NAME

dict − Manipulate dictionaries

SYNOPSIS

dict *option arg* ?*arg ...*?

DESCRIPTION

Performs one of several operations on dictionary values or variables containing dictionary values (see the **DICTIONARY VALUES** section below for a description), depending on *option*. The legal *option*s (which may be abbreviated) are:

dict append *dictionaryVariable key* ?*string ...*?

This appends the given string (or strings) to the value that the given key maps to in the dictionary value contained in the given variable, writing the resulting dictionary value back to that variable. Non-existent keys are treated as if they map to an empty string. The updated dictionary value is returned.

dict create ?*key value ...*?

Return a new dictionary that contains each of the key/value mappings listed as arguments (keys and values alternating, with each key being followed by its associated value.)

dict exists *dictionaryValue key* ?*key ...*?

This returns a boolean value indicating whether the given key (or path of keys through a set of nested dictionaries) exists in the given dictionary value. This returns a true value exactly when **dict get** on that path will succeed.

dict filter *dictionaryValue filterType arg* ?*arg ...*?

This takes a dictionary value and returns a new dictionary that contains just those key/value pairs that match the specified filter type (which may be abbreviated.) Supported filter types are:

dict filter *dictionaryValue* **key** ?*globPattern ...*?

The key rule only matches those key/value pairs whose keys match any of the given patterns (in the style of **string match**.)

dict filter *dictionaryValue* **script {***keyVariable valueVariable***}** *script*

The script rule tests for matching by assigning the key to the *keyVariable* and the value to the *valueVariable*, and then evaluating the given script which should result in a boolean value (with the key/value pair only being included in the result of the **dict filter** when a true value is returned.) Note that the first argument after the rule selection word is a two-element list. If the *script* returns with a condition of **TCL_BREAK**, no further key/value pairs are considered for inclusion in the resulting dictionary, and a condition of **TCL_CONTINUE** is equivalent to a false result. The key/value pairs are tested in the order in which the keys were inserted into the dictionary.

dict filter *dictionaryValue* **value** ?*globPattern ...*?

The value rule only matches those key/value pairs whose values match any of the given patterns (in the style of **string match**.)

dict for {*keyVariable valueVariable***}** *dictionaryValue body*

This command takes three arguments, the first a two-element list of variable names (for the key and value respectively of each mapping in the dictionary), the second the dictionary value to iterate across, and the third a script to be evaluated for each mapping with the key and value variables set appropriately (in the manner of **foreach**.) The result of the command is an empty string. If any evaluation of the body generates a **TCL_BREAK** result, no further pairs from the dictionary will be iterated over and the **dict for** command will terminate successfully immediately. If any evaluation of the body generates a **TCL_CONTINUE** result, this shall be treated exactly like a normal **TCL_OK** result. The order of iteration is the order in which the keys were inserted into the dictionary.

dict get *dictionaryValue* ?*key ...*?
Given a dictionary value (first argument) and a key (second argument), this will retrieve the value for that key. Where several keys are supplied, the behaviour of the command shall be as if the result of **dict get $dictVal $key** was passed as the first argument to **dict get** with the remaining arguments as second (and possibly subsequent) arguments. This facilitates lookups in nested dictionaries. For example, the following two commands are equivalent:

```
dict get $dict foo bar spong
dict get [dict get [dict get $dict foo] bar] spong
```

If no keys are provided, **dict get** will return a list containing pairs of elements in a manner similar to **array get**. That is, the first element of each pair would be the key and the second element would be the value for that key.

It is an error to attempt to retrieve a value for a key that is not present in the dictionary.

dict incr *dictionaryVariable key* ?*increment*?
This adds the given increment value (an integer that defaults to 1 if not specified) to the value that the given key maps to in the dictionary value contained in the given variable, writing the resulting dictionary value back to that variable. Non-existent keys are treated as if they map to 0. It is an error to increment a value for an existing key if that value is not an integer. The updated dictionary value is returned.

dict info *dictionaryValue*
This returns information (intended for display to people) about the given dictionary though the format of this data is dependent on the implementation of the dictionary. For dictionaries that are implemented by hash tables, it is expected that this will return the string produced by **Tcl_HashStats**, similar to **array statistics**.

dict keys *dictionaryValue* ?*globPattern*?
Return a list of all keys in the given dictionary value. If a pattern is supplied, only those keys that match it (according to the rules of **string match**) will be returned. The returned keys will be in the order that they were inserted into the dictionary.

dict lappend *dictionaryVariable key* ?*value ...*?
This appends the given items to the list value that the given key maps to in the dictionary value contained in the given variable, writing the resulting dictionary value back to that variable. Non-existent keys are treated as if they map to an empty list, and it is legal for there to be no items to append to the list. It is an error for the value that the key maps to to not be representable as a list. The updated dictionary value is returned.

dict map {*keyVariable valueVariable*} *dictionaryValue body*
This command applies a transformation to each element of a dictionary, returning a new dictionary. It takes three arguments: the first is a two-element list of variable names (for the key and value respectively of each mapping in the dictionary), the second the dictionary value to iterate across, and the third a script to be evaluated for each mapping with the key and value variables set appropriately (in the manner of **lmap**). In an iteration where the evaluated script completes normally (**TCL_OK**, as opposed to an **error**, etc.) the result of the script is put into an accumulator dictionary using the key that is the current contents of the *keyVariable* variable at that point. The result of the **dict map** command is the accumulator dictionary after all keys have been iterated over.

If the evaluation of the body for any particular step generates a **break**, no further pairs from the dictionary will be iterated over and the **dict map** command will terminate successfully immediately. If the evaluation of the body for a particular step generates a **continue** result, the current iteration is aborted and the accumulator dictionary is not modified. The order of iteration is the natural order of the dictionary (typically the order in which the keys were added to the dictionary; the order is the same as that used in **dict for**).

dict merge ?*dictionaryValue* ...?
: Return a dictionary that contains the contents of each of the *dictionaryValue* arguments. Where two (or more) dictionaries contain a mapping for the same key, the resulting dictionary maps that key to the value according to the last dictionary on the command line containing a mapping for that key.

dict remove *dictionaryValue* ?*key* ...?
: Return a new dictionary that is a copy of an old one passed in as first argument except without mappings for each of the keys listed. It is legal for there to be no keys to remove, and it also legal for any of the keys to be removed to not be present in the input dictionary in the first place.

dict replace *dictionaryValue* ?*key value* ...?
: Return a new dictionary that is a copy of an old one passed in as first argument except with some values different or some extra key/value pairs added. It is legal for this command to be called with no key/value pairs, but illegal for this command to be called with a key but no value.

dict set *dictionaryVariable key* ?*key* ...? *value*
: This operation takes the name of a variable containing a dictionary value and places an updated dictionary value in that variable containing a mapping from the given key to the given value. When multiple keys are present, this operation creates or updates a chain of nested dictionaries. The updated dictionary value is returned.

dict size *dictionaryValue*
: Return the number of key/value mappings in the given dictionary value.

dict unset *dictionaryVariable key* ?*key* ...?
: This operation (the companion to **dict set**) takes the name of a variable containing a dictionary value and places an updated dictionary value in that variable that does not contain a mapping for the given key. Where multiple keys are present, this describes a path through nested dictionaries to the mapping to remove. At least one key must be specified, but the last key on the key-path need not exist. All other components on the path must exist. The updated dictionary value is returned.

dict update *dictionaryVariable key varName* ?*key varName* ...? *body*
: Execute the Tcl script in *body* with the value for each *key* (as found by reading the dictionary value in *dictionaryVariable*) mapped to the variable *varName*. There may be multiple *key*/*varName* pairs. If a *key* does not have a mapping, that corresponds to an unset *varName*. When *body* terminates, any changes made to the *varName*s is reflected back to the dictionary within *dictionaryVariable* (unless *dictionaryVariable* itself becomes unreadable, when all updates are silently discarded), even if the result of *body* is an error or some other kind of exceptional exit. The result of **dict update** is (unless some kind of error occurs) the result of the evaluation of *body*.

  Each *varName* is mapped in the scope enclosing the **dict update**; it is recommended that this command only be used in a local scope (**proc**edure, lambda term for **apply**, or method). Because of this, the variables set by **dict update** will continue to exist after the command finishes (unless explicitly **unset**). Note that the mapping of values to variables does not use traces; changes to the *dictionaryVariable*'s contents only happen when *body* terminates.

dict values *dictionaryValue* ?*globPattern*?
: Return a list of all values in the given dictionary value. If a pattern is supplied, only those values that match it (according to the rules of **string match**) will be returned. The returned values will be in the order of that the keys associated with those values were inserted into the dictionary.

dict with *dictionaryVariable* ?*key* ...? *body*
: Execute the Tcl script in *body* with the value for each key in *dictionaryVariable* mapped (in a manner similarly to **dict update**) to a variable with the same name. Where one or more *key*s are available, these indicate a chain of nested dictionaries, with the innermost dictionary being the one opened out for the execution of *body*. As with **dict update**, making *dictionaryVariable* unreadable will make the updates to the dictionary be discarded, and this also happens if the contents of

dictionaryVariable are adjusted so that the chain of dictionaries no longer exists. The result of **dict with** is (unless some kind of error occurs) the result of the evaluation of *body*.

The variables are mapped in the scope enclosing the **dict with**; it is recommended that this command only be used in a local scope (**proc**edure, lambda term for **apply**, or method). Because of this, the variables set by **dict with** will continue to exist after the command finishes (unless explicitly **unset**). Note that the mapping of values to variables does not use traces; changes to the *dictionaryVariable*'s contents only happen when *body* terminates.

If the *dictionaryVariable* contains a value that is not a dictionary at the point when the *body* terminates (which can easily happen if the name is the same as any of the keys in dictionary) then an error occurs at that point. This command is thus not recommended for use when the keys in the dictionary are expected to clash with the *dictionaryVariable* name itself. Where the contained key does map to a dictionary, the net effect is to combine that inner dictionary into the outer dictionary; see the **EXAMPLES** below for an illustration of this.

DICTIONARY VALUES

Dictionaries are values that contain an efficient, order-preserving mapping from arbitrary keys to arbitrary values. Each key in the dictionary maps to a single value. They have a textual format that is exactly that of any list with an even number of elements, with each mapping in the dictionary being represented as two items in the list. When a command takes a dictionary and produces a new dictionary based on it (either returning it or writing it back into the variable that the starting dictionary was read from) the new dictionary will have the same order of keys, modulo any deleted keys and with new keys added on to the end. When a string is interpreted as a dictionary and it would otherwise have duplicate keys, only the last value for a particular key is used; the others are ignored, meaning that, "apple banana" and "apple carrot apple banana" are equivalent dictionaries (with different string representations).

Operations that derive a new dictionary from an old one (e.g., updates like **dict set** and **dict unset**) preserve the order of keys in the dictionary. The exceptions to this are for any new keys they add, which are appended to the sequence, and any keys that are removed, which are excised from the order.

EXAMPLES

Basic dictionary usage:

```
# Make a dictionary to map extensions to descriptions
set filetypes [dict create .txt "Text File" .tcl "Tcl File"]

# Add/update the dictionary
dict set filetypes .tcl "Tcl Script"
dict set filetypes .tm  "Tcl Module"
dict set filetypes .gif "GIF Image"
dict set filetypes .png "PNG Image"

# Simple read from the dictionary
set ext ".tcl"
set desc [dict get $filetypes $ext]
puts "$ext is for a $desc"

# Somewhat more complex, with existence test
foreach filename [glob *] {
    set ext [file extension $filename]
    if {[dict exists $filetypes $ext]} {
        puts "$filename is a [dict get $filetypes $ext]"
    }
}
```

Constructing and using nested dictionaries:

```
# Data for one employee
dict set employeeInfo 12345-A forenames "Joe"
dict set employeeInfo 12345-A surname   "Schmoe"
dict set employeeInfo 12345-A street "147 Short Street"
dict set employeeInfo 12345-A city   "Springfield"
dict set employeeInfo 12345-A phone  "555-1234"
# Data for another employee
dict set employeeInfo 98372-J forenames "Anne"
dict set employeeInfo 98372-J surname   "Other"
dict set employeeInfo 98372-J street "32995 Oakdale Way"
dict set employeeInfo 98372-J city   "Springfield"
dict set employeeInfo 98372-J phone  "555-8765"
# The above data probably ought to come from a database...

# Print out some employee info
set i 0
puts "There are [dict size $employeeInfo] employees"
dict for {id info} $employeeInfo {
    puts "Employee #[incr i]: $id"
    dict with info {
        puts "   Name: $forenames $surname"
        puts "   Address: $street, $city"
        puts "   Telephone: $phone"
    }
}
# Another way to iterate and pick out names...
foreach id [dict keys $employeeInfo] {
    puts "Hello, [dict get $employeeInfo $id forenames]!"
}
```

A localizable version of **string toupper**:

```
# Set up the basic C locale
set capital [dict create C [dict create]]
foreach c [split {abcdefghijklmnopqrstuvwxyz} ""] {
    dict set capital C $c [string toupper $c]
}

# English locales can luckily share the "C" locale
dict set capital en [dict get $capital C]
dict set capital en_US [dict get $capital C]
dict set capital en_GB [dict get $capital C]

# ... and so on for other supported languages ...

# Now get the mapping for the current locale and use it.
set upperCaseMap [dict get $capital $env(LANG)]
set upperCase [string map $upperCaseMap $string]
```

Showing the detail of **dict with**:

```
proc sumDictionary {varName} {
    upvar 1 $varName vbl
    foreach key [dict keys $vbl] {
        # Manufacture an entry in the subdictionary
```

```
        dict set vbl $key total 0
        # Add the values and remove the old
        dict with vbl $key {
            set total [expr {$x + $y + $z}]
            unset x y z
        }
    }
    puts "last total was $total, for key $key"
}

set myDict {
    a {x 1 y 2 z 3}
    b {x 6 y 5 z 4}
}

sumDictionary myDict
#   prints:last total was 15, for k ey b

puts "dictionary is now \"$myDict\""
#   prints:dictionary is now "a {total 6} b {total 15}"
```

When **dict with** is used with a key that clashes with the name of the dictionary variable:

```
set foo {foo {a b} bar 2 baz 3}
dict with foo {}
puts $foo
#   prints:a b foo {a b} bar 2 baz 3
```

SEE ALSO

append(n), array(n), foreach(n), incr(n), list(n), lappend(n), lmap(n), set(n)

KEYWORDS

dictionary, create, update, lookup, iterate, filter, map