



**objarray**

**Tcl package for arrays**

# objarray

1 Introduction .....	4
2 Syntax .....	5
2.1 Creation .....	6
2.1.1 new .....	7
2.1.2 new_from_to .....	8
2.1.3 new_n_from_increment .....	9
2.1.4 clone .....	10
2.1.5 range .....	11
2.1.6 new_structured_coordinates .....	12
2.1.7 new_structured_connectivities .....	13
2.2 Access .....	14
2.2.1 length .....	15
2.2.2 type .....	16
2.2.3 set .....	17
2.2.4 get .....	18
2.2.5 replace_value .....	19
2.2.6 search .....	20
2.2.7 get_binary .....	21
2.2.8 foreach .....	22
2.2.9 swap .....	23
2.2.10 permute .....	24
2.3 Operations .....	25
2.3.1 resize .....	26
2.3.2 concat .....	27
2.3.3 insert .....	28
2.3.4 replace .....	29
2.3.5 remove .....	30
2.3.6 interleave .....	31
2.3.7 deinterleave .....	32
2.3.8 renumber .....	33
2.3.9 sort .....	34
2.3.10 reverse .....	35
2.3.11 incr .....	36
2.3.12 scale .....	37
2.3.13 translate .....	38
2.3.14 rotate .....	39
2.3.15 sum .....	40
2.3.16 vector_sum .....	41
2.3.17 minimum .....	42
2.3.18 maximum .....	43
2.3.19 snap .....	44
2.3.20 objarray_map .....	45
2.3.21 objarray_replace .....	46
2.4 Boolean .....	47
2.4.1 intersection .....	48
2.4.2 union .....	49
2.4.3 substraction .....	50
3 What's new .....	51

objarray



objarray 1.18: Tcl package for arrays

## Introduction

objarray is a special type of `Tcl_Obj` to handle efficiently (in memory and cpu) arrays of basic numeric types at Tcl level. This package was developed at CIMNE ([www.cimne.com](http://www.cimne.com)) mainly for GiD (<http://www.gidsimulation.com>), but is implemented as an independent tcl package that can be used by other applications

To use it, it should be loaded load with

```
package require objarray
```

`Tcl_Obj` objects automatically do internal conversion between strings and its special type and vice-versa, this facilitate its use.

The hdf5 plugin uses internally objarrays as bridge between HDF5-like arrays and GiD data.

The vtkobjarray is another specialized Tcl package that act as link between objarrays and vtk-like arrays (e.g. it is used by the GiD-vtk results import/export plugin)

Note: objarray, hdf5 and vtk\_objarray are Tcl packages developed at CIMNE for GiD

## Syntax

```
package require objarray
```

```
objarray length | type | incr | scale | translate | rotate | new_from_to | new | clone | set | get |  
interleave | deinterleave | range | concat | search | sort | reverse | sum | minimum | maximum | insert  
| replace | intersection | union | subtraction | renumber | snap | get_binary | vector_sum ...
```



## new

```
objarray new <type> (-values {<value_1> ... <value_n>}) | (<size> ?<fill_value>?) | (-binary <data>)
```

return a new objarray

<type>: chararray shortarray intarray longarray longlongarray floatarray doublearray

objarray new <type> -values {<value\_1> ... <value\_n>}

<value\_i> must be numeric values compatible with the <type>

In fact it is possible to provide another other objarray instead the list of values to copy its values. It can be of other type, a casting will be applied to convert each value to the destination type.

objarray new <type> <size> ?<fill\_value>?

<size>: integer > 0 to create an array of this size. If <fill\_value> is provided the array items are set to this value.

objarray new <type> -binary <data>

<data> is an arbitrary array of bytes (unsigned char), e.g. to store data of an image.

e.g.

```
set obj [objarray new intarray 3]
-> 1518360 1518610 0 (3 random uninitialized values)
set obj [objarray new intarray 5 1]
-> 1 1 1 1 1 (5 values initialized to 1)
set obj [objarray new intarray -values {6 3 8}]
-> 6 3 8
set obj [objarray new intarray -binary [binary decode base64 AQAAAAIAAADAAAABAAAAA=]]
-> 1 2 3 4
set obj2 [objarray new intarray -binary [objarray get_binary $obj]]
-> 1 2 3 4
```

### new\_from\_to

```
set obj [objarray new_from_to <type> <from> <to>]
```

Allocate an array of type \$type filled with integer-like values increasing from \$from to \$to

<type>: chararray shortarray intarray longarray longlongarray floatarray doublearray

<from> arbitrary integer value

<to> arbitrary integer value greater or equal than <from>

e.g.

```
set obj [objarray new_from_to intarray -3 2]  
-> -3 -2 -1 0 1 2
```



### new\_n\_from\_increment

```
set obj [objarray new_n_from_increment <type> <n> <from> <increment>]
```

Command similar to `new_from_to` but not only for integers and with better control of the amount `n`

Allocate an array of type `$type` with `<n>` values filled starting from `$from` increasing by `<increment>`

`<type>`: chararray shortarray intarray longarray longlongarray floatarray doublearray

`<n>` an integer with the amount of items to be created

`<from>` arbitrary value (of type) to be set as first item

`<increment>` arbitrary value (of type) to increase each item

e.g.

```
set obj [objarray new_n_from_increment floatarray 6 -3.0 1.5]  
-> -3.0 -1.5 0.0 1.5 3.0 4.5
```

## clone

```
set obj2 [objarray clone <obj>]
```

Allocate an array copy of another objarray

## range

```
set obj2 [objarray range <obj> <index_start> <index_end>]
```

Allocate an array with a sub-range of another objarray

e.g.

```
set obj [objarray new_from_to intarray 1 5]
-> 1 2 3 4 5
set obj2 [objarray range $obj 1 end-2]
-> 2 3
```

Indices < 0 or > n are clamped to 0 and n respectively.

## new\_structured\_coordinates

```
objarray new_structured_coordinates {<xs> ?<ys>? ?<zs>?}
```

Allocate an array of type float filled with the values representing a matrix of 1, 2 or 3 dimensions for the node coordinates of an structured cartesian mesh of lines, quadrilaterals or hexahedra

<xs> is a list of real values increasing, representing the x coordinate of the grid nodes

<ys> and <zs> are the same, for the y and z cartesian directions respectively

e.g.

```
set obj [objarray new_structured_coordinates {{2.0 4.0 9.0 15.0} {6.0 12.0}}]  
-> 2.0 6.0 0.0 4.0 6.0 0.0 9.0 6.0 0.0 15.0 6.0 0.0 2.0 12.0 0.0 4.0 12.0 0.0 9.0 12.0 0.0 15.0 12.0 0.0
```

## new\_structured\_connectivities

```
objarray new_structured_connectivities {<nx> ?<ny>? ?<nz>?} ?<offset_nodes>?
```

Allocate an array of type int filled with the values representing a matrix of 1, 2 or 3 dimensions for the element's connectivities of tan structured cartesian mesh of lines, quadrilaterals or hexahedra

<nx> is an integer>0 representing the num of grid nodes in x cartesian direction

<ny> and <nz> are the same, for the y and z cartesian directions respectively

<offset\_nodes> is an optional integer number (0 default), to be increased to node ids generated (default node\_ids start from 0)

e.g.

```
set obj [objarray new_structured_connectivities {4 2}]
-> 0 1 5 4 1 2 6 5 2 3 7 6
```



## length

```
objarray length <obj>
```

Get the length of an existent object

## type

```
objarray type <obj>
```

Returns the type of <obj>:

- chararray shortarray intarray longarray longlongarray floatarray doublearray for objarray objects
- or the Tcl type of the<obj>, for instance "string", "path", "booleanString", etc. or even "".



## set

```
objarray set <obj> {<value_1> ... <value_n>}|<index> <value> | -binary <data>
```

```
objarray set <obj> <index> <value>
```

Set the value of the item \$index (from 0 to n-1) with the value \$value

The value could be the special string "NaN" (not a number) or "Inf" (infinite)

<value> could be a single number or a list of multiple numbers (to set the values of index, index+1, ...)

To set some index the array space must be allocated previously with a size greater of the index.

```
objarray set <obj> {<value_1> ... <value_n>}
```

Set all values from a list {value\_1 ... value\_n}.

In fact instead the list could be another objarray of the same length (can be of other type, a casting will be done for each item).

```
objarray set <obj> -binary <data>
```

copy directly the bytes of <data>, for example to store the data of an image.

e.g.

```
set obj [objarray new intarray 3 0]
# --> 0 0 0
objarray set $obj {6 3 8}
# --> 6 3 8

# in floating point arrays, 'nan' values are also allowed
set oa [ objarray new doublearray 4 nan]
# --> NaN(7fffffffffffffff) NaN(7fffffffffffffff) NaN(7fffffffffffffff) NaN(7fffffffffffffff)
objarray set $oa 0 0.123
# --> 0.123 NaN(7fffffffffffffff) NaN(7fffffffffffffff) NaN(7fffffffffffffff)
objarray set $oa 0 nan
# --> NaN(7fffffffffffffff) NaN(7fffffffffffffff) NaN(7fffffffffffffff) NaN(7fffffffffffffff)
objarray set $oa 0 {0 1 2}
# --> 0 1 2 NaN(7fffffffffffffff)

set oa [ objarray new_from_to intarray 1 10]
# --> 1 2 3 4 5 6 7 8 9 10
set obj [objarray new intarray -values {5 6 7 8}]
# --> 5 6 7 8
objarray set $oa -binary [objarray get_binary $obj]
# --> 5 6 7 8

set oa [ objarray new_from_to doublearray 1 10]
# --> 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
objarray set $oa -binary [objarray get_binary $obj]
# --> 1.27319747483e-313 1.6975966331e-313
```

## get

```
set value [objarray get <obj> <index>]
```

Get the value of the item \$index (from 0 to n-1)

## replace\_value

```
objarray replace_value <obj> <value_old> <value_new>
```

To replace in the whole array <value\_old> by <value\_new>

<value\_old> or <value\_new> could be the special string "NaN" (not a number), but only for floatarray and doublearray cases

e.g.

```
set obj [objarray new floatarray -values {1.0 2.0 3.0 2.0} ]
# -> 1.0 2.0 3.0 2.0
objarray replace_value $obj 2.0 5.0
# -> 1.0 5.0 3.0 5.0
objarray replace_value $obj 3.0 NaN
# -> 1.0 5.0 NaN 5.0
set GID_NO_RESULT -3.4028234663852886e+38
objarray replace_value $obj NaN $GID_NO_RESULT
# -> 1.0 5.0 -3.4028234663852886e+38 5.0
objarray replace_value $obj $GID_NO_RESULT NaN
# -> 1.0 5.0 NaN 5.0
```

## search

```
objarray search ?-sorted? ?-start <index>? <obj> <value>
```

Search the index of an item in the array. returns -1 if it is not found.

(if array is sorted it performs a faster binary search).

-start <index> (index>=0 and < length) to start searching from this index (default 0)

e.g.

```
set obj [objarray new_from_to intarray 2 5]
-> 2 3 4 5
objarray search -sorted $obj 4
-> 2
set obj [objarray new intarray -values {1 2 3 -9999 5 -9999 7 8 -9999}]
-> 1 2 3 -9999 5 -9999 7 8 -9999
objarray search -start 0 $obj -9999
-> 3
objarray search -start 4 $obj -9999
-> 5
objarray search -start 6 $obj -9999
-> 8
objarray search -start 9 $obj -9999
-> -1
```

## get\_binary

```
objarray get_binary <obj>
```

Returns the binary internal representation of the objarray. Useful to avoid conversion from string or to string, or, for instance, to change from integer to chars.

e.g.

```
set obj [objarray new intarray -values {5 6 7 8}]
# --> 5 6 7 8
puts [binary encode base64 [objarray get_binary $obj]]
# --> BQAAAAAYAAAAHAAAAACAAAAA==
set obj2 [objarray new intarray -binary [objarray get_binary $obj]]
# --> 5 6 7 8
set obj3 [objarray new chararray -binary [objarray get_binary $obj]]
# --> 5 0 0 0 6 0 0 0 7 0 0 0 8 0 0 0
```

## foreach

```
objarray foreach <variable_name> <obj> <body>
```

Do a loop over all elements of the objarray, on each iteration set the value of <variable\_name> to the i-value and evaluate the script of the body, similar to a for or a foreach Tcl command.

A break command in the body exit the loop

e.g.

```
set obj [objarray new intarray -values {5 6 7 8}]
-> 5 6 7 8
objarray foreach x $obj {
    puts "x=$x"
}
->
x=5
x=6
x=7
x=8
```

## swap

```
objarray swap <obj> <index_i> <index_j>
```

To swap the values pointed by <index\_i> and <index\_j>

e.g.

```
set obj [objarray new floatarray -values {1.0 2.0 3.0 4.0} ]  
# -> 1.0 2.0 3.0 4.0  
objarray swap $obj 1 2  
# -> 1.0 3.0 2.0 4.0
```

## permute

```
objarray permute <obj> <permutation indices>
```

To permute the values with the order of <permutation indices>, that is a list of unrepeatd indices, in the range [0,n-1]

e.g.

```
package require objarray
set x [objarray new floatarray -values {1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0}]
objarray permute $x {0 1 2 3 4 5 6 7 8 12 13 14 9 10 11}
-> 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 13.0 14.0 15.0 10.0 11.0 12.0
```





## resize

```
objarray resize ?-preserve? <obj> <size>
```

Resize the objarray

<obj> the objarray to be modified

-preserve then the contents will be unchanged up to the lesser of the new and old sizes

<size> the new arrays size

Warning: the values of <obj> itself are modified, without creating another copy of the object

## concat

```
set obj3 [objarray concat <obj> <obj2> ... <objn>]
```

allocate an array joining two or more arrays (of same type)

## insert

```
objarray insert <obj> <index> <obj_to_insert>
```

Inserts another objarray (that must be of the same type) at <index> (from 0 to n)

<obj> the objarray where <obj\_to\_insert> will be inserted

<index> is an integer insertion location. It is possible to use the word 'end' to specify n, to insert at the end of \$obj. A negative index is considered like 0 to concatenate at the beginning, and greater than n is considered like n to append at the end

Warning: the values of <obj> itself are modified, without creating another copy of the object

e.g.

```
set obj [objarray new_from_to intarray 2 5]
-> 2 3 4 5
set obj2 [objarray new_from_to intarray 4 6]
-> 4 5 6
objarray insert $obj end-1 $obj2
-> obj = {2 3 4 5 6 4 5}
```

## replace

```
objarray replace <obj> <index_start> <index_end> <obj_to_insert>
```

Replace the block from index\_start to index\_end by another objarray (that must be of the same type)

<obj> the objarray to be modified

<index\_start> <index\_end> integer indices of the block to be replaced

<obj\_to\_insert> objarray to be inserted

Warning: the values of <obj> itself are modified, without creating another copy of the object

e.g.

```
set obj [objarray new_from_to intarray 1 12]
-> 1 2 3 4 5 6 7 8 9 10 11 12
set obj_insert [objarray new_from_to intarray 20 25]
-> 20 21 22 23 24 25
objarray replace $obj 4 5 $obj_insert
-> obj = {1 2 3 4 20 21 22 23 24 25 7 8 9 10 11 12}
set obj_empty [objarray new intarray 0]
objarray replace $obj 4 9 $obj_empty
-> obj = {1 2 3 4 7 8 9 10 11 12}
```

## remove

```
objarray remove <obj> <index_start> ?<index_end>?
```

Remove the block from index\_start to index\_end

<obj> the objarray to be modified

<index\_start> <index\_end> integer indices of the block to be replaced. If <index\_end> is missing only 1 item is removed

It is similar to use objarray replace inserting an array of zero size

Warning: the values of <obj> itself are modified, without creating another copy of the object

e.g.

```
set obj [objarray new_from_to intarray 1 12]
-> 1 2 3 4 5 6 7 8 9 10 11 12
objarray remove $obj 4
-> obj = {1 2 3 4 6 7 8 9 10 11 12}
objarray remove $obj 4 6
-> obj = {1 2 3 4 9 10 11 12}
```

## interleave

```
objarray interleave <obj> <num_components>
```

Reorder the array of \$num\_components blocks interleaving items of each component

<obj> the objarray to be modified

<num\_components> integer>0 , the amount of blocks

Warning: the values of <obj> itself are modified, without creating another copy of the object

e.g.

```
set obj [objarray concat [objarray new intarray 3 1] [objarray new intarray 3 5]]
-> 1 1 1 5 5 5
objarray interleave $obj 2
->1 5 1 5 1 5
```

## deinterleave

```
objarray deinterleave <obj> <num_components>
```

Reorder the array to separate each component in consecutive blocks. The array size must be multiple of num\_components.

<obj> the objarray to be reordered

<num\_components> integer>0

Warning: the values of <obj> itself are modified, without creating another copy of the object

e.g.

```
set obj {x0 y0 x1 y1 x2 y2}  
set num_components 2  
objarray deinterleave $obj $num_components  
-> x0 x1 x2 y0 y1 y2
```



## renumber

```
objarray renumber <obj> <old_ids> ?<new_ids>?
```

Replace in <obj>, that must be of integer type, the values of the list <old\_ids> by the values of <new\_ids>. <old\_ids> and <new\_ids> must be intarray, and if <new\_ids> is missing it is assumed as an increasing array starting from 0.

Warning: the values of <obj> itself are modified, without creating another copy of the object.

e.g.

```
set obj [objarray new intarray -values {10 15 3 15 20 3}]
-> 10 15 3 15 20 3
set old_ids [objarray new intarray -values {3 10 15 20}]
objarray renumber $obj $old_ids
-> 1 2 0 2 3 0
```

## sort

```
objarray sort ?-unique? <obj>
```

Sort an array increasing.

if -unique is specified duplicated values are removed (the array size decrease)

<obj> the objarray to be sorted

Warning: the values of <obj> itself are modified, without creating another copy of the object

## reverse

```
objarray reverse <obj> ?<blocksize>?
```

Reverse the order of the array.

<obj> the objarray to be reversed

<blocksize> optional integer >=1 and <=num items, to reverse by blocks of this size.

Warning: the values of <obj> itself are modified, without creating another copy of the object

e.g.

```
set obj [objarray new doublearray -values {3.5 2.1 0.3 3.5 2.6 0.3}]
-> 3.5 2.1 0.3 3.5 2.6 0.3
objarray reverse $obj
-> 0.3 2.6 3.5 0.3 2.1 3.5
# reverse again but as blocks of 2 items
objarray reverse $obj 2
-> 2.1 3.5 3.5 0.3 0.3 2.6
```

## incr

```
objarray incr <obj> ?<increment>? ?<index>?
```

Increment in \$increment an item or all array items (default increment is 1)

<obj> the objarray to be modified

<increment> usually an integer value (1 by default). If the array is of type float or double then increment can be also a real number to be added.

<index> optional integer (starting from 0) to increment only the value of the array at this index

Warning: the values of <obj> itself are modified, without creating another copy of the object

## scale

```
objarray scale <obj> <factor> ?<index>?
```

Multiply by \$factor all array items.

<obj> the objarray to be modified

<factor> a real value to multiply all values of the array

<index> optional integer (starting from 0) to multiply only this component of the array

The array doesn't change its type, e.g. an array of integers multiplied by a real factor will truncate the result again as integer.

Warning: the values of <obj> itself are modified, without creating another copy of the object

e.g.

```
set obj [objarray new doublearray -values {3.5 2.1 0.3 3.5 2.6 0.3}]
-> 3.5 2.1 0.3 3.5 2.6 0.3
objarray scale $obj 2.5
-> 8.75 5.25 0.75 8.75 6.5 0.75
```

## translate

```
objarray translate <obj> <dx dy dz>
```

For arrays with x y z coordinates, sum a displacement dx dy dz to all array items (the object length must be multiple of 3)

<obj> the objarray with coordinates to be translated

<dx dy dz> vector of 3 numbers to define the translation

The array doesn't change its type, e.g. an array of integers increased by a real value will truncate the result again as integer.

Warning: the values of <obj> itself are modified, without creating another copy of the object

e.g.

```
set obj [objarray new doublearray -values {3.5 2.1 0.3 3.5 2.6 0.3}]
-> 3.5 2.1 0.3 3.5 2.6 0.3
objarray translate $obj {0.5 0.4 0.0}
-> 4.0 2.5 0.3 4.0 3.0 0.3
```

## rotate

```
objarray rotate <obj> <r11 r12 r13 r21 r22 r23 r31 r32 r33>
```

For arrays with x y z coordinates, multiply a rotation matrix 3x3 for all array items (the object length must be multiple of 3)

<obj> the objarray with coordinates that will be rotated

<r11 r12 r13 r21 r22 r23 r31 r32 r33> a vector of 9 real numbers to define the rotation

The array doesn't change its type, e.g. an array of integers increased by a real value will truncate the result again as integer.

Warning: the values of <obj> itself are modified, without creating another copy of the object.

e.g.

```
set obj [objarray new doublearray -values {3.5 2.1 0.3 3.5 2.6 0.3}]
-> 3.5 2.1 0.3 3.5 2.6 0.3
objarray rotate $obj {0.0 -1.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0}
-> -2.1 3.5 0.3 -2.6 3.5 0.3
```

## sum

```
set total [objarray sum <obj>]
```

Return a single value with the summatory reduction of all array items

<obj> the objarray to sum all its components

e.g.

```
set obj [objarray new doublearray -values {3.5 2.1 0.3 3.5 2.6 0.3}]
-> 3.5 2.1 0.3 3.5 2.6 0.3
objarray sum $obj
-> 12.299999999999999
```



## vector\_sum

```
objarray vector_sum ?-type <type>? <obj_1> <obj_2> ... <obj_n>
```

Return a vector sum of other 'n' vectors

if -type is specified all the arguments are converted from any representation (including text or list) to this type

<type>: chararray shortarray intarray longarray longlongarray floatarray doublearray

the default type is doublearray, but if first argument, if it is some kind of objarray try to preserve it and is used as type.

<obj\_1> ... <obj\_n> are objarrays (or can be converted to objarrays of <type>) all with the same length

e.g.

```
set obj_1 [objarray new doublearray -values {3.5 2.1}]
set obj_2 [objarray new doublearray -values {0.3 3.5}]
set obj_3 [objarray new doublearray -values {2.6 0.4}]
set obj_sum [objarray vector_sum $obj_1 $obj_2 $obj_3]
->6.4 6.0
```

## minimum

```
set min [objarray minimum <obj> ?-index?]
```

Return the minimum value of all array items

<obj> the objarray to get its minimum value

if -index is specified, then instead the minimum value it return the index (starting from 0) of this value.

## maximum

```
set max [objarray maximum <obj> ?-index?]
```

Return the maximum value of all array items

<obj> the objarray to get its maximum value

if -index is specified, then instead the maximum value it return the index (starting from 0) of this value.

## snap

```
objarray snap <obj> <value> <tolerance> <index_start> <index_increment>
```

To modify the values that are closed to <value>, within <tolerance>. <index\_start> and <index\_increment> allow to jump some indices

<obj> an objarray with x y z values to be modified

<value> a real number

<tolerance> a real number>0

<index\_start> integer>=0 (usually 0)

<index\_increment> to jump indices (usually 1)

A scenario could be to set z=0 in an obj with values {x y z ... x y z}, to loop only on z values the index\_start=2 and the index\_increment=3

e.g.

```
set obj [objarray new doublearray -values {10.0 0.001 0.8 1.5 -5.02 0.0000004}]
objarray snap $obj 0.0 1e-4 2 3
-> 10.0 0.001 0.8 1.5 -5.02 0.0
```

## objarray\_map

```
set obj [ objarray_map dst_type {list_arguments body_proc} list_of_arrays
```

Creates a new objarray of type `dst_type` whose `i`-element is created by evaluating the lambda function with arguments `<list_arguments>` and body `<body_proc>` on the `i`-element of each array in `<list_of_arrays>`.

e.g.

```
# example:
set oa1 [ objarray new_from_to intarray 2 5]
set oa2 [ objarray new_from_to intarray 6 9]
objarray_map floatarray {{x y} {expr $x + $y}} [ list $oa1 $oa2]
# --> 8.0 10.0 12.0 14.0
```

## objarray\_replace

```
# objarray_replace objarray_name value_to_search value_to_replace
```

In the objarray with name `objarray_name` each element with `value_to_search` is replaced with `value_to_replace`. No new objarray is created.

e.g.

```
# example:
set oa [ objarray new_from_to floatarray 0 10]
# --> 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0
set oa [ objarray set $oa 4 nan]
# --> 0.0 1.0 2.0 3.0 NaN(7ffffe00000000) 5.0 6.0 7.0 8.0 9.0 10.0
objarray_replace oa NaN(7ffffe00000000) 12345
# --> 0.0 1.0 2.0 3.0 12345.0 5.0 6.0 7.0 8.0 9.0 10.0
```

## Boolean

Boolean operations

## intersection

```
objarray intersection ?-sorted? <obj1> <obj2>
```

Return a new objarray with the items of both objects (boolean and)

<obj1> and <obj2> are the objarrays to get its intersection.

if -sorted is set obj1 and obj2 are expected to be in ascending order (to avoid an extra effort to sort them)

The result is also an ordered array.

e.g

```
set obj1 [objarray new_from_to intarray 2 5]
-> 2 3 4 5
set obj2 [objarray new_from_to intarray 3 8]
-> 3 4 5 6 7 8
set obj3 [objarray intersection -sorted $obj1 $obj2]
-> 3 4 5
```



## union

```
objarray union ?-sorted? <obj1> <obj2>
```

Return a new objarray with the items without repetition that are at least in some of both objects (boolean or)

<obj1> and <obj2> are the objarrays to get its union.

if -sorted is set obj1 and obj2 are expected to be in ascending order (to avoid an extra effort to sort them)

The result is also an ordered array.

e.g

```
set obj1 [objarray new_from_to intarray 2 5]
-> 2 3 4 5
set obj2 [objarray new_from_to intarray 3 8]
-> 3 4 5 6 7 8
set obj3 [objarray union -sorted $obj1 $obj2]
-> 2 3 4 5 6 7 8
```

## subtraction

```
objarray subtraction ?-sorted?<obj1> <obj2>
```

Return a new objarray removing from obj1 the items that are in obj2

<obj1> is the objarrays where <obj2> will be subtracted.

if -sorted is set obj1 and obj2 are expected to be in ascending order (to avoid an extra effort to sort them)

The result is also an ordered array.

## What's new

- v 1.18:
  - objarray set now allow to set multiple <n> values of index, index+1, index+<n>
  - new command objarray swap
  - new command objarray permute
  - new command objarray new\_structured\_connectivities
  - new command objarray new\_structured\_coordinates
  - objarray reverse now allow <blocksize> to reverse considering as blocks of this size
  - new command objarray new\_n\_from\_increment to create n items starting from a value and increment
  - objarray search now allow -start <index> to start finding from this index
  - new command objarray resize
  - new command objarray remove
- v 1.17: objarray range allow index <0 or >n then become 0 and n, acting like Tcl standard lrange command.
- v 1.16: objarray replace\_value
- v 1.15:
  - objarray set allow 'nan' special value
  - -binary <data>
  - proc objarray\_map { dst\_objarray\_type lambda lst\_arrays }
- v 1.14: foreach to simplify and to loop faster over all items of the array.
- v 1.13: vector\_sum to return a vector sum of other 'n' vectors (but it is not a reduction sum).
- v 1.12:
  - new -binary <data> option when creating objarrays
  - added new 'get\_binary' option to get the byte representation of the objarray.
- v 1.11:
  - translate and rotate, for arrays representing xyz coordinates.
  - Efficiency enhanced of replace command.
- v 1.10:
  - snap to force values close (with tolerance) to a target.
  - minimum/maximum -index sub-option.
- v 1.9: reverse
- v 1.8:
  - incr <index> sub-option.
  - Some commands parallelized with OpenMP
- v 1.7: scale, minimum, maximum, renumber
- v 1.5:
  - boolean intersection, union and subtraction.
  - objarray new and objarray set modified to allow set all values from a list of values.
- v 1.4: search
- v 1.3: insert and replace