The Array Operations Reference Manual

Array operations for array-like data structures, version 1.0.0

Steve Nunez
Tamas K. Papp <tkpapp@gmail.com>

This manual was generated automatically by Declt 4.0b2.

Table of Contents

Copying

This program is distributed under the terms of the Microsoft Public License.

1 Systems

The main system appears first, followed by any subsystem dependency.

1.1 array-operations

Array operations library for Common Lisp

Long Name
Array operations for array-like data structures

Maintainer
Steve Nunez

Author Tamas K. Papp <tkpapp@gmail.com>

Home Page
https://lisp-stat.dev/docs/manuals/array-operations

Bug Tracker
https://github.com/Lisp-Stat/array-operations/issues

License MS-PL

Long Description
The array-operations system is a collection of functions and macros for manipulating Common Lisp arrays and performing numerical calculations with them.

Array-operations is a 'generic' way of operating on array like data structures using a syntax that is natural for Common Lisp. Several aops functions have been implemented for data-frame. For those that haven't, you can transform arrays to data frames using the df:matrix-df function, and a data-frame to an array using df:as-array. This make it convenient to work with the data sets using either system.

Version 1.0.0

Dependencies
- let-plus (system).
- [array-operations/all], page 3 (system).

Source [array-operations.asd], page 9.

1.2 array-operations/all

Long Name
Array operations for array-like data structures

Maintainer
Steve Nunez

Author Tamas K. Papp <tkpapp@gmail.com>

Home Page
https://lisp-stat.dev/docs/manuals/array-operations

Bug Tracker
https://github.com/Lisp-Stat/array-operations/issues

License MS-PL

Dependencies

- [array-operations/generic], page 4 (system).
- [array-operations/reducing], page 4 (system).
- [array-operations/matrices], page 5 (system).
- [array-operations/creating], page 5 (system).
- [array-operations/indexing], page 6 (system).
- [array-operations/displacing], page 6 (system).
- [array-operations/transforming], page 7 (system).
- [array-operations/stacking], page 7 (system).

Source [array-operations.asd], page 9.

1.3 array-operations/generic

Long Name
Array operations for array-like data structures

Maintainer
Steve Nunez

Author Tamas K. Papp <tkpapp@gmail.com>

Home Page
https://lisp-stat.dev/docs/manuals/array-operations

Bug Tracker
https://github.com/Lisp-Stat/array-operations/issues

License MS-PL

Dependency
let-plus (system).

Source [array-operations.asd], page 9.

1.4 array-operations/reducing

Long Name
Array operations for array-like data structures

Maintainer
Steve Nunez

Author Tamas K. Papp <tkpapp@gmail.com>

Home Page
https://lisp-stat.dev/docs/manuals/array-operations

Bug Tracker
https://github.com/Lisp-Stat/array-operations/issues

License MS-PL

Source [array-operations.asd], page 9.

1.5 array-operations/matrices

Long Name
Array operations for array-like data structures

Maintainer
Steve Nunez

Author Tamas K. Papp <tkpapp@gmail.com>

Home Page
https://lisp-stat.dev/docs/manuals/array-operations

Bug Tracker
https://github.com/Lisp-Stat/array-operations/issues

License MS-PL

Dependencies
- [array-operations/generic], page 4 (system).
- alexandria (system).

Source [array-operations.asd], page 9.

1.6 array-operations/creating

Long Name
Array operations for array-like data structures

Maintainer
Steve Nunez

Author Tamas K. Papp <tkpapp@gmail.com>

Home Page
https://lisp-stat.dev/docs/manuals/array-operations

Bug Tracker
https://github.com/Lisp-Stat/array-operations/issues

License MS-PL

Dependencies
- [array-operations/generic], page 4 (system).
- [array-operations/utilities], page 5 (system).

Source [array-operations.asd], page 9.

1.7 array-operations/utilities

Long Name
Array operations for array-like data structures

Maintainer
Steve Nunez

Author Tamas K. Papp <tkpapp@gmail.com>

Home Page
https://lisp-stat.dev/docs/manuals/array-operations

Bug Tracker
https://github.com/Lisp-Stat/array-operations/issues

License MS-PL

Dependencies

- [array-operations/generic], page 4 (system).
- alexandria (system).

Source [array-operations.asd], page 9.

1.8 array-operations/indexing

Long Name
Array operations for array-like data structures

Maintainer
Steve Nunez

Author Tamas K. Papp <tkpapp@gmail.com>

Home Page
https://lisp-stat.dev/docs/manuals/array-operations

Bug Tracker
https://github.com/Lisp-Stat/array-operations/issues

License MS-PL

Dependencies

- [array-operations/generic], page 4 (system).
- [array-operations/utilities], page 5 (system).

Source [array-operations.asd], page 9.

1.9 array-operations/displacing

Long Name
Array operations for array-like data structures

Maintainer
Steve Nunez

Author Tamas K. Papp <tkpapp@gmail.com>

Home Page
https://lisp-stat.dev/docs/manuals/array-operations

Bug Tracker
https://github.com/Lisp-Stat/array-operations/issues

License MS-PL

Dependencies

- [array-operations/generic], page 4 (system).
- [array-operations/utilities], page 5 (system).
- alexandria (system).

Source [array-operations.asd], page 9.

1.10 array-operations/transforming

Long Name
Array operations for array-like data structures

Maintainer
Steve Nunez

Author Tamas K. Papp <tkpapp@gmail.com>

Home Page
https://lisp-stat.dev/docs/manuals/array-operations

Bug Tracker
https://github.com/Lisp-Stat/array-operations/issues

License MS-PL

Dependencies
- [array-operations/generic], page 4 (system).
- [array-operations/utilities], page 5 (system).
- [array-operations/displacing], page 6 (system).
- alexandria (system).

Source [array-operations.asd], page 9.

1.11 array-operations/stacking

Long Name
Array operations for array-like data structures

Maintainer
Steve Nunez

Author Tamas K. Papp <tkpapp@gmail.com>

Home Page
https://lisp-stat.dev/docs/manuals/array-operations

Bug Tracker
https://github.com/Lisp-Stat/array-operations/issues

License MS-PL

Dependencies
- [array-operations/generic], page 4 (system).
- [array-operations/utilities], page 5 (system).
- [array-operations/displacing], page 6 (system).
- [array-operations/transforming], page 7 (system).
- alexandria (system).

Source [array-operations.asd], page 9.

2 Files

Files are sorted by type and then listed depth-first from the systems components trees.

2.1 Lisp

2.1.1 array-operations/array-operations.asd

Source [array-operations.asd], page 9.

Parent Component
[array-operations], page 3 (system).

ASDF Systems
- [array-operations], page 3.
- [array-operations/all], page 3.
- [array-operations/generic], page 4.
- [array-operations/reducing], page 4.
- [array-operations/matrices], page 5.
- [array-operations/creating], page 5.
- [array-operations/utilities], page 5.
- [array-operations/indexing], page 6.
- [array-operations/displacing], page 6.
- [array-operations/transforming], page 7.
- [array-operations/stacking], page 7.

2.1.2 array-operations/all/file-type.lisp

Source [array-operations.asd], page 9.

Parent Component
[array-operations/all], page 3 (system).

Packages [array-operations/all], page 16.

2.1.3 array-operations/generic/file-type.lisp

Source [array-operations.asd], page 9.

Parent Component
[array-operations/generic], page 4 (system).

Packages [array-operations/generic], page 15.

Public Interface
- [&dims], page 21 (macro).
- [as-array], page 37 (generic function).
- [dim], page 37 (generic function).
- [dims], page 37 (generic function).
- [element-type], page 37 (generic function).
- [ncol], page 38 (generic function).
- [nrow], page 38 (generic function).
- [rank], page 38 (generic function).
- [size], page 38 (generic function).

2.1.4 array-operations/reducing/file-type.lisp

Source [array-operations.asd], page 9.

Parent Component
[array-operations/reducing], page 4 (system).

Packages [array-operations/reducing], page 18.

Public Interface

- [argmax], page 26 (function).
- [argmin], page 26 (function).
- [best], page 26 (function).
- [most], page 30 (function).
- [vectorize-reduce], page 25 (macro).

2.1.5 array-operations/matrices/file-type.lisp

Source [array-operations.asd], page 9.

Parent Component
[array-operations/matrices], page 5 (system).

Packages [array-operations/matrices], page 18.

Public Interface

- [array-matrix], page 40 (type).
- [matrixp], page 30 (function).
- [square-matrix-p], page 34 (function).

2.1.6 array-operations/creating/file-type.lisp

Source [array-operations.asd], page 9.

Parent Component
[array-operations/creating], page 5 (system).

Packages [array-operations/creating], page 16.

Public Interface

- [fill!], page 28 (function).
- [generate], page 28 (function).
- [generate*], page 28 (function).
- [linspace], page 29 (function).
- [linspace!], page 29 (function).
- [linspace*], page 29 (function).
- [ones], page 30 (function).
- [ones!], page 31 (function).
- [ones*], page 31 (function).
- [rand], page 32 (function).
- [rand!], page 32 (function).
- [rand*], page 32 (function).
- [randn], page 32 (function).
- [randn!], page 33 (function).

- [randn*], page 33 (function).
- [similar-array], page 34 (function).
- [zeros], page 36 (function).
- [zeros!], page 36 (function).
- [zeros*], page 36 (function).

2.1.7 array-operations/utilities/file-type.lisp

Source [array-operations.asd], page 9.

Parent Component
[array-operations/utilities], page 5 (system).

Packages [array-operations/utilities], page 18.

Public Interface

- [ensure-dimensions], page 27 (function).
- [multf], page 22 (macro).
- [nested-loop], page 22 (macro).
- [product], page 31 (function).
- [same-dimensions-p], page 34 (function).
- [walk-subscripts], page 25 (macro).
- [walk-subscripts-list], page 25 (macro).

2.1.8 array-operations/indexing/file-type.lisp

Source [array-operations.asd], page 9.

Parent Component
[array-operations/indexing], page 6 (system).

Packages [array-operations/indexing], page 15.

Public Interface

- [each-index], page 21 (macro).
- [each-index!], page 21 (macro).
- [each-index*], page 22 (macro).
- [reduce-index], page 23 (macro).
- [sum-index], page 23 (macro).

Internals

- [find-array-dimensions], page 41 (function).
- [foreach], page 40 (macro).

2.1.9 array-operations/displacing/file-type.lisp

Source [array-operations.asd], page 9.

Parent Component
[array-operations/displacing], page 6 (system).

Packages [array-operations/displacing], page 17.

Public Interface

- [combine], page 26 (function).
- [copy-into], page 27 (function).

- [displace], page 27 (function).
- [fill-in-dimensions], page 28 (function).
- [flatten], page 28 (function).
- [partition], page 31 (function).
- [(setf partition)], page 31 (function).
- [reshape], page 33 (function).
- [reshape-col], page 33 (function).
- [reshape-row], page 34 (function).
- [split], page 34 (function).
- [sub], page 36 (function).
- [(setf sub)], page 36 (function).
- [subvec], page 36 (function).
- [(setf subvec)], page 36 (function).

Internals [sub-location%], page 41 (function).

2.1.10 array-operations/transforming/file-type.lisp

Source [array-operations.asd], page 9.

Parent Component
[array-operations/transforming], page 7 (system).

Packages [array-operations/transforming], page 19.

Public Interface

- [check-permutation], page 26 (function).
- [coercing], page 26 (function).
- [complement-permutation], page 27 (function).
- [complete-permutation], page 27 (function).
- [each], page 27 (function).
- [each*], page 27 (function).
- [identity-permutation-p], page 28 (function).
- [invert-permutation], page 29 (function).
- [margin], page 30 (function).
- [margin*], page 30 (function).
- [outer], page 31 (function).
- [outer*], page 31 (function).
- [permutation-incompatible-rank], page 39 (condition).
- [permutation-invalid-index], page 39 (condition).
- [permutation-repeated-index], page 40 (condition).
- [permute], page 31 (function).
- [recycle], page 33 (function).
- [turn], page 36 (function).
- [vectorize], page 24 (macro).
- [vectorize!], page 24 (macro).
- [vectorize*], page 25 (macro).

Internals

- [array-index-row-major], page 40 (function).
- [array-rank-element], page 41 (type).
- [permutation-flags], page 41 (function).

2.1.11 array-operations/stacking/file-type.lisp

Source [array-operations.asd], page 9.

Parent Component
[array-operations/stacking], page 7 (system).

Packages [array-operations/stacking], page 20.

Public Interface

- [copy-row-major-block], page 27 (function).
- [stack], page 34 (function).
- [stack*], page 34 (function).
- [stack-cols], page 35 (function).
- [stack-cols*], page 35 (function).
- [stack-cols-copy], page 38 (generic function).
- [stack-rows], page 35 (function).
- [stack-rows*], page 35 (function).
- [stack-rows-copy], page 39 (generic function).

Internals [stack*0], page 41 (function).

3 Packages

Packages are listed by definition order.

3.1 array-operations/indexing

Macros for operating over indexes of arrays.

Source [file-type.lisp], page 11.

Use List

- [array-operations/generic], page 15.
- [array-operations/utilities], page 18.
- common-lisp.

Used By List

[array-operations/all], page 16.

Public Interface

- [each-index], page 21 (macro).
- [each-index!], page 21 (macro).
- [each-index*], page 22 (macro).
- [reduce-index], page 23 (macro).
- [sum-index], page 23 (macro).

Internals

- [find-array-dimensions], page 41 (function).
- [foreach], page 40 (macro).

3.2 array-operations/generic

Generic functions for elementary array operations, with methods on 'array. Enables new methods to be defined to enable treating other data structures as arrays.

Source [file-type.lisp], page 9.

Use List

- common-lisp.
- let-plus.

Used By List

- [array-operations/all], page 16.
- [array-operations/creating], page 16.
- [array-operations/displacing], page 17.
- [array-operations/indexing], page 15.
- [array-operations/matrices], page 18.
- [array-operations/stacking], page 20.
- [array-operations/transforming], page 19.
- [array-operations/utilities], page 18.

Public Interface

- [&dims], page 21 (macro).
- [as-array], page 37 (generic function).

- [dim], page 37 (generic function).
- [dims], page 37 (generic function).
- [element-type], page 37 (generic function).
- [ncol], page 38 (generic function).
- [nrow], page 38 (generic function).
- [rank], page 38 (generic function).
- [size], page 38 (generic function).

3.3 array-operations/all

This top level package re-exports the individual packages: generic, reducing, matrices, creating, indexing, displacing, transforming and stacking. It does not export utilities. The reason for this structure is the use of the ASDF package-inferred-system, where each file is its own package. None of Papp's other libraries use this, and it seems to have been added after he abandoned the library.

Source [file-type.lisp], page 9.

Nicknames

- aops
- array-operations

Use List

- [array-operations/creating], page 16.
- [array-operations/displacing], page 17.
- [array-operations/generic], page 15.
- [array-operations/indexing], page 15.
- [array-operations/matrices], page 18.
- [array-operations/reducing], page 18.
- [array-operations/stacking], page 20.
- [array-operations/transforming], page 19.

Used By List

lisp-stat.

3.4 array-operations/creating

Functions for creating arrays or data frames filled with various values.

Source [file-type.lisp], page 10.

Use List

- [array-operations/generic], page 15.
- [array-operations/utilities], page 18.
- common-lisp.

Used By List

[array-operations/all], page 16.

Public Interface

- [fill!], page 28 (function).
- [generate], page 28 (function).
- [generate*], page 28 (function).

- [linspace], page 29 (function).
- [linspace!], page 29 (function).
- [linspace*], page 29 (function).
- [make-array-like], page 29 (function).
- [ones], page 30 (function).
- [ones!], page 31 (function).
- [ones*], page 31 (function).
- [rand], page 32 (function).
- [rand!], page 32 (function).
- [rand*], page 32 (function).
- [randn], page 32 (function).
- [randn!], page 33 (function).
- [randn*], page 33 (function).
- [similar-array], page 34 (function).
- [zeros], page 36 (function).
- [zeros!], page 36 (function).
- [zeros*], page 36 (function).

3.5 array-operations/displacing

Functions that return arrays displaced in various ways from another array.

Source [file-type.lisp], page 11.

Use List

- [array-operations/generic], page 15.
- [array-operations/utilities], page 18.
- common-lisp.

Used By List

- [array-operations/all], page 16.
- [array-operations/stacking], page 20.
- [array-operations/transforming], page 19.

Public Interface

- [combine], page 26 (function).
- [copy-into], page 27 (function).
- [displace], page 27 (function).
- [fill-in-dimensions], page 28 (function).
- [flatten], page 28 (function).
- [partition], page 31 (function).
- [(setf partition)], page 31 (function).
- [reshape], page 33 (function).
- [reshape-col], page 33 (function).
- [reshape-row], page 34 (function).
- [split], page 34 (function).
- [sub], page 36 (function).

- [(setf sub)], page 36 (function).
- [subvec], page 36 (function).
- [(setf subvec)], page 36 (function).

Internals [sub-location%], page 41 (function).

3.6 array-operations/reducing

Functions for reducing arrays, or performing reducing like operations over the elements of an array.

Source [file-type.lisp], page 10.

Use List common-lisp.

Used By List

[array-operations/all], page 16.

Public Interface

- [argmax], page 26 (function).
- [argmin], page 26 (function).
- [best], page 26 (function).
- [most], page 30 (function).
- [vectorize-reduce], page 25 (macro).

3.7 array-operations/matrices

Functions for representing matrices as 2D arrays. A matrix is a two-dimensional array often used for linear algebra. See also the matrix functions in NUM-UTILS, which should be migrated to AOPS.

Source [file-type.lisp], page 10.

Use List

- [array-operations/generic], page 15.
- common-lisp.

Used By List

[array-operations/all], page 16.

Public Interface

- [array-matrix], page 40 (type).
- [matrix?], page 30 (function).
- [matrixp], page 30 (function).
- [square-matrix-p], page 34 (function).
- [square-matrix?], page 34 (function).

3.8 array-operations/utilities

Source [file-type.lisp], page 11.

Use List

- [array-operations/generic], page 15.
- common-lisp.

Used By List

- [array-operations/creating], page 16.
- [array-operations/displacing], page 17.
- [array-operations/indexing], page 15.
- [array-operations/stacking], page 20.
- [array-operations/transforming], page 19.

Public Interface

- [ensure-dimensions], page 27 (function).
- [multf], page 22 (macro).
- [nested-loop], page 22 (macro).
- [product], page 31 (function).
- [same-dimensions-p], page 34 (function).
- [walk-subscripts], page 25 (macro).
- [walk-subscripts-list], page 25 (macro).

3.9 array-operations/transforming

Functions for transforming arrays in various ways.

Source [file-type.lisp], page 12.

Use List

- [array-operations/displacing], page 17.
- [array-operations/generic], page 15.
- [array-operations/utilities], page 18.
- common-lisp.

Used By List

[array-operations/all], page 16.

Public Interface

- [check-permutation], page 26 (function).
- [coercing], page 26 (function).
- [complement-permutation], page 27 (function).
- [complete-permutation], page 27 (function).
- [each], page 27 (function).
- [each*], page 27 (function).
- [identity-permutation-p], page 28 (function).
- [identity-permutation?], page 29 (function).
- [invert-permutation], page 29 (function).
- [margin], page 30 (function).
- [margin*], page 30 (function).
- [outer], page 31 (function).
- [outer*], page 31 (function).
- [permutation-incompatible-rank], page 39 (condition).
- [permutation-invalid-index], page 39 (condition).
- [permutation-repeated-index], page 40 (condition).

- [permute], page 31 (function).
- [recycle], page 33 (function).
- [turn], page 36 (function).
- [vectorize], page 24 (macro).
- [vectorize!], page 24 (macro).
- [vectorize*], page 25 (macro).

Internals

- [array-index-row-major], page 40 (function).
- [array-rank-element], page 41 (type).
- [permutation-flags], page 41 (function).

3.10 array-operations/stacking

Functions for composing arrays into new arrays, by stacking.

One may think of stacking blocks as the guiding metaphor.

For example, stack two row vectors to yield a 2x2 matrix:

(stack-rows #(1 2) #(3 4)) -> #2A((1 2)
(3 4))

Source [file-type.lisp], page 13.

Use List

- [array-operations/displacing], page 17.
- [array-operations/generic], page 15.
- [array-operations/utilities], page 18.
- common-lisp.

Used By List

[array-operations/all], page 16.

Public Interface

- [copy-row-major-block], page 27 (function).
- [stack], page 34 (function).
- [stack*], page 34 (function).
- [stack-cols], page 35 (function).
- [stack-cols*], page 35 (function).
- [stack-cols-copy], page 38 (generic function).
- [stack-rows], page 35 (function).
- [stack-rows*], page 35 (function).
- [stack-rows-copy], page 39 (generic function).

Internals [stack*0], page 41 (function).

4 Definitions

Definitions are sorted by export status, category, package, and then by lexicographic order.

4.1 Public Interface

4.1.1 Macros

`&dims` (**&rest** *dimensions*) [Macro]
Dimensions of array-like object.

Package [`array-operations/generic`], page 15.

Source [`file-type.lisp`], page 9.

`each-index` (*index* **&body** *body*) [Macro]
Given one or more symbols INDEX, walks the BODY expression to determine the index ranges by looking for AREF and ROW-MAJOR-AREF calls.

Transpose of 2D array A

(each-index (i j)
(aref A j i))

Diagonal of a square 2D array

(each-index i (aref A i i))

Turn a 2D array into an array of arrays

(each-index i
(each-index j
(aref A i j)))

Matrix-vector product:

(each-index i
(sum-index j
(* (aref A i j) (aref x j))))

Package [`array-operations/indexing`], page 15.

Source [`file-type.lisp`], page 11.

`each-index!` (*array index* **&body** *body*) [Macro]
Sets elements of the given ARRAY to values of the BODY, evaluated at array indices INDEX

Note: This has the same semantics as each-index and each-index*,
but the INDEX ranges are taken from the ARRAY dimensions, not
a code walker.

Package [`array-operations/indexing`], page 15.

Source [`file-type.lisp`], page 11.

`each-index*` (*element-type index* **&body** *body*) [Macro]
Given one or more symbols INDEX, creates an array with ELEMENT-TYPE, then iterates over the index ranges with the innermost loop using the last index. Each iteration evaluates BODY, and sets the array element.

To find the range of the indices, walks the BODY expression
to determine the index ranges by looking for
AREF and ROW-MAJOR-AREF calls.

Transpose of 2D array A

(each-index* t (i j)
(aref A j i))

Diagonal of a square 2D array

(each-index* t i (aref A i i))

Turn a 2D array into an array of arrays

(each-index* t i
(each-index* t j
(aref A i j)))

Outer product of two 1D arrays to create a 2D array

(each-index* t (i j)
(* (aref x i) (aref y j)))

Matrix-vector product:

(each-index* t i
(sum-index j
(* (aref A i j) (aref x j))))

Package [`array-operations/indexing`], page 15.

Source [`file-type.lisp`], page 11.

`multf` (*place* **&rest** *values*) [Macro]
Multiply by the arguments

Package [`array-operations/utilities`], page 18.

Source [`file-type.lisp`], page 11.

`nested-loop` (*syms dimensions* **&body** *body*) [Macro]
Iterates over a multidimensional range of indices.

SYMS must be a list of symbols, with the first symbol corresponding to the outermost loop.

DIMENSIONS will be evaluated, and must be a list of dimension sizes, of the same length as SYMS.

Example:
(nested-loop (i j) '(10 20) (format t '~a ~a~%' i j))
expands to:

; Check dimensions
(destructuring-bind (g1 g2) '(10 20)
(loop for i from 0 below g1 do
(loop for j from 0 below g2 do
(format t '~a ~a~%' i j))))

with some additional type and dimension checks.

Package [array-operations/utilities], page 18.

Source [file-type.lisp], page 11.

`reduce-index` (*function index* **&body** *body*) [Macro]

Reduction over one or more INDEX symbols in an array expression.
The range of these symbols is determined by walking the tree
for AREF and ROW-MAJOR-AREF calls.

Example:

(defparameter A #2A((1 2) (3 4)))

(reduce-index #'+ i (row-major-aref A i)) ; Sum all elements (sum-index) => 10

(reduce-index #'* (i j) (aref A i j)) ; Multiply all elements
=> 24

(reduce-index #'max i (row-major-aref A i)) ; Maxmum value
=> 4

Package [array-operations/indexing], page 15.

Source [file-type.lisp], page 11.

`sum-index` (*index* **&body** *body*) [Macro]

Sums over one or more INDEX symbols in an array expression. The range of these symbols is determined by walking the tree for AREF and ROW-MAJOR-AREF calls.

Example:

(defparameter A #2A((1 2) (3 4)))

(sum-index i (row-major-aref A i)) ; Sum all elements => 10

```
(sum-index (i j) (aref A i j)) ; Sum all elements
=> 10

(sum-index i (aref A i i)) ; Trace of array
=> 5
```

Package [array-operations/indexing], page 15.

Source [file-type.lisp], page 11.

`vectorize` (*variables* **&body** *body*) [Macro]

Makes a new array of type ELEMENT-TYPE, containing the result of an array expression. All input and outputs have the same shape, and BODY is evaluated for each index

VARIABLES must be a list of symbols bound to arrays.
Each array must have the same dimensions. These are
checked at compile and run-time respectively.

```
(let ((a #2A((1 2) (3 4))))
(vectorize (a) (+ a 1)))
-> #2A((2 3) (4 5))

(let ((a #(1 2 3))
(b #(4 5 6)))
(vectorize (a b) (+ a (* b 2))))
-> #(9 12 15)
```

Package [array-operations/transforming], page 19.

Source [file-type.lisp], page 12.

`vectorize!` (*result variables* **&body** *body*) [Macro]

Fills an array RESULT with the result of an array expression. All input and outputs have the same shape, and BODY is evaluated for each index

VARIABLES must be a list of symbols bound to arrays. Each array must have the same dimensions. These are checked at compile and run-time respectively.

```
(let ((a #2A((1 2) (3 4)))
(b (make-array '(2 2))))
(vectorize! b (a) (+ a 1)))
-> #2A((2 3) (4 5))

(let ((a #(1 2 3))
(b #(4 5 6)))
(vectorize! b (a b) (+ a (* b 2))))
-> #(9 12 15)
```

Package [array-operations/transforming], page 19.

Source [file-type.lisp], page 12.

`vectorize*` (*element-type variables* **&body** *body*) [Macro]

Makes a new array of type ELEMENT-TYPE, containing the result of an array expression. All input and outputs have the same shape, and BODY is evaluated for each index

VARIABLES must be a list of symbols bound to arrays.
Each array must have the same dimensions. These are
checked at compile and run-time respectively.

(let ((a #2A((1 2) (3 4))))
(vectorize* t (a) (+ a 1)))
-> #2A((2 3) (4 5))

(let ((a #(1 2 3))
(b #(4 5 6)))
(vectorize* t (a b) (+ a (* b 2))))
-> #(9 12 15)

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`vectorize-reduce` (*fn variables* **&body** *body*) [Macro]

Performs a reduction using FN over all elements in a vectorized expression on array VARIABLES.

VARIABLES must be a list of symbols bound to arrays.
Each array must have the same dimensions. These are
checked at compile and run-time respectively.

Example: Maximum value in an array A

(vectorize-reduce #'max (a) a)

Example: Maximum absolute difference between two arrays A and B

(vectorize-reduce #'max (a b) (abs (- a b)))

Package [`array-operations/reducing`], page 18.

Source [`file-type.lisp`], page 10.

`walk-subscripts` ((*dimensions subscripts* **&optional** *position)* **&body** *body*) [Macro]

Iterate over the subscripts of an array with given DIMENSIONS. SUBSCRIPTS contains the current subscripts as a vector of fixnums, POSITION has the row-major index. Consequences are undefined if either POSITION or SUBSCRIPTS is modified.

Package [`array-operations/utilities`], page 18.

Source [`file-type.lisp`], page 11.

`walk-subscripts-list` ((*dimensions subscripts* **&optional** *position)* **&body** *body*) [Macro]

Like WALK-SUBSCRIPTS, but SUBSCRIPTS is a newly created list for each position that does not share structure and can be freely used/modified/kept etc.

Package [`array-operations/utilities`], page 18.

Source [file-type.lisp], page 11.

4.1.2 Ordinary functions

`argmax` (*array*) [Function]
Find the row-major-aref in ARRAY with the maximum value Returns both the index and the value of ARRAY at that index

Package [array-operations/reducing], page 18.

Source [file-type.lisp], page 10.

`argmin` (*array*) [Function]
Find the row-major-aref in ARRAY with the minimum value Returns both the index and the value of ARRAY at that index

Package [array-operations/reducing], page 18.

Source [file-type.lisp], page 10.

`best` (*fn array*) [Function]
FN must accept two inputs and return true/false. This function is applied
to elements of ARRAY, to find the 'best'. The row-major-aref index is returned.

Example: The index of the maximum is

```
* (best #'> #(1 2 3 4))
3 ; row-major index
4 ; value
```

This function was adapted from Paul Graham's On Lisp

Package [array-operations/reducing], page 18.

Source [file-type.lisp], page 10.

`check-permutation` (*permutation* **&optional** *rank*) [Function]
Check if PERMUTATION is a valid permutation (of the given RANK), and signal an error if necessary.

Package [array-operations/transforming], page 19.

Source [file-type.lisp], page 12.

`coercing` (*element-type* **&optional** *function*) [Function]
Return a function composed of a univariate function that coerces to ELEMENT-TYPE and function. When FUNCTION is not given, return a closure that coerces to ELEMENT-TYPE.

Package [array-operations/transforming], page 19.

Source [file-type.lisp], page 12.

`combine` (*array* **&optional** *element-type*) [Function]
The opposite of SUBARRAYS.
If ELEMENT-TYPE is not given, it is inferred from the first element of array, which also determines the dimensions. If that element is not an array, the original ARRAY is returned as it is.

Package [array-operations/displacing], page 17.

Source [file-type.lisp], page 11.

`complement-permutation` (*permutation rank*) [Function]
Return a list of increasing indices that complement PERMUTATION, i.e. form a permutation when appended. Atoms are accepted and treated as lists of a single element.

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`complete-permutation` (*permutation rank*) [Function]
Return a completed version of permutation, appending it to its complement.

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`copy-into` (*target source*) [Function]
Copy SOURCE into TARGET, for array arguments of compatible dimensions (checked). Return TARGET, making the implementation of the semantics of SETF easy.

Package [`array-operations/displacing`], page 17.

Source [`file-type.lisp`], page 11.

`copy-row-major-block` (*source-array destination-array element-type* **&key** *source-start source-end destination-start*) [Function]
Copy elements with row major indexes between the given start and end from SOURCE to DESTINATION, respectively. Elements are coerced to ELEMENT-TYPE when necessary. Return no values.

This function should be used to implement copying of contiguous row-major blocks of elements, most optimizations should happen here.

Package [`array-operations/stacking`], page 20.

Source [`file-type.lisp`], page 13.

`displace` (*array dimensions* **&optional** *offset*) [Function]
Shorthand function for displacing an array.

Package [`array-operations/displacing`], page 17.

Source [`file-type.lisp`], page 11.

`each` (*function array* **&rest** *other-arrays*) [Function]
Like EACH*, with ELEMENT-TYPE T.

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`each*` (*element-type function array* **&rest** *other-arrays*) [Function]
Apply function to the array arguments elementwise, and return the result as an array with the given ELEMENT-TYPE. Arguments are checked for dimension compatibility.

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`ensure-dimensions` (*object*) [Function]
Return a list of dimensions corresponding to OBJECT. Positive integers are treated as dimensions of rank 1, lists are returned as they are, and arrays are queried for their dimensions.

OBJECTS accepted by this function as valid dimensions are called 'dimension specifications' in this library.

Package [`array-operations/utilities`], page 18.

Source [file-type.lisp], page 11.

`fill!` (*array value*) [Function]
Fills a given ARRAY with VALUE, coerced to the same element type as ARRAY

Package [array-operations/creating], page 16.

Source [file-type.lisp], page 10.

`fill-in-dimensions` (*dimensions size*) [Function]
If one of the dimensions is missing (indicated with T), replace it with a dimension so that the total product equals SIZE. If that's not possible, signal an error. If there are no missing dimensions, just check that the product equals size. Also accepts other dimension specifications (integer, array).

Package [array-operations/displacing], page 17.

Source [file-type.lisp], page 11.

`flatten` (*array*) [Function]
Return ARRAY flattened to a vector. Will share structure.

Package [array-operations/displacing], page 17.

Source [file-type.lisp], page 11.

`generate` (*function dimensions* **&optional** *arguments*) [Function]
Like GENERATE*, with ELEMENT-TYPE T.

Package [array-operations/creating], page 16.

Source [file-type.lisp], page 10.

`generate*` (*element-type function dimensions* **&optional** *arguments*) [Function]
Return an array with given DIMENSIONS and ELEMENT-TYPE, with elements generated by calling FUNCTION.

Function is called with:

- no arguments, when ARGUMENTS is nil
- the position (= row major index), when ARGUMENTS is :POSITION
- a list of subscripts, when ARGUMENTS is :SUBSCRIPTS
- both when ARGUMENTS is :POSITION-AND-SUBSCRIPTS

The traversal order is unspecified and may be nonlinear.

Package [array-operations/creating], page 16.

Source [file-type.lisp], page 10.

`identity-permutation-p` (*permutation* **&optional** *rank*) [Function]
Test if PERMUTATION is the identity permutation, i.e. a sequence of consecutive integers starting at 0. Note that permutation is otherwise not checked, i.e. it may not be a permutation.

Package [array-operations/transforming], page 19.

Source [file-type.lisp], page 12.

`identity-permutation?` (*permutation* **&optional** *rank*) [Function]
Test if PERMUTATION is the identity permutation, i.e. a sequence of consecutive integers starting at 0. Note that permutation is otherwise not checked, i.e. it may not be a permutation.

Package [`array-operations/transforming`], page 19.

Alias for [`identity-permutation-p`], page 28.

`invert-permutation` (*permutation*) [Function]
Invert a permutation.

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`linspace` (*start stop n*) [Function]
Make a vector of N elements and type T, containing evenly spaced numbers over an interval.
The first element is equal to START and last element STOP,
with constant difference between consecutive elements.

(linspace 0 4 5) -> #(0 1 2 3 4)
(linspace 1 3 5) -> #(0 1/2 1 3/2 2)
(linspace 0 4d0 3) -> #(0.0d0 2.0d0 4.0d0)

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`linspace!` (*array start stop*) [Function]
Fill an array with evenly spaced numbers over an interval.
The first element is equal to START and last element STOP,
with constant difference between consecutive elements in ROW-MAJOR-INDEX.

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`linspace*` (*element-type start stop n*) [Function]
Make a vector of N elements and type ELEMENT-TYPE, containing evenly spaced numbers over an interval. The first element is equal to START and last element STOP,
with constant difference between consecutive elements.

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`make-array-like` (*array* **&key** *dimensions adjustable element-type initial-element fill-pointer*) [Function]
Returns new array with the same properties as ARRAY.
Keyword arguments will override properties of ARRAY.
If INITIAL-ELEMENT is specified, it is coerced to ELEMENT-TYPE.

Package [`array-operations/creating`], page 16.

Alias for [`similar-array`], page 34.

`margin` (*function array inner* **&optional** *outer*) [Function]
Like MARGIN*, with ELEMENT-TYPE T.

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`margin*` (*element-type function array inner* **&optional** *outer*) [Function]
PERMUTE ARRAY with '(,@OUTER ,@INNER), split the inner subarrays, apply FUNCTION to each, return the results in an array of dimensions OUTER, with the given ELEMENT-TYPE.

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`matrix?` (*matrix*) [Function]
Return NIL if MATRIX does not have rank 2.

Package [`array-operations/matrices`], page 18.

Alias for [`matrixp`], page 30.

`matrixp` (*matrix*) [Function]
Return NIL if MATRIX does not have rank 2.

Package [`array-operations/matrices`], page 18.

Source [`file-type.lisp`], page 10.

`most` (*fn array*) [Function]
Finds the element of ARRAY that returns the value closest to positive infinity when FN is applied to the array value. Returns the row-major-aref index, and the winning value.

Example: The maximum of an array is
(most #'identity #(1 2 3))
-> 2 (row-major index)
3 (value)

Minimum of an array is
(most #'- #(1 2 3))
0
-1

This function was adapted from Paul Graham's On Lisp

Package [`array-operations/reducing`], page 18.

Source [`file-type.lisp`], page 10.

`ones` (*dimensions*) [Function]
Makes an array of shape DIMENSIONS and type T, filled with ones

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`ones!` (*array*) [Function]
Fills the given ARRAY with 1's, coerced to the element type. Returns ARRAY.

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`ones*` (*element-type dimensions*) [Function]
Makes an array of shape DIMENSIONS and type ELEMENT-TYPE, filled with ones coerced to the specified type ELEMENT-TYPE.

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`outer` (*function* **&rest** *arrays*) [Function]
Like OUTER, with ELEMENT-TYPE t.

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`outer*` (*element-type function* **&rest** *arrays*) [Function]
Generalized outer product of ARRAYS with FUNCTION. The resulting array has the concatenated dimensions of ARRAYS, and the given ELEMENT-TYPE.

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`partition` (*array start* **&optional** *end*) [Function]
Return a subset of the array, on the first indexes between START and END.

Package [`array-operations/displacing`], page 17.

Source [`file-type.lisp`], page 11.

`(setf partition)` (*array start* **&optional** *end*) [Function]

Package [`array-operations/displacing`], page 17.

Source [`file-type.lisp`], page 11.

`permute` (*permutation array*) [Function]
Return ARRAY with the axes permuted by PERMUTATION, which is a sequence of indexes. Specifically, an array A is transformed to B, where
B[b_1,...,b_n] = A[a_1,...,a_n] with b_i=a_{P[i]}

P is the permutation.

Array element type is preserved.

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`product` (*dimensions*) [Function]
Product of elements in the argument. NOT EXPORTED.

Package [`array-operations/utilities`], page 18.

Source [`file-type.lisp`], page 11.

`rand` (*dimensions*) [Function]

Makes an array of shape DIMENSIONS and type T, filled with random numbers uniformly distributed between 0 and 1.

Uses the built-in RANDOM function.

(rand 3) -> #(0.39319038 0.69693553 0.5021677)
(rand '(2 2)) -> #2A((0.91003513 0.23208928) (0.5577954 0.94657767))

NOTE: If it's important that these numbers are really random (e.g. cryptographic applications), then you should probably not use this function.

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`rand!` (*array*) [Function]

Fills a given ARRAY with random numbers, uniformly distributed between 0 and 1. Uses the built-in RANDOM function. Returns ARRAY.

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`rand*` (*element-type dimensions*) [Function]

Makes an array of shape DIMENSIONS and type ELEMENT-TYPE, filled with random numbers uniformly distributed between 0 and 1.

Uses the built-in RANDOM function.

(rand 3) -> #(0.39319038 0.69693553 0.5021677)
(rand '(2 2)) -> #2A((0.91003513 0.23208928) (0.5577954 0.94657767))

NOTE: If it's important that these numbers are really random (e.g. cryptographic applications), then you should probably not use this function.

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`randn` (*dimensions*) [Function]

Creates an array of shape DIMENSIONS and type T, and fills with normally distributed numbers with a mean of zero and standard deviation of 1

Uses the Box-Muller algorithm and built-in random number generator.

(rand 3) -> #(-0.82067037 -0.60068226 -0.21494178)
(randn '(2 2)) -> #2A((1.6905352 -2.5379088) (0.8461403 -1.505984))

NOTE: If it's important that these numbers are really random (e.g. cryptographic applications), then you should probably not use this function.

Package [`array-operations/creating`], page 16.

Source [file-type.lisp], page 10.

`randn!` (*array*) [Function]
Fills ARRAY with normally distributed numbers with a mean of zero and standard deviation of 1

Uses the Box-Muller algorithm and built-in random number generator.

NOTE: If it's important that these numbers are really random
(e.g. cryptographic applications), then you should probably
not use this function.

Package [array-operations/creating], page 16.

Source [file-type.lisp], page 10.

`randn*` (*element-type dimensions*) [Function]
Creates an array of shape DIMENSIONS and type ELEMENT-TYPE, and fills with normally distributed numbers with a mean of zero and standard deviation of 1

Uses the Box-Muller algorithm and built-in random number generator.

(rand 3) -> #(-0.82067037 -0.60068226 -0.21494178)
(randn '(2 2)) -> #2A((1.6905352 -2.5379088) (0.8461403 -1.505984))

NOTE: If it's important that these numbers are really random
(e.g. cryptographic applications), then you should probably
not use this function.

Package [array-operations/creating], page 16.

Source [file-type.lisp], page 10.

`recycle` (*object* **&key** *inner outer element-type*) [Function]
Recycle elements of OBJECT, extending the dimensions by outer (repeating OBJECT) and inner (repeating each element of OBJECT). When both INNER and OUTER are nil, the OBJECT is returned as is. Non-array OBJECTs are interpreted as rank 0 arrays, following the usual semantics.

Package [array-operations/transforming], page 19.

Source [file-type.lisp], page 12.

`reshape` (*array dimensions* **&optional** *offset*) [Function]
Reshape ARRAY using DIMENSIONS (which can also be dimension specifications).
If DIMENSIONS is a list, it may contain a single element T which will be calculated to match the total size of the resulting array.

Package [array-operations/displacing], page 17.

Source [file-type.lisp], page 11.

`reshape-col` (*array*) [Function]
Array reshaped as an Nx1 matrix.

Package [array-operations/displacing], page 17.

Source [file-type.lisp], page 11.

`reshape-row` (*array*) [Function]
Array reshaped as an 1xN matrix.

Package [`array-operations/displacing`], page 17.

Source [`file-type.lisp`], page 11.

`same-dimensions-p` (*array* **&rest** *arrays*) [Function]
Test if arguments have the same dimensions. NOT EXPORTED.

Package [`array-operations/utilities`], page 18.

Source [`file-type.lisp`], page 11.

`similar-array` (*array* **&key** *dimensions adjustable element-type initial-element fill-pointer*) [Function]
Returns new array with the same properties as ARRAY.
Keyword arguments will override properties of ARRAY.
If INITIAL-ELEMENT is specified, it is coerced to ELEMENT-TYPE.

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`split` (*array rank*) [Function]
Return an array of subarrays, split off at RANK. All subarrays are displaced and share structure.

Package [`array-operations/displacing`], page 17.

Source [`file-type.lisp`], page 11.

`square-matrix-p` (*matrix*) [Function]
Test if MATRIX has two dimensions and that they are equal.

Package [`array-operations/matrices`], page 18.

Source [`file-type.lisp`], page 10.

`square-matrix?` (*matrix*) [Function]
Test if MATRIX has two dimensions and that they are equal.

Package [`array-operations/matrices`], page 18.

Alias for [`square-matrix-p`], page 34.

`stack` (*axis array* **&rest** *arrays*) [Function]
Like STACK*, with element-type T.

Package [`array-operations/stacking`], page 20.

Source [`file-type.lisp`], page 13.

`stack*` (*element-type axis array* **&rest** *arrays*) [Function]
Stack array arguments along AXIS. ELEMENT-TYPE determines the element-type of the result.

Package [`array-operations/stacking`], page 20.

Source [`file-type.lisp`], page 13.

`stack-cols` (**&rest** *objects*) [Function]
Like STACK-COLS*, with ELEMENT-TYPE T.

Package [`array-operations/stacking`], page 20.

Source [`file-type.lisp`], page 13.

`stack-cols*` (*element-type* **&rest** *objects*) [Function]
Stack OBJECTS column-wise into an array of the given ELEMENT-TYPE, coercing if necessary. Always return a simple array of rank 2.

How objects are used depends on their dimensions, queried by DIMS:

- when the object has 0 dimensions, fill a column with the element.

- when the object has 1 dimension, use it as a column.

- when the object has 2 dimensions, use it as a matrix.

When applicable, compatibility of dimensions is checked, and the result is used to determine the number of rows. When all objects have 0 dimensions, the result has one row.

Package [`array-operations/stacking`], page 20.

Source [`file-type.lisp`], page 13.

`stack-rows` (**&rest** *objects*) [Function]
Like STACK-ROWS*, with ELEMENT-TYPE T.

Package [`array-operations/stacking`], page 20.

Source [`file-type.lisp`], page 13.

`stack-rows*` (*element-type* **&rest** *objects*) [Function]
Stack OBJECTS row-wise into an array of the given ELEMENT-TYPE, coercing if necessary. Always return a simple array of rank 2.

How objects are used depends on their dimensions, queried by DIMS:

- when the object has 0 dimensions, fill a row with the element.

- when the object has 1 dimension, use it as a row.

- when the object has 2 dimensions, use it as a matrix.

When applicable, compatibility of dimensions is checked, and the result is used to determine the number of columns. When all objects have 0 dimensions, the result has one column.

Package [`array-operations/stacking`], page 20.

Source [`file-type.lisp`], page 13.

`sub` (*array* **&rest** *subscripts*) [Function]
Given a partial list of subscripts, return the subarray that starts there, with all the other subscripts set to 0, dimensions inferred from the original.

If no subscripts are given, the original array is returned. Implemented by displacing, may share structure.

Package [`array-operations/displacing`], page 17.

Source [`file-type.lisp`], page 11.

`(setf sub)` (*array* **&rest** *subscripts*) [Function]

Package [`array-operations/displacing`], page 17.

Source [`file-type.lisp`], page 11.

`subvec` (*vector start* **&optional** *end*) [Function]
Displaced vector between START and END.

Package [`array-operations/displacing`], page 17.

Source [`file-type.lisp`], page 11.

`(setf subvec)` (*vector start* **&optional** *end*) [Function]

Package [`array-operations/displacing`], page 17.

Source [`file-type.lisp`], page 11.

`turn` (*array nturns* **&optional** *rank-1 rank-2*) [Function]
Turns an array by a specified number of clockwise 90° rotations. The axis of rotation is specified by RANK-1 (defaulting to 0) and RANK-2 (defaulting to 1).

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`zeros` (*dimensions*) [Function]
Makes an array of shape DIMENSIONS and type T, filled with zeros

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`zeros!` (*array*) [Function]
Fills the given ARRAY with zero values, coerced to the element type. Returns ARRAY.

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

`zeros*` (*element-type dimensions*) [Function]
Makes an array of shape DIMENSIONS and type ELEMENT-TYPE, filled with zeros coerced to the specified type ELEMENT-TYPE.

Package [`array-operations/creating`], page 16.

Source [`file-type.lisp`], page 10.

4.1.3 Generic functions

`as-array` (*object*) [Generic Function]
Return the contents of OBJECT as an array. Exact semantics depends on OBJECT, but generally objects which contain elements in a rectilinear coordinate system should have a natural mapping to arrays.

When the second value is T, the array itself does not share structure with OBJECT, but its elements may. Otherwise, it is indeterminate whether the two objects share structure, and consequences of modifying the result are not defined. Methods are encouraged but not required to return a second value.

Package [`array-operations/generic`], page 15.

Source [`file-type.lisp`], page 9.

Methods

`as-array` ((*array* `array`)) [Method]

`as-array` (*object*) [Method]

`dim` (*array axis*) [Generic Function]
Return specificed dimension of ARRAY.

Package [`array-operations/generic`], page 15.

Source [`file-type.lisp`], page 9.

Methods

`dim` ((*array* `array`) *axis*) [Method]

`dim` (*array axis*) [Method]

`dims` (*array*) [Generic Function]
Return a list of dimensions.

For non-array objects, SIZE, DIM, NROW and NCOL use this method by default, so it is enough to define it (unless efficiency is a concern).

When DIMS is not defined for an object, it falls back to as-array, which may be very inefficient for objects which need to be consed. It is always advisable to define DIMS.

Package [`array-operations/generic`], page 15.

Source [`file-type.lisp`], page 9.

Methods

`dims` ((*array* `array`)) [Method]

`dims` (*array*) [Method]

`element-type` (*array*) [Generic Function]
Return TYPE such that

1. all elements of ARRAY are guaranteed to be a subtype of TYPE,

2. if applicable, elements of ARRAY can be set to values which are of a type that is a subtype of TYPE.

Package [`array-operations/generic`], page 15.

Source [file-type.lisp], page 9.

Methods

element-type ((*array* array)) [Method]

element-type (*array*) [Method]

ncol (*array*) [Generic Function]
Number of columns. Will signal an error if ARRAY is not a matrix.

Package [array-operations/generic], page 15.

Source [file-type.lisp], page 9.

Methods

ncol ((*array* array)) [Method]

ncol (*array*) [Method]

nrow (*array*) [Generic Function]
Number of rows. Will signal an error if ARRAY is not a matrix.

Package [array-operations/generic], page 15.

Source [file-type.lisp], page 9.

Methods

nrow ((*array* array)) [Method]

nrow (*array*) [Method]

rank (*array*) [Generic Function]
Return the rank of ARRAY.

Package [array-operations/generic], page 15.

Source [file-type.lisp], page 9.

Methods

rank ((*array* array)) [Method]

rank (*array*) [Method]

size (*array*) [Generic Function]
Return the total number of elements in array.

Package [array-operations/generic], page 15.

Source [file-type.lisp], page 9.

Methods

size ((*array* array)) [Method]

size (*array*) [Method]

stack-cols-copy (*source destination element-type start-col*) [Generic Function]
Method used to implement the copying of objects in STACK-COL*, by copying the elements of SOURCE to DESTINATION, starting with the column index START-COL in the latter. Elements are coerced to ELEMENT-TYPE.

This method is only called when (DIMS SOURCE) was non-nil. It is assumed that it only changes elements in DESTINATION which are supposed to be copies of SOURCE. DESTINATION is always a matrix with element-type upgraded from ELEMENT-TYPE, and its NROW should match the relevant dimension of SOURCE.

All objects have a fallback method, defined using AS-ARRAY. The only reason for defining a method is efficiency.

Package [array-operations/stacking], page 20.

Source [file-type.lisp], page 13.

Methods

`stack-cols-copy` (*source destination element-type start-col*) [Method]

`stack-cols-copy` ((*source* `array`) *destination element-type start-col*) [Method]

`stack-rows-copy` (*source destination element-type start-row*) [Generic Function]

Method used to implement the copying of objects in STACK-ROW*, by copying the elements of SOURCE to DESTINATION, starting with the row index START-ROW in the latter. Elements are coerced to ELEMENT-TYPE.

This method is only called when (DIMS SOURCE) was non-nil. It is assumed that it only changes elements in DESTINATION which are supposed to be copies of SOURCE. DESTINATION is always a matrix with element-type upgraded from ELEMENT-TYPE, and its NCOL should match the relevant dimension of SOURCE.

All objects have a fallback method, defined using AS-ARRAY. The only reason for defining a method is efficiency.

Package [array-operations/stacking], page 20.

Source [file-type.lisp], page 13.

Methods

`stack-rows-copy` (*source destination element-type start-row*) [Method]

`stack-rows-copy` ((*source* `array`) *destination element-type start-row*) [Method]

4.1.4 Conditions

`permutation-incompatible-rank` [Condition]

Package [array-operations/transforming], page 19.

Source [file-type.lisp], page 12.

Direct superclasses

`error`.

`permutation-invalid-index` [Condition]

Package [array-operations/transforming], page 19.

Source [file-type.lisp], page 12.

Direct superclasses

`error`.

Direct slots

`index` [Slot]

Initargs `:index`

`permutation-repeated-index` [Condition]

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

Direct superclasses

`error`.

Direct slots

`index` [Slot]

Initargs `:index`

4.1.5 Types

`array-matrix` () [Type]

A rank-2 array.

Package [`array-operations/matrices`], page 18.

Source [`file-type.lisp`], page 10.

4.2 Internals

4.2.1 Macros

`foreach` (**&key** *index sum value*) [Macro]

Examples:

Matrix-matrix multiply

(foreach :index (i j) :sum k
:value (* (aref A i k) (aref B k j)))

Sum over vector

(foreach :sum i :value (aref A i))

Package [`array-operations/indexing`], page 15.

Source [`file-type.lisp`], page 11.

4.2.2 Ordinary functions

`array-index-row-major` (*array rmi* **&optional** *result*) [Function]

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`find-array-dimensions` (*expr*) [Function]
Walks an expression tree EXPR, finds AREF and ROW-MAJOR-AREF, SVREF or ELT calls. Returns a list of (symbol, expr) where EXPR is an expression which evaluates to the array dimension size for SYMBOL.

Example:
(find-array-dimensions '(+ (aref a i) (* 2 (aref b j k))))

-> ((I ARRAY-DIMENSION A 0) (K ARRAY-DIMENSION B 1) (J ARRAY-DIMENSION B 0))

Package [`array-operations/indexing`], page 15.

Source [`file-type.lisp`], page 11.

`permutation-flags` (*permutation* **&optional** *rank*) [Function]
Make a bit vector of flags with indexes from PERMUTATION, signaling errors for invalid and repeated indices. NOT EXPORTED.

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

`stack*0` (*element-type arrays*) [Function]
Stack arrays along the 0 axis, returning an array with given ELEMENT-TYPE.

Package [`array-operations/stacking`], page 20.

Source [`file-type.lisp`], page 13.

`sub-location%` (*dimensions subscripts*) [Function]
Return (values OFFSET REMAINING-DIMENSIONS) that can be used to displace a row-major subarray starting at SUBSCRIPTS in an array with the given DIMENSIONS. NOT EXPORTED.

Package [`array-operations/displacing`], page 17.

Source [`file-type.lisp`], page 11.

4.2.3 Types

`array-rank-element` () [Type]

Package [`array-operations/transforming`], page 19.

Source [`file-type.lisp`], page 12.

Appendix A Indexes

A.1 Concepts

(Index is nonexistent)

A.2 Functions

A.3 Variables

I

S

A.4 Data types

A

C

F

P

S

T