

J

join-bottom, 37
join-right, 38

K

-key, 160

L

last, 26
last-column, 31
last-row, 29
left, 66
left-arrow, 184
length, 20
less?, 127
less-or-equal?, 128
letters, 55

ln, 137
load-extension, 233
local-name?, 57
log, 137
loop, 6
ltype, 82

M

mail, 202
mail-relay-host, 224
make-color, 89
make-diet-sprites, 217
make-transparent-graphics-boxes, 215
mark-for-saving, 231
math & logic characters
 *, 123
 **, 136
 /, 124
 +, 123
 <, 126
 <=, 127
 =, 16, 126
 >, 126
 >=, 127
max, 135
max-viewable-message-size, 224
member?, 23
min, 135
minus, 124
minus?, 131
mod, 133
modified-trigger, 150
mouse-box, 164
mouse-box-on-, 166
mouse-buttons, 164
mouse-click, 162
mouse-click-on-bottom-left, 163
mouse-click-on-bottom-right, 163
mouse-click-on-graphics, 114
mouse-click-on-name, 163
mouse-click-on-scroll-bar, 163
mouse-click-on-sprite, 115
mouse-click-on-top-left, 163
mouse-click-on-top-right, 163
mouse-click-on-type, 163
mouse-double-click, 162
mouse-double-click-on-, 163
mouse-double-click-on-graphics, 114
mouse-double-click-on-sprite, 115
mouse-position, 116
mouse-position-on-click, 116
mouse-position-on-release, 116
mouse-rc, 169
mouse-rc-box, 167
mouse-rc-box-on-, 168

mouse-rc-on-, 170
mouse-x-position, 117
mouse-x-position-on-click, 117
mouse-x-position-on-release, 117
mouse-y-position, 118
mouse-y-position-on-click, 118
mouse-y-position-on-release, 118
move-cursor, 193

N

name?, 56
name-help, 56, 226
name-in-box?, 57
name-new-sprites, 216
names, 58
newline-after-serial-writes, 225, 240
not, 145
number?, 21, 129
number-of-columns, 19
number-of-items, 19
number-of-rows, 19
numerator, 141

O

odd?, 132
open, 175, 197
open-serial-line, 235
option left-arrow (command-a), 184
option-/ , 187
option-c, 187
option-delete, 186
option-f, 187
option-g, 185
option-Help, 187
option-l, 187
option-mouse-click, 162
option-mouse-double-click, 162
option-p, 185
option-return, 187
option-right-arrow (command-e), 184
option-s, 185
option-space, 187
option-t, 185
option-tab, 184
option-u, 187
option-x, 187
or, 145

P

pen, 105
pen-color, 107
pendown, 72
penerase, 73
penreverse, 73

penup, 72
pen-width, 106
PgDn, 184
PgUp, 184
playback-dribble-file, 231
plus, 124
plus?, 131
port-to, 53
position, 103
power, 136
preserve-empty-lines-in-build, 210
primitive-shadow-warning, 214
print-fractions, 143
printing-precision, 143
PrSc, 187

R

random, 135
rational?, 130
rationalize, 142
rc, 33
read-text-file, 181
redisplay, 158
remainder, 133
remove-extension, 234
repeat, 6
retarget, 41
right, 66
right-arrow, 184
round, 140
row, 28
row-numbers, 23
rows, 30
rtype, 83
run, 10

S

save, 177, 199
save-as, 178, 200
save-box-as, 178, 201
save-text-file, 181
self, 35
send, 204
serial-line-parameters, 236
serial-listen, 237
serial-read-base, 225, 240
serial-read-byte, 238
serial-read-char, 237
serial-read-char-no-hang, 237
serial-read-char-with-timeout, 237
serial-read-line, 237
serial-read-line-no-hang, 237
serial-read-line-with-timeout, 237
serial-write, 239

serial-write-byte, 239
set-background, 94
set-color-at, 93
set-current-directory, 180
set-default-line-parameters, 236
set-graphics-mode, 86
set-graphics-size, 85
setheading, 67
set-home-position, 113
set-pen-color, 75
set-pen-width, 74
setposition, 68
setshape, 112
set-sprite-size, 112
set-type-font, 75
setxy, 68
shape, 104
show-border-type-labels, 219
shown?, 105
show-sprite-properties, 109
show-sprites, 71
showturtle, 70
shrink-box, 194
signum, 134
sin, 138
sleep, 155
smooth-scrolling, 219
snap, 87
snip, 87
Special Character
!, 190
. (dot), 34, 76, 191
:, 192
;, 192
@, 190
[, 185
^, 11, 191
{, 185
|, 187
special characters
@, 10
], 184
, 184
sprite-size, 107
sqrt, 134
stamp, 76
stamp-arc, 80
stamp-circle, 77
stamp-ellipse, 78
stamp-hollow-circle, 77
stamp-hollow-ellipse, 78
stamp-hollow-rectangle, 79
stamp-rectangle, 79
stamp-self, 81
stamp-wedge, 80
status-line-message, 158

status-line-y-or-n, 159
step-time, 212
step-wait-for-key-press, 211
stop, 8
stop-loop, 8
superior, 36
supershrink-box, 195
system-preferences, 209

T

tab, 184
tan, 138
target-name, 58
tell, 9
terse-file-status, 223
times, 125
top-level-name?, 58
touching?, 111
towards, 111
true, 144
truncate, 141
turtle-shape, 113
type, 81
type-font, 106

U

unbox, 54

unboxable?, 22
unique-symbol, 155
unless, 4
up-arrow, 184, 187
update-, 109
update-color-box, 93
user-mail-address, 224

W

warn-about-outlink-ports, 223
when, 5
width, 20
without-recording, 101
word, 55
word?, 22

X

x-position, 102

Y

y-position, 102

Z

zero?, 131
zoom-pause, 218