**MEMO**

**WildCard Quick Reference Apri l 1987**
**Mike Farr**
**Wildcard Test Team**

**This is a quick reference for WildTalk programmers. It covers Expressions, Messages, Control structures, Commands, Functions, Error handling and WCMDs. For more information, see the Help Stacks.**

# Intro to Expressions

Expressions are the building blocks of WildTalk. In order to refer to WC objects correctly or to create text strings and numeric values, we must be able to construct legal expressions. Most Wildtalk commands make use of the expressions explained below. There are four types of expressions: logical, arithmetic, string, and container. A **logical expression** is an expression like *4 < 5* that evaluates to true or false. An **arithmetic expression** is just an expression like *4 + 5* that evaluates to a number. A **string expression** is just an expression that operates on character strings using special operators to join two or more strings together: *"this " & "that*". A **container expression** uses WildCard fields and/or variables to store logical, arithmetic, and string expressions. **Chunk expressions** like *first word* or *line 1 to 5* specify a part of a container.

## Logical Expressions:

Anything that returns "true" or "false".
May use the logical operators below to compare
1) arithmetic expressions using any one of the logical arithmetic operators (< > >= etc. ),
2) string expressions using any of the logical string operators (is in/of, is not in/of, contains)
Several expressions may be combined with **and** and **or.**

**Logical Operators**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| > | < | >= | <= | ≤ | ≥ | contains | | is in | not |
| is not in | is of | is not of | | and | or | is | <> | ≠ | = |

**Examples:**

5≤6
var1 <> first word of field "ffo"
("this that, and the other" contains "this") is not contained in bkgnd field "foo"
(var1 contains word 3 of field "blah") and (var1 > var1) or (the value of field "foobar" < 5)

## Arithmetic Expressions:

Anything that evaluates to a number. May use functions, operators, parentheses, variables, fields, or any word, line, item, or character part of variables or fields:

^ (exponent)    /    div    mod    +    -

**Examples:**

5 * 4.987 + line 4 of card field 3 - (-74.9 ^ 5 + 4 ^ (var1 +2))
the value of line 4 of field "bar" div 9

### String Expressions:

May use functions, variables, fields, and string literals ("a string"), combined with the string operators *& and &&* and chunk operators such as *word*, *line*, and *character* etc. Functions returning numbers, or variables, or fields containing numbers can be treated as strings, thus "5438" is a 4 character string not the number 5438:

### String Constants

empty formFeed lineFeed quote return space tab

**Examples**

"literal" & variable & return & linefeed & card field "bad" & third word of field "dude"
field "amos" of card id 234 & return & item 2 of field 4 & the value of ("field " & var4)

## Misc. Constants:

These miscellaneos constanst didn't fit in any where else:

| | |
|---|---|
| down | up (for testing the state of the mouse button or a key) |
| false true | pi (3.1415926589793...) |

### Examples:

if tabKey is up then ...
repeat until <some functrion> = false

## Objects

An object can send and receive messages. Wild Card contains the following types of objects:

**buttons fields cards backgrounds stacks**

Buttons and fields are contained in cards or backgrounds. They are referred to by Name, Number, and ID Number.

The following words and phrases may be used to refer to particular objects.

### Ordinals and Constants:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| any | | last | | | | | | | |
| first | second | third | fourth | fifth | sixth | seventh | eighth | ninth | tenth |
| one | two | three | four | five | six | seven | eight | nine | ten |

### Additional Ordinals for cards:

next previous prev this middle mid recent

## Containers

Containers are storage areas that may hold text or numeric values. Variables are one kind of container. Fields are another. You use the command **put** to put the value of an expression into a container as in *put "this that" into var1*.

WildCard contains the following containers:

**variables** Variables can be local to objects or global to any object in WildCard. Global variables are created with the command "global VarName" in an object's script creates a global. Every object's script containing *global VarName* shares the same variable *VarName*.

**message box, msg**
**message** can put into the message box, can type text into it, can execute commands from it

**fields** can access field's contents with put command, or with chunk expressions (see below)

**it** A global variable that always exists
**selection** When field text has been selected from within a script (by using the drag command) the selected text is put into a container called *selection*.

## Chunk Expressions

Chunk expressions like "first word of" and "line 1 to 3 of" specify the componant parts of a container. Componants--for example a word--can be used like a container. You can perform operations on, or store values into a componant just as you can into a container. For example, **put "this string" into word 2 of field "foo"**

### Basic componants of Chunk Expressions:

word character line
item --an item is a section of a string bounded by commas, "item1, item2, item3,...".

### character:

character 1 to 25 of <field designator>

### word:

word of <field designator>
word 1 of varName

### item:

item 5 of <field designator>

### line:

line 1 of <field designator> item 2 to 3 of field "foobar"

### ranges:

line 1 to 5 of field "foobar" item 2 to 3 of field "foobar"

### Examples of Chunk Expressions:

```
third word of field "foo"
character 3 to 48 of card field 5
lines 1 to 10 of bkgnd field ID 234
third word of Var1
lines 1 to 10 of third word of field "foo"
--where the third word of field "foo" is the name of a currently defined variable.
```

## Designating Objects, Containers et. al.

### Putting it all together

Arithmetic and string expressions can be used in specifying objects: *card button "foo"&"bar"*, and containers: *word (5*3+2) of field 35+Var1*.  Functions, (documented later in this reference), return numeric or string values and can also be used to specify objects or containers.  An example is

```
put line (the mousey div 12 + 32 of field "foo") into msg
```

which uses an arithmetic expression to specify a line number, in this case the line in field "foo" that the mouse is pointing to.  (The top of the field is at vertical position 32 and in 10 point type on 12 point leading.)

Throughout the rest of this document I refer to various **<designators>**.   This is short hand for the ways to specify objects and containers.  Anywhere you see an expression in <...>, you may substitute the appropriate specifying phrase.  An *<Object Designator>* is a phrase designating a particular object whereas *<card designator>* refers only to cards.  Ordinals, constants, expressions, functions results, and containers can be used to specify the ID, Name or Number of an object.

A *<container designator>* is a phrase designating a container, or a part of a container (like the first word of the container).  Ordinals, constants, expressions, functions results, and even other containers can be used to specify a container, and chunk expressions can be used to specify a part of a container.

Fields are both objects and containers.  If you send a message to a field or change its position, it will be used as an object.  If you put something into it, or refer to part of its text in a chunk expression, it will be used as a container.

### Object Designators

Only the **go** command (described below) allows you to refer directly to a card within another stack: **go to card 5 of stack "foo"**.  In all other cases, the object you refer to must be within the same stack.  If you want to refer to an object outside of the current stack you must first go to that stack.  (You can do this without the user noticing it by setting the property lockScreen to true; see the **set** command.)

#### <stack designator>

| | |
|---|---|
| **Name:** | stack "foo" |
| | stack "VolName:stackname" |
| **Number:** | (stacks not numbered) |
| **ID:** | (stacks have no ID) |

#### <card designator>

| | |
|---|---|
| **Name:** | card "foo" |
| | card "foo" of stack "VolName:stackname" |
| **Number:** | (stacks not numbered) |

|  |  |  |  |
|---|---|---|---|
| **ID:** | (stacks have no ID) |  |  |

### <button designator>

|  |  |  |  |
|---|---|---|---|
| **Name:** | button "foo" | (defaults to card button) | bkgnd button "foo" |
| **Number:** | card button 5 | background button 1 |  |
| **ID:** | card button ID 234 | background button ID 234+3 |  |

### <field designator>

|  |  |  |  |
|---|---|---|---|
| **Name:** | field "foo" | (defaults to card button) | bkgnd field "foo" |
| **Number:** | card field 5 | background field 1 |  |
| **ID:** | card field ID 234 | background field ID 234+3 |  |

**Using Ordinals:**

first card in stack third field in card recent card next card
card ten of this stack first card button?

**Using Arith Expression:**

card 5+3-2 of card

**Using String Expressions:**

card "foo" & "bar"

**Using Containers :** see below

```
card cardName                                           ;where cardName is a variable
card first item of field 1
card (first word of field (first word of var1))         ;var1 is a variable
card cardName                                           ;where cardName is a variable
card first item of field 1
card (first word of field (first word of var1))         ;var1 is a variable
first card in stack word 1 of field 3 of earlier card
bkgnd button "sam" & "fred"
```

### Complex Chunk Expressions

Ordinals, string expressions, and container expressions can be mixed together to form really complex and convoluted <designators> and chunk expressions.  Here are some examples.

```
item var2 of message                    first word of field "joe" of card 5
line five of field one of card id 548   char var1 of var2
any char of bkgnd field "sam"           character 5 of var1
last word of it                         first card field of card "foo"
first word of card field "foo"
first card in stack word 1 of field 3 of earlier card
```

## Paths

We've all had the experience of forgetting where we put a file. We've buried it so deep inside folders within folders that we need help (like the findfile DA) to discover where we put it.  WC also needs help to discover where you put your files so that it doesn't have to search the whole disk.  A **search path,** like the examples below, determine where WC will look for stacks, applications, and document files.  WC accepts complete pathnames:

```
HDSC:folder1:folder2:folder3
```

And relative pathnames:  if WC is contained in folder2 in the line above, then folder3 may be referred to as

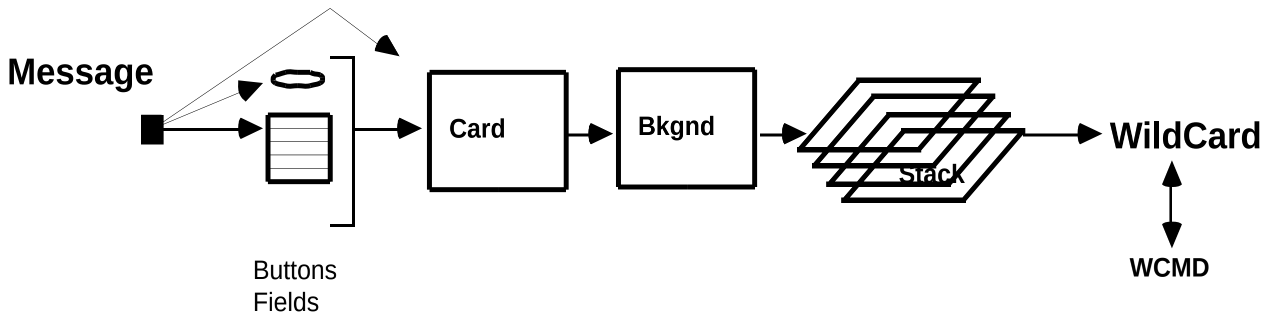
```
:folder3:
```

the complete pathname is unecessary.  When given a relative path name, WC will start looking within its own folder, folder2 and will see folder3.

# Messages

## WildCard Messages

Wildcard sends several messages automatically. These include messages when the mouse is clicked and when the user goes to a different card. Wildcard generates these automatic messages and sends them to either to a button, a field or the current card. All the messages below get sent to one of these three objects. If the object does not have a handler for the message, (an *on message ... end message* script), then the message is automatically inherited by the next higher object. A card is the next higher object for a button or field. The complete inheritence path is as follows:



We start with a message sent by WildCard to a button, field or card. In the case of a button or field, if the object has no handler for the message then the message is inherited by the card containing the button or field. When a card receives a message, whether it be from WC or inherited from an button or field, it acts on the message if it has ahandler for it, or lets it be inherited by the background containing the card. The background may acto on it or pass it along to the stack, and then perhaps to WildCard. WildCard will check to see if it knows what to do with the message, (for example, *arrowKey left*) or if it has a user added WCMD for the message (for example, SoundCapToRes).

It may seem more logical that some messages, such as openBackground, be sent to the background, instead of to the card. Within WC, it is up to inheritence to get the message where it is supposed to go.

Messages typed into the message box are sent to the card.

## User Messages

There are several ways for the user to send messages. The first is by just naming the message from within a script or the message box. The user can send predefined WC messages like mouseDown, openCard, or his own messages. For example, one might have a script for a button that contains:

```
on mouseUp
        messageName
end mouseUp

on messageName
        ...
end messageName
```

WildCard will momentarily suspend the button's script and will send the message *messageName* to the current object. In this case the button. If the button script contains a handler for the message messageName as shown above, then messageName would get run. If there is no messageName handler, then the message would get inherited as normal. When the message is finished running, or no handler is found for it, then the button's message will resume running.

Messages can also **pass arguments** to their message handlers in this form: *message arg1, arg2, arg3,...* Spaces are allowed after commas. Here is an example:

```
messageName "a string", word 1 of field 2, stringWithNoSpaces, var1
```

The handler for this message looks like:

```
on messageName arg1, arg2, arg3, arg4
        put arg2 into it
        put word 1 of field arg1
        put 5 into arg4
        ...
end messageName
```

## Passing Variables

When a variable is passed as an argument to a message, it is passed like a *var* parameter in Pascal. That is, when the receiving handler changes the value of its parameter, it also changes the value of the variable used by the caller. This way the receiving handler can pass values back to the calling handler.

The user can also send a message from a script or the message box and specify the target as well:

```
send "mouseUp "
        -or-
 send "mouseUp " to button "foo"
        -or-
send "myMessage arg1, arg2" to button "foo"
```

Using send, the message name and arguments must be passed within quotes. This limits the arguments to being variables or one word unquoted strings, as WildCard cannot have quotes within quotes.

A script can also process a message and then pass the message on:

```
pass "mouseUp"
```

This allows the message to be executed within the script, and then to be inherited as well. A message handler may only pass the same message on, e.g. a mouseUp handler may only pass mouseUp. See the messages section.

## Mouse messages:

| | |
|---|---|
| mouseEnter | is sent to a button or a field when the mouse enters it, providing the mouse button is not down. |
| mouseWithin | is sent to a button or field periodically while the mouse cursor is within it. |

| | |
|---|---|
| mouseLeave | is sent to a button or field when the mouse cursor leaves it. |
| mouseDown | is sent to a button when the mouse is clicked within it. It is sent to a field only if the mouse is within the field and the field's text is locked. If the text is not locked, then the mouseDown is interpreted as the selection of an insertion point. If the mouse is not over any field or button, then the mouseDown is sent to the Card. Not sent to hidden buttons or fields. |
| mouseStillDown | sent to a button periodically while the mouse button is held down. It is sent to a field only if the field's text is locked. |
| mouseUp | is sent to a button or card if the previous mouseDown was also sent to that button or card. MouseUp is sent to a field if the field's text is locked, and the preceding mouseDown was also sent to that field. If the mouse is clicked within a button or field and dragged outside it, no mouseUp is sent to any object. |
| mouseLeave | is sent to a button or field when the mouse cursor leaves it. |

**Mouse messages are not sent to hidden buttons or fields.**

## New messages:

| | |
|---|---|
| newButton | Sent to the newly created button. |
| newField | Sent to the newly created field. |
| newCard | Sent to the newly created card. |
| newBackground | Sent to the card created at the same time as the newly created background. |
| newStack | Sent to the card created at the same time as the newly created stack. |

## Delete messages:

| | |
|---|---|
| deleteButton | sent (as a final wish) to the condemned button before it is deleted. |
| deleteField | sent to the field before it is deleted. |
| deleteCard | sent to the card before it is deleted. |
| deleteBackground | sent to the current card before its background (and iself) is deleted. |
| deleteStack | sent to the current card before its stack (and iself) is deleted. |

## open messages:

| | |
|---|---|
| openField | sent to the field when the user clicks within it to get an insertion point |
| openCard | sent to a card when the user "opens" or goes to it. If a card is opened from within a script, the script is momentarily interrupted and the openCard message is sent to the card and handled or inherited. |
| openBackground | sent to a newly opend card when it shares a different background from the previous card . |
| openStack | sent to the first card opened when the user goes to a new stack. |

### Close messages:

| | |
|---|---|
| closefield | sent to the field when the user has altered its contents and then clicked the mouse outside the field. |
| closecard | sent to the current card just before going to a new card. |
| closebackground | sent to the card just before opening a new card that has a different background. |
| closestack | sent to the current card just before opening a new stack. |

### Special messages:

arrowKey {right | left | up | down}

sent to the current card.  If the user does not intercept the arrowKey message, then depending on the value of the argument, WC will :

| **Argument** | **Action** |
|---|---|
| right | go to the next card |
| left | go to the previous card |
| up | push the current card on the card stack |
| down | pop the card stack |

| | |
|---|---|
| help | send to the current card when the user types "help" in the message box, or when he types cmd-? anywhere except while entering text into a field. |
| idle | periodically sent to the current card |
| resume | sent to the current card when returning from application |
| returnKey | sent to the current card when the return key is hit, and no insertion point is currently set in a field. |
| startUp | sent to the current card when the program is fired up (first card in Home stack if WC double clicked.) |
| suspend | sent to the current card when an "open application" command is executed.  The application is opened anyway. |
| enterKey | sent to the current card, when the enter key is hit, and no insertion point is currently set in a field. |
| tabKey | sent to the current card when the tab key is pressed, providing an insertion point is not currently set in a field. Can also be sent by typing tabKey in the message box. |
| userMessage | any message name.  If typed from the message box, it is is sent to the current card.  If sent from within a message handler, then it is sent to the target of the original message, and is then available for inheritence. |

## Control Structures

The following are legal control structures.

### if then else

if <logical expression> then <statement>        if var1 < var2 +4 then add 1 to var1

```
if <logical expression> then          if var1 +4 > 5 then
      <statement>                             put var1 into msg
      <statement>                             add 1 to var1
end if                                     end if


if <logical expression> then
      <statement>
      <statement>
else
      <statement>
      <statement>
end if
```

Example of nested ifs:

```
if <logical expression> then
      <statement>
      if <logical expression> then <statement>

      <statement>

      if <logical expression> then
            <statement>
            <statement>
      else
            <statement>
            <statement>
      end if
end if
```

**repeat**

```
repeat while <logical expression>                --repeat while the mouse is down
      <statement>
      <statement>
end repeat


repeat until <logical expression>  --repeat until var1 < var2
      <statement>
      <statement>
end repeat

repeat with <cont. exp.> = <arith. exp.> to <arith. exp.>
      <statement>                                 --repeat with indexVar = 1 to 5
      <statement>
end repeat

repeat with <cont. exp.> = <arith. exp.> down to  <arith. exp.>
      <statement>                                 --repeat with indexVar = 15 down to 5
      <statement>
end repeat
```

## Commands

Notation used:

| | |
|---|---|
| <some arg> | means insert a proper argument to the command or function |
| [ ] | indicates an optional structure, may be excluded from command |
| { } | indicates choose one of the possibilities |
| \| | separates possible choices within [] or {}. Ex: [<choice1>\|<choice2>] or {<choice> \| <choice>} |
| * | means repeat the preceding argument any number of times. For example you may pass many musical notes to the play command. |

| | |
|---|---|
| <cont. exp.> | any container or part of one. See *containers* earlier in this paper |
| <arith. exp.> | any expression that returns an arithmetic value. See *aritimetic expressions* |
| <logical exp.> | any expression that evaluates to true or false. See *logical expressions* |
| <string exp.> | an expression using strings and string operators *&* and *&&* |

```
add       <dest>          to     <source>
            <cont. exp.>         <cont. exp.>
            <arith. exp.>
          add (var1+4) to third word of line2 of field "foo"

answer    <question>      [with  <reply>] [or <reply>] [or <reply>]
            <cont. exp.>         <cont. exp.>
            <string exp.> <string exp.>
            <arith. exp>         <arith. exp.>
          answer "Name of stack:" with Var1 or Var2 or "Help"

ask       <question>      [with  <reply>]
            <cont. exp.>         <cont. exp.>
            <string exp.> <string exp.>
            <arith. exp>         <arith. exp.>
          ask line 1 of field "bar" with pathVar1

beep      <number of beeps>
            <arith. exp.>
            <cont. exp.>
           beep 4*4

choose    <tool name>     tool
            browse   button     field  select     lasso   pencil        brush   eraser
          line rectangle round rect bucket        oval    curve  text   regular
polygon   polygon    spray
click     at <horPos>, <vertPos>
          click at 50,100     range = 0-511 for horizontal, and 0-341 for vertical
```

**close file** **<filename>** ;closes file for reading or writing (see read, write)
<cont. exp.> close file first line of field 5
close file var1
<string exp.> close file "HD:folder:foobar"
close file "CustomerRec" & short date

**delete** **< container exp.>** ;delete contents of container, part of container or object
delete characters 1 to 3 to last of var1
delete word 1 to 5 of field "sam"

**dial** **<phone number> [with [modem] <hayes modem commands>]**
<cont. exp.> Dialing with modem takes the <hayes modem commands>
and prepends them to the <phone number> My favorite are
ATTD which means get the ATtention of the modem, and Tone
Dial the number.
<string exp.>
dial "408-973-6683" with ATTD
dial first line of field "Phone Number" with ATTD

**divide** **<dest>** **by** **<source>**
<cont. exp.> <cont. exp.>
<arith. exp.>
divide third word of var1 by word three of field 2

**domenu** **<menu item>**
If a menu is more than one word, make sure the menu item is in quotes.
domenu "new card..." domenu "Find File"

**drag** **from <xpos>,<ypos> to <xpos>,<ypos>**
Move the mouse cursor from position to position just as if moved by the user.
move from 20,30 to 40,50 You may drag off the visible screen. If drawing, WC will properly clip to the screen. Visible screen = 0-511 for horizontal, and 0-341 for vertical.

**flash** **<number of times>**
flash 5+3-2
flash var1
flash word 1 of field "foobar"

**find** **[chars | word]** **<string pattern> [of | in] { <field designator> }**
<cont. exp.>
<string exp.>
find "string" in card field 1 ;find word starting with "string" in field 1
find chars first word of Var1 in first field of card 4 ;find first word of Var1 in field 4
find word "The rat" ;find exactly "the rat" in any field

**get <property> [of | in] <target> ->property returned in it**

NOTE: An <object designator> as used below is just an expression that specifies an object. Container or string expressions may be used. The following string expression is a sample of a <button designator>:

*card button ID 3834 of card "foo" of background 3*.

For more information on <object designators> see the section on expressions.

**WildCard Properties**

| | |
|---|---|
| fullMenus | -> true \| false |
| powerKeys | -> true \| false |
| lockScreen | -> true \| false |
| pattern | -> 1 .. 40 |
| lineSize | -> 1.. 6 |
| userLevel | -> 1 .. 5 |
| brush | -> 1 .. 32 |

**Stack Properties**

| | | | |
|---|---|---|---|
| | freeSize | of <stack designator> | -> space left in current allocation record |
| | size | of <stack designator> | ->size of thestack in bytes |
| | name | of <stack designator> | -> stack "name " |
| short | name | of <stack designator> | -> name |
| longname | | of <stack designator> | -> stack ":MyStacks:Note" |
| | script | of <stack designator> | -> the text of the script in it |

**Bkgnd Properties**

| | | | |
|---|---|---|---|
| | name | of <bkgnd. designator> | -> bkgnd "name " |
| short | name | of <bkgnd. designator> | -> name |
| longname | | of <bkgnd. designator> | -> bkgnd "blah" of stack ":MyStacks:Note" if bkgnd has no name, its ID will be returned |
| | id | of <bkgnd. designator> | -> 434 |
| | script | of <bkgnd. designator> | -> the text of the script in it |

**Card Properties**

| | | | |
|---|---|---|---|
| | name | of <card designator> | -> card "name" |
| short | name | of <card designator> | -> name |
| longname | | of <card designator> | -> card "name" of stack ":MyStacks:Note" if card has no name, its ID will be returned |
| | number | of <card designator> | -> the no. of the card in the stack |
| | id | of <card designator> | -> card id 345 |
| short | id | of <card designator> | -> 345 |
| longid | | of <card designator> | -> card id 345 of stack ":MyStacks:Note" |
| | script | of <card designator> | -> the text of the script in it |

**Field Properties**

| | | | |
|---|---|---|---|
| | name | of <field designator> | -> field "foobar" |
| short | name | of <field designator>" | -> foobar |
| long | name | of <field designator> | -> bkgnd field "foobar" of card ID 345 of stack ":MyStacks:foo" if field has no name, its ID will be returned. |
| | number | of <field designator> | -> the number of the field in the background or card |
| | style | of <field designator> | -> transparent, opaque, rect, rectangle, shadow |
| id | | of <field designator> | -> 234 |
| | loc, location | of <field designator> | ->xpos, ypos |
| | textAlign | of <field designator> | -> center \| left\| right |
| | textFont | of <field designator> | ->Font Name |
| | textSize | of <field designator> | -> number |
| | textStyle | of <field designator> | Bold\|Plain\|Italic \|Underline\|Outline \|Shadow\|Condense \|Expand |
| | textHeight | of <field designator> | -> number |
| | lockText | of <field designator> | -> true or false |
| | showLines | of <field designator> | -> true or false |
| | wideMargins | of <field designator> | -> true or false |
| | hidden | of <field designator> | -> true or false |
| | script | of <field designator> | -> the text of the script in it |

**Button Properties:**

| | | | |
|---|---|---|---|
| | icon | of <button designator> | -> <icon number> |
| | name | of <button designator> | -> button "do it" |
| short | name | of <button designator> | -> button "do it" |
| longname | | of <button designator> | -> button "do it" of card id 345 of stack ":MyStacks:Note" if button has no name, its ID will be returned. |
| | number | of <button designator> | -> 3 |
| | id | of <button designator> | -> 345 |
| | showName | of <button designator> | ->true or false |
| | hidden | of <button designator> | ->true or false |
| | size | of <button designator> | ->x, y, x, y |
| | location,loc | of <button designator> | ->xpos, ypos |
| | style | of <button designator> | ->transparent \|round rect, rectangle \| radioButton \|checkBox |
| | hilite | of <button designator> | -> true\| false |
| | icon | of <button designator> | -> the icon number |
| | script | of <button designator> | -> the text of the script into it |

## global <variable name> [, <variable name>, <variable name>... ]*

Any name. Becomes a variable container accessable from any object's script. Once a script has declared a global of a given name, any other script declaring a global of the same name will use the same variable.

<u>global foo, bar, sam, space, fatman</u>

**go** **[to] {card <card designator>} {of <stack designator>}**

go to card "mycard"of stack "mystack"
go to card id 89A4 of stack varName
go to card varname of stack "mystack"&short date
go to card (first word of var2)
go to first card of stack word 1 of var2

**[go] help**

**hide** **<button or field designator>**

hide button id 5 of card "foobar"
hide field "fun"

**hide** **<window name>**

*pattern windowtool window message | message box | msg*

hide msg
hide pattern window

**lock** **screen**

**mark** **card**

**multiply** **<dest>** **by** **<source>**

| | |
|---|---|
| <cont. exp.> | <cont. exp.> |
| | <arith. exp.> |

multiply third word of field "foobar" by 25
multiply var1 by 23*2+4^2

**open** **file <textfile>** ;open file for reading or writing

<cont. exp.> ;creates file if doesn't already exist
<string exp.> ;quote paths or filenames that contain spaces, or periods

Open is used in conjuction with read, write and close:

```
open file "filename"
write"string" & tabkey & "string" & return to file "filename"
close file "filename"
open file "filename"
read from file "filename" until "g"
read from file "filename" for 5
close file "filename"
```

Notes: You must close a file before reading back anything you send to it. You always start reading from the beginning of a file. To write to the end of a file, read to the end and then write. If you write into the middle of the file, the rest of the file will be lost. Currently, (will soon be fixed) if you close a file before reading to the end, andy data after the last char you read will be lost. A mistyped read or write command will also have this effect, closing ALL open files and truncating them.

**open** **<Application>**

open MacDraw

**open** **<file name>** **with <Application>**

| | |
|---|---|
| <cont. exp.> | <cont. exp.> |
| string exp. | variable |

variable string exp.
open file "textfile" with "Word"
open file fileName1 with App1

**play** **<sound > [tempo <arith. exp>] "{ <note> [octave] [#|b] [duration]"}*"**

| | | | |
|---|---|---|---|
| boing | 200 is medium | 4 = middle octave | |
| | | w = whole note | |
| for more info see help stack | | q = | quarter note |
| | | | e = eighth note |
| snd resources must be moved with ResEdit | | | s = 16th note |
| **Example:** | | | t = 32nd note |

play "Boing" tempo 120 "e c d# g3h. gq d4 e3 d4 e3 d4 e3 cbw."
"." for dotted,
3 in d4e3 for triplet
# for sharp or b for flat

**pop** **card**

pop card ;return to that card

**print** **<filename> with <Application>**

| | |
|---|---|
| <cont. exp.> | <cont. exp.> |
| <string exp.> | <string exp.> |

print file "textfile" with "Word"
print file fileName1 with App1

**push** **<card designator>**

push last card ;save path to last card in this stack
push earlier card ;save path to the card we just came from
push message ;restore previous state of message box
push card id 54 of stack "foobar"

**put** **<source> <preposition><destination>**

| | | |
|---|---|---|
| <cont. exp.> | before | <cont. exp.> |
| <arith. exp.> | after | |
| <string exp.> | into | |

put "string" into field 1
put var1 into iy
put third word of card field "foo" after bkgnd field Var1 of card "sam"
put character 4 of word (word 2 of card field "ffoo") into word 2 of message
put 35^2/.002
put "open FnameVar1 with" && CurrentApp into it

**read** **from file <filename> until <single char>**

| | |
|---|---|
| <cont. exp.> | <cont. exp.> |
| <string exp.> | <string exp.> |

read from file "sentences" until return
read from file var1 until "&"
See *open* command for fulle example and notes .

| | | | | |
|---|---|---|---|---|
| **read** | **from file** | **<filename> for** | | **<number of bytes>** |
| | | <cont. exp.> | <cont. exp.> | |
| | | <string exp.> | <arith. exp.> | |

read from file "sentences" until return

read from file var1 until "&"

See *open* command for fulle example and notes .

| | | |
|---|---|---|
| **send** | **"<messageName> [<arg1>, <arg2> ...]" to** | **<object designator>** |
| messageName is any | <string exp.> | card, field, button, bkgnd, |
| WC message or any | <cont. exp.> | (home?) |
| user defined message | Args may not contain quotes " ". | |

send "mouseUp" to button ID 345 of card "foo"

send "myMsg var1, string, var2" to this stack

By using a message name as the first word on a line in a script, that message is sent to the current object and then possibly inherited.  See the Messages section above.

myMessage var1, "first word field 2"

## set <property> [of | in] <target>

NOTE: An <object designator> as used below is just an expression that specifies an object. Container or string expressions may be used. The following string expression is a sample of a <button designator>:

*card button ID 3834 of card "foo" of background 3*.

For more information on <object designators> see the section on expressions.

**WildCard Properties**

| | |
|---|---|
| fullMenus | to true \| false |
| powerKeys | to true \| false |
| userLevel | to 1 .. 5 |
| brush | to 1 .. 32 |
| pattern | to 1 .. 40 |
| lineSize | to 1 .. 6 |

**Stack Properties**

| | | |
|---|---|---|
| name | of <stack designator> | to <cont. exp.> or <string exp.> |
| script | of <stack designator> | to <container or string exp.> |

**Bkgnd Properties**

| | | |
|---|---|---|
| name | of <bkgnd designator> | to <cont. exp.> or <string exp.> |
| script | of <bkgnd designator> | to <container or string exp.> |

**Card Properties**

| | | |
|---|---|---|
| name | of <card designator> | to <cont. exp.> or <string exp.> |
| script | of <card designator> | to < container or string exp.> |

**Field Properties**

| | | |
|---|---|---|
| name | of <field designator> | to <cont. exp.> or <string exp.> |
| number | of <field designator>" | to <arith. exp.> |
| style | of <field designator> | to transparent\|opaque\| rectangle \|shadow \| rect |
| loc, location | of <field designator> | to horPos, vertPos |
| textAlign | of <field designator> | to center \| left\| right |
| textFont | of <field designator> | to <cont. exp.> that = a fontname |
| textSize | of <field designator> | to <arith. exp.> |
| textStyle | of <field designator> | to Bold\|Plain\|Italic\|Underline \|Outline\|Shadow\|Condense\|Expand |
| textHeight | of <field designator> | to <arith. exp.> |
| lockText | of <field designator> | to true \| false |
| showLines | of <field designator> | to true \| false |
| wideMargins | of <field designator> | to true \| false |
| hidden | of <field designator> | to true \| false |
| script | of <field designator> | to <cont. exp.> or <string exp.> |

**Button Properties:**

| | |
|---|---|
| icon | of <button designator> to <icon number> |
| name | of <button designator> to <string exp.> |
| number | of <button designator> to <arith. exp.> |
| id | of <button designator> to <arith. exp.> |
| showName | of <button designator> to true \| false |
| hidden | of <button designator> totrue \| false |
| size | of <button designator> to x, y, x, y |
| location,loc | of <button designator> to horpos, vertPos |
| style | of <button designator> to transparent\|round rect\|rectangle \| radioButton \| checkBox |
| icon | of <button designator> to <arith.exp.> (an icon number) |
| hilite | of <button designator> to true \| false |
| script | of <button designator> to < container or string exp.> |

**show** **<window name>** **[at <horPos>, <vertPos>]** tool window| pattern window
0 - xbound 0 - ybound
message box <cont. exp> <arith. exp>
<u>show tool window at 50, 50</u>
<u>show message box at HorPos, VertPos</u>
Screen boundries for Mac Plus and SC are 0 - 511 and 0 - 341

**show** **<buttonOrField>** **[at <horPos>, <vertPos>]** <container exp.>
0 - HorMax 0 - VertMax (unchecked by WT)
<string exp.>
Screen boundries for Mac Plus and SC are 0 - 511 and 0 - 341
<u>show button 5 at 30,20</u>
<u>show button third word of field "foo" at Xpos, Ypos</u>
<u>show line 1 of field "foo" at line 2 of Var2</u> ;line 1 of field contains button or field

**show** **<arith. exp.> cards**
<u>show 5 cards</u>
<u>show all cards</u>
<u>show word 1 of var1 cards</u>

**sort** **[ascending | descending]** **[text | numeric | international] by <field exp.>**
<u>sort ascending text by field 1</u>
<u>sort by line 1 of field "foobar"</u>
<u>sort numeric by word 1 of field "foobar"</u>

**subtract** **<source>** **from <dest>**
<cont. exp.> <cont. exp.>
<arith. exp.>
<u>subtract 35.08 from line 3 of field "foo"</u>

**visual** **[effect]** **<effect name>** **[<speed>]**

| | | | | | |
|---|---|---|---|---|---|
| *plain* | *dissolve* | *scroll right* | *barn door close* | *slow* | *very slow* |
| *scroll up* | *scroll down* | *iris open* | *barn door open* | *fast* | *very fast* |
| *iris close* | *checkerboard* | *wipe left* | | | |
| *wipe right* | *wipe up* | | *wipe down* | | |

**unlock** **screen**

| | | |
|---|---|---|
| **wait** | **<time amount>** | **<time scale>** |
| | <cont. exp.> | ticks |
| | <arith. exp.> | seconds |
| | <u>wait 25 seconds</u> | <u>wait 2300 ticks</u> |

| | | |
|---|---|---|
| **write** | **<string expression> to** | **<filename>** |
| | <cont. exp.> | <cont. exp.> |
| | <string exp.> | <arith. exp.> |

<u>write "string" & tabkey & "string" & return</u>
See *open* command for complete example and notes.

**wait** **<conditional exp.>**

forever
*until* <boolean exp.>
*while* <boolean exp.>

<u>wait forever</u>
<u>wait until the returnKey is down</u>
<u>wait while the mouse is up</u>
<u>wait until the mousev > 50</u>
<u>wait field 1</u>

## Functions:

In order to tell functions from commands, messages or user variable names, you must include the word *the*, before the function name, *of,* after the function name, or parenthesis *()* around the arguments to the functionwhen you call a function. The following forms are acceptable:

[the] **functionName** (arg1,afg2,...)
[the] **functionName** of arg1, arg2,...
the **functionName** [of]

The functions below appear in **the funcName of** format.

| Function | | Arguments | Returns |
|---|---|---|---|
| **the** | **chartonum of** | <an ascii char> | the ascii number of the char |
| **the** | **clickloc** | | xpos, ypos of where mouse was clicked |
| **the** | **commandKey** | | up\|Down; Is the command key up or down? |
| **the** | **date** | | 4/20/87 |
| **the** | **long date** | | Tuesday March 19, 1987 |
| **the** | **day week** | | 1-7 |
| **the** | **day year** | | 1-366 |
| **the** | **day** {**month**} | | 1-31 |
| **the** | **heapspace** | | some large number, amount of space left in heap |
| **the** | **hour** | | 0-23 |
| **the** | **length** | **of** <stack\|field> | no. of cards in the stack \| no. of chars in field |
| **the** | **minute** | | 0-59 |
| **the** | **month** | | 1-12 |
| **the** | **mouse** | | up\|down |
| **the** | **mouseclick** | true\|false (don't know what this is) | |
| **the** | **mouseh** | | xpos (0 to 511 or greater) of mouse. |

| | | | | |
|---|---|---|---|---|
| **the** | **mouseloc** | | | xpos,ypos of mouse. |
| **the** | **mousev** | | | ypos (o-341 or greater) of mouse. |
| **the** | **number** | **of** | **cards** | number of cards in the current stack. |
| **the** | **number** | **of** | **buttons\|fields** | number of bkgnd, or fgnd btns or flds in the card. |
| **the** | **number** | **of** | **chars\| words\|lines** **\| items** [ **of** \| **in** ] <container\|field designator> | How many there are. |
| **the** | **numtochar** | **of** | <arith. exp.> | a char, the ascii char associated with the number. |
| **the** | **offset** | **of** | <char exp.> [ **of** \| **in** ] <field> | offset from start of container to the char. |
| **the** | **optionKey** | | | up\|down; is option key up or down? |
| **the** | **param** | **of** | <arith. exp> | the nth parameter to the current message |
| **the** | **paramcount** | | | number of parameters |
| **the** | **params** | | | the parameter list |
| **the** | **random** | **of** | <arith. exp. for upperBound> | integer (0 - upperBound) |
| **the** | **seconds** | | | unsigned integer |
| **the** | **secs** | | | unsigned Integer |
| **the** | **shiftKey** | | | up\|Down |
| **the** | **sounddone** | | | true\|false |
| **the** | **stackspace** | | | unsigned Integer, amount of stack space left |
| **the** | **target** | | | string indicating the original recipient of the current message, e.g. card id 235, button id 2345 |
| **the** | **ticks** | | | 60ths of second since boot |
| **the** | **time** | | | 2:34: PM |
| **the** | **long time** | | | 2:34:18 PM |
| **the** | **tool** | | | browse\|button\|field\|various art tools |
| **the** | **value** | **of** | <container \| <string exp.> | evaluate the expression. Currently we on the test team are not sure of the exact specifications of **value of**. The following is our the argument is a container that contains an arith exp. then **value of** returns the numeric value of that expression. If it the container contains a string expression, then return its string value. In the case of an actual in line string or arith. exp. being passed as an argument, evaluate the expression. Thus, **value of** "string exp" and **value of** a container that contains "string exp" will return the same thing. **Value of** only does one level of evaluation. Thus if the value returned by **value of** was, say, a variable name, you could then take the **value of** that variable. Thus expressions like value of (value of first line of field "foobar") would be legal. |
| **the** | **year** | | | unsigned Integer: 1986 |

## Error Handling

Unspecified as of this writing.

## WCMD Format

Unspecified as of this writing.