

**aocl-sparse API Guide**  
Version v3.1.0.0



# Table of Contents

- aocl-sparse API Guide.....i
  - Version v3.1.0.0 .....i
- Table of Contents ..... iii
- File Index.....2
  - File List.....2
- Introduction .....3
  - Auxiliary Functions .....3
    - Functions .....3
    - Detailed Description .....4
    - Function Documentation.....4
  - Conversion Functions.....10
    - Functions .....10
    - Detailed Description .....11
    - Function Documentation.....11
  - Sparse Level 2 & 3 Functions.....20
    - Functions .....20
    - Detailed Description .....21
    - Function Documentation.....21
- aoclsparse\_types.h File Reference.....34
  - Macros .....34
  - Typedefs .....34
  - Enumerations .....34
  - Detailed Description .....35
  - Macro Definition Documentation .....35
  - Typedef Documentation .....35
  - Enumeration Type Documentation .....37



# File Index

## File List

Here is a list of all documented files with brief descriptions:

- aoclsparse\_auxiliary.h (Aoclsparse\_auxiliary.h provides auxiliary functions in aoclsparse ) .....3**
- aoclsparse\_convert.h (Aoclsparse\_convert.h provides Sparse Format conversion Subprograms ) .....10**
- aoclsparse\_functions.h (Aoclsparse\_functions.h provides Sparse Linear Algebra Subprograms of Level 1, 2 and 3, for AMD CPU hardware ) .....20**
- aoclsparse\_types.h (Aoclsparse\_types.h defines data types used by aoclsparse ) .....34**

# Introduction

aocl-sparse is a library that contains basic linear algebra subroutines for sparse matrices and vectors optimized for AMD EPYC family of processors. It is designed to be used with C and C++.

The current functionality of aocl-sparse is organized in the following categories:

- Sparse Level 3 Functions describe operations between a matrix in sparse format and a matrix in dense format.
- Sparse Level 2 Functions describe operations between a matrix in sparse format and a vector in dense format.
- Sparse Format Conversion Functions describe operations on a matrix in sparse format to obtain a different matrix format.
- Sparse Auxiliary Functions describe auxiliary functions.

## Auxiliary Functions

**aoclsparse\_auxiliary.h** provides auxiliary functions in aoclsparse

### Functions

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_get\_version (aoclsparse\_int \*version)**

*Get aoclsparse version.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_create\_mat\_descr (aoclsparse\_mat\_descr \*descr)**

*Create a matrix descriptor.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_copy\_mat\_descr (aoclsparse\_mat\_descr dest, const aoclsparse\_mat\_descr src)**

*Copy a matrix descriptor.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_destroy\_mat\_descr (aoclsparse\_mat\_descr descr)**

*Destroy a matrix descriptor.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_set\_mat\_index\_base (aoclsparse\_mat\_descr descr, aoclsparse\_index\_base base)**

*Specify the index base of a matrix descriptor.*

**DLL\_PUBLIC aoclsparse\_index\_base aoclsparse\_get\_mat\_index\_base (const aoclsparse\_mat\_descr descr)**

*Get the index base of a matrix descriptor.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_set\_mat\_type (aoclsparse\_mat\_descr descr, aoclsparse\_matrix\_type type)**

*Specify the matrix type of a matrix descriptor.*

**DLL\_PUBLIC aoclsparse\_matrix\_type aoclsparse\_get\_mat\_type (const aoclsparse\_mat\_descr descr)**

*Get the matrix type of a matrix descriptor.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_set\_mat\_fill\_mode (aoclsparse\_mat\_descr descr, aoclsparse\_fill\_mode fill\_mode)**

*Specify the matrix fill mode of a matrix descriptor.*

**DLL\_PUBLIC aoclsparse\_fill\_mode aoclsparse\_get\_mat\_fill\_mode** (const **aoclsparse\_mat\_descr** descr)

*Get the matrix fill mode of a matrix descriptor.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_set\_mat\_diag\_type** (**aoclsparse\_mat\_descr** descr, **aoclsparse\_diag\_type** diag\_type)

*Specify the matrix diagonal type of a matrix descriptor.*

**DLL\_PUBLIC aoclsparse\_diag\_type aoclsparse\_get\_mat\_diag\_type** (const **aoclsparse\_mat\_descr** descr)

*Get the matrix diagonal type of a matrix descriptor.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_create\_mat\_csr** (**aoclsparse\_mat\_csr** &csr, **aoclsparse\_index\_base** base, **aoclsparse\_int** m, **aoclsparse\_int** n, **aoclsparse\_int** csr\_nnz, **aoclsparse\_int** \*csr\_row\_ptr, **aoclsparse\_int** \*csr\_col\_ind, void \*csr\_val)

*Update a CSR matrix structure.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_destroy\_mat\_csr** (**aoclsparse\_mat\_csr** csr)

*Destroy a CSR matrix structure.*

---

## Detailed Description

**aoclsparse\_auxiliary.h** provides auxiliary functions in **aoclsparse**

---

## Function Documentation

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_copy\_mat\_descr** (**aoclsparse\_mat\_descr** dest, const **aoclsparse\_mat\_descr** src)

Copy a matrix descriptor.

**aoclsparse\_copy\_mat\_descr** copies a matrix descriptor. Both, source and destination matrix descriptors must be initialized prior to calling **aoclsparse\_copy\_mat\_descr**.

### Parameters

out	dest	the pointer to the destination matrix descriptor.
in	src	the pointer to the source matrix descriptor.

### Return values

<b>aoclsparse_status_success</b>	the operation completed successfully.
<b>aoclsparse_status_invalid_pointer</b>	src or dest pointer is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_create\_mat\_csr** (**aoclsparse\_mat\_csr** &csr, **aoclsparse\_index\_base** base, **aoclsparse\_int** m, **aoclsparse\_int** n, **aoclsparse\_int** csr\_nnz, **aoclsparse\_int** \*csr\_row\_ptr, **aoclsparse\_int** \*csr\_col\_ind, void \*csr\_val)

Update a CSR matrix structure.

`aoclsparse_create_mat_csr` updates a structure that holds the matrix in CSR storage format. It should be destroyed at the end using `aoclsparse_destroy_mat_csr()`.

#### Parameters

in,out	<i>csr</i>	the pointer to the CSR sparse matrix.
in	<i>base</i>	<b>aoclsparse_index_base_zero</b> or <b>aoclsparse_index_base_one</b> .
in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of columns of the sparse CSR matrix.
in	<i>csr_nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of m+1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of nnz elements containing the column indices of the sparse CSR matrix.
in	<i>csr_val</i>	array of nnz elements of the sparse CSR matrix.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	csr pointer is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_create\_mat\_descr (aoclsparse\_mat\_descr \* descr)**

Create a matrix descriptor.

`aoclsparse_create_mat_descr` creates a matrix descriptor. It initializes **aoclsparse\_matrix\_type** to **aoclsparse\_matrix\_type\_general** and **aoclsparse\_index\_base** to **aoclsparse\_index\_base\_zero**. It should be destroyed at the end using `aoclsparse_destroy_mat_descr()`.

#### Parameters

out	<i>descr</i>	the pointer to the matrix descriptor.
-----	--------------	---------------------------------------

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	descr pointer is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_destroy\_mat\_csr (aoclsparse\_mat\_csr csr)**

Destroy a CSR matrix structure.

`aoclsparse_destroy_mat_csr` destroys a structure that holds the matrix in CSR storage format.

#### Parameters

in	<i>csr</i>	the pointer to the CSR sparse matrix.
----	------------	---------------------------------------

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	csr pointer is invalid.



## **DLL\_PUBLIC aoclsparse\_status aoclsparse\_destroy\_mat\_descr (aoclsparse\_mat\_descr descr)**

Destroy a matrix descriptor.

`aoclsparse_destroy_mat_descr` destroys a matrix descriptor and releases all resources used by the descriptor.

### **Parameters**

in	<i>descr</i>	the matrix descriptor.
----	--------------	------------------------

### **Return values**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> is invalid.

## **DLL\_PUBLIC aoclsparse\_diag\_type aoclsparse\_get\_mat\_diag\_type (const aoclsparse\_mat\_descr descr)**

Get the matrix diagonal type of a matrix descriptor.

`aoclsparse_get_mat_diag_type` returns the matrix diagonal type of a matrix descriptor.

### **Parameters**

in	<i>descr</i>	the matrix descriptor.
----	--------------	------------------------

### **Returns**

`aoclsparse_diag_type_unit` or `aoclsparse_diag_type_non_unit`.

## **DLL\_PUBLIC aoclsparse\_fill\_mode aoclsparse\_get\_mat\_fill\_mode (const aoclsparse\_mat\_descr descr)**

Get the matrix fill mode of a matrix descriptor.

`aoclsparse_get_mat_fill_mode` returns the matrix fill mode of a matrix descriptor.

### **Parameters**

in	<i>descr</i>	the matrix descriptor.
----	--------------	------------------------

### **Returns**

`aoclsparse_fill_mode_lower` or `aoclsparse_fill_mode_upper`.

## **DLL\_PUBLIC aoclsparse\_index\_base aoclsparse\_get\_mat\_index\_base (const aoclsparse\_mat\_descr descr)**

Get the index base of a matrix descriptor.

`aoclsparse_get_mat_index_base` returns the index base of a matrix descriptor.

### **Parameters**

in	<i>descr</i>	the matrix descriptor.
----	--------------	------------------------

### **Returns**

`aoclsparse_index_base_zero` or `aoclsparse_index_base_one`.

## **DLL\_PUBLIC aoclsparse\_matrix\_type aoclsparse\_get\_mat\_type (const aoclsparse\_mat\_descr descr)**

Get the matrix type of a matrix descriptor.

`aoclsparse_get_mat_type` returns the matrix type of a matrix descriptor.

### **Parameters**

in	<i>descr</i>	the matrix descriptor.
----	--------------	------------------------

### **Returns**

`aoclsparse_matrix_type_general`, `aoclsparse_matrix_type_symmetric`,  
`aoclsparse_matrix_type_hermitian` or `aoclsparse_matrix_type_triangular`.

## **DLL\_PUBLIC aoclsparse\_status aoclsparse\_get\_version (aoclsparse\_int \* version)**

Get aoclsparse version.

`aoclsparse_get_version` gets the aoclsparse library version number.

patch = version % 100

minor = version / 100 % 1000

major = version / 100000

### **Parameters**

out	<i>version</i>	the version number of the aoclsparse library.
-----	----------------	---

### **Return values**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<i>version</i> is invalid.

## **DLL\_PUBLIC aoclsparse\_status aoclsparse\_set\_mat\_diag\_type (aoclsparse\_mat\_descr descr, aoclsparse\_diag\_type diag\_type)**

Specify the matrix diagonal type of a matrix descriptor.

`aoclsparse_set_mat_diag_type` sets the matrix diagonal type of a matrix descriptor. Valid diagonal types are `aoclsparse_diag_type_unit` or `aoclsparse_diag_type_non_unit`.

### **Parameters**

in,out	<i>descr</i>	the matrix descriptor.
in	<i>diag_type</i>	<code>aoclsparse_diag_type_unit</code> or <code>aoclsparse_diag_type_non_unit</code> .

### **Return values**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> pointer is invalid.
<i>aoclsparse_status_invalid_value</i>	<i>diag_type</i> is invalid.

## **DLL\_PUBLIC aoclsparse\_status aoclsparse\_set\_mat\_fill\_mode (aoclsparse\_mat\_descr descr, aoclsparse\_fill\_mode fill\_mode)**

Specify the matrix fill mode of a matrix descriptor.

`aoclsparse_set_mat_fill_mode` sets the matrix fill mode of a matrix descriptor. Valid fill modes are `aoclsparse_fill_mode_lower` or `aoclsparse_fill_mode_upper`.

#### Parameters

in,out	<i>descr</i>	the matrix descriptor.
in	<i>fill_mode</i>	<code>aoclsparse_fill_mode_lower</code> or <code>aoclsparse_fill_mode_upper</code> .

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr</code> pointer is invalid.
<i>aoclsparse_status_invalid_value</i>	<code>fill_mode</code> is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_set\_mat\_index\_base**  
(`aoclsparse_mat_descr descr`, `aoclsparse_index_base base`)

Specify the index base of a matrix descriptor.

`aoclsparse_set_mat_index_base` sets the index base of a matrix descriptor. Valid options are `aoclsparse_index_base_zero` or `aoclsparse_index_base_one`.

#### Parameters

in,out	<i>descr</i>	the matrix descriptor.
in	<i>base</i>	<code>aoclsparse_index_base_zero</code> or <code>aoclsparse_index_base_one</code> .

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr</code> pointer is invalid.
<i>aoclsparse_status_invalid_value</i>	<code>base</code> is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_set\_mat\_type** (`aoclsparse_mat_descr descr`, `aoclsparse_matrix_type type`)

Specify the matrix type of a matrix descriptor.

`aoclsparse_set_mat_type` sets the matrix type of a matrix descriptor. Valid matrix types are `aoclsparse_matrix_type_general`, `aoclsparse_matrix_type_symmetric`, `aoclsparse_matrix_type_hermitian` or `aoclsparse_matrix_type_triangular`.

#### Parameters

in,out	<i>descr</i>	the matrix descriptor.
in	<i>type</i>	<code>aoclsparse_matrix_type_general</code> , <code>aoclsparse_matrix_type_symmetric</code> , <code>aoclsparse_matrix_type_hermitian</code> or <code>aoclsparse_matrix_type_triangular</code> .

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr</code> pointer is invalid.
<i>aoclsparse_status_invalid_value</i>	<code>type</code> is invalid.



## Conversion Functions

**aoclsparse\_convert.h** provides Sparse Format conversion Subprograms

### Functions

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_csr2ell\_width** (aoclsparse\_int m, aoclsparse\_int nnz, const aoclsparse\_int \*csr\_row\_ptr, aoclsparse\_int \*ell\_width)

*Convert a sparse CSR matrix into a sparse ELL matrix.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_csr2dia\_ndiag** (aoclsparse\_int m, aoclsparse\_int n, aoclsparse\_int nnz, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, aoclsparse\_int \*dia\_num\_diag)

*Convert a sparse CSR matrix into a sparse DIA matrix.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_csr2bsr\_nnz** (aoclsparse\_int m, aoclsparse\_int n, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, aoclsparse\_int block\_dim, aoclsparse\_int \*bsr\_row\_ptr, aoclsparse\_int \*bsr\_nnz)

*aoclsparse\_csr2bsr\_nnz computes the number of nonzero block columns per row and the total number of nonzero blocks in a sparse BSR matrix given a sparse CSR matrix as input.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsr2ell** (aoclsparse\_int m, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, const float \*csr\_val, aoclsparse\_int \*ell\_col\_ind, float \*ell\_val, aoclsparse\_int ell\_width)

*Convert a sparse CSR matrix into a sparse ELLPACK matrix.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsr2ell** (aoclsparse\_int m, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, const double \*csr\_val, aoclsparse\_int \*ell\_col\_ind, double \*ell\_val, aoclsparse\_int ell\_width)

*Convert a sparse CSR matrix into a sparse ELLPACK matrix.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsr2dia** (aoclsparse\_int m, aoclsparse\_int n, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, const float \*csr\_val, aoclsparse\_int dia\_num\_diag, aoclsparse\_int \*dia\_offset, float \*dia\_val)

*Convert a sparse CSR matrix into a sparse DIA matrix.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsr2dia** (aoclsparse\_int m, aoclsparse\_int n, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, const double \*csr\_val, aoclsparse\_int dia\_num\_diag, aoclsparse\_int \*dia\_offset, double \*dia\_val)

*Convert a sparse CSR matrix into a sparse DIA matrix.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsr2bsr** (aoclsparse\_int m, aoclsparse\_int n, const float \*csr\_val, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, aoclsparse\_int block\_dim, float \*bsr\_val, aoclsparse\_int \*bsr\_row\_ptr, aoclsparse\_int \*bsr\_col\_ind)

*Convert a sparse CSR matrix into a sparse BSR matrix.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsr2bsr** (aoclsparse\_int m, aoclsparse\_int n, const double \*csr\_val, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, aoclsparse\_int block\_dim, double \*bsr\_val, aoclsparse\_int \*bsr\_row\_ptr, aoclsparse\_int \*bsr\_col\_ind)

*Convert a sparse CSR matrix into a sparse BSR matrix.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsr2csc** (aoclsparse\_int m, aoclsparse\_int n, aoclsparse\_int nnz, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, const float \*csr\_val, aoclsparse\_int \*csc\_row\_ind, aoclsparse\_int \*csc\_col\_ptr, float \*csc\_val)

*Convert a sparse CSR matrix into a sparse CSC matrix.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsr2csc** (aoclsparse\_int m, aoclsparse\_int n, aoclsparse\_int nnz, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, const double \*csr\_val, aoclsparse\_int \*csc\_row\_ind, aoclsparse\_int \*csc\_col\_ptr, double \*csc\_val)

*Convert a sparse CSR matrix into a sparse CSC matrix.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsr2dense** (aoclsparse\_int m, aoclsparse\_int n, const aoclsparse\_mat\_descr descr, const float \*csr\_val, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, float \*A, aoclsparse\_int ld, aoclsparse\_order order)

*This function converts the sparse matrix in CSR format into a dense matrix.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsr2dense** (aoclsparse\_int m, aoclsparse\_int n, const aoclsparse\_mat\_descr descr, const double \*csr\_val, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_int \*csr\_col\_ind, double \*A, aoclsparse\_int ld, aoclsparse\_order order)

*This function converts the sparse matrix in CSR format into a dense matrix.*

---

## Detailed Description

**aoclsparse\_convert.h** provides Sparse Format conversion Subprograms

---

## Function Documentation

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_csr2bsr\_nnz** (aoclsparse\_int m, aoclsparse\_int n, const aoclsparse\_int \* csr\_row\_ptr, const aoclsparse\_int \* csr\_col\_ind, aoclsparse\_int block\_dim, aoclsparse\_int \* bsr\_row\_ptr, aoclsparse\_int \* bsr\_nnz)

aoclsparse\_csr2bsr\_nnz computes the number of nonzero block columns per row and the total number of nonzero blocks in a sparse BSR matrix given a sparse CSR matrix as input.

### Parameters

in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of columns of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	integer array containing $m+1$ elements that point to the start of each row of the CSR matrix
in	<i>csr_col_ind</i>	integer array of the column indices for each non-zero element in the CSR matrix
in	<i>block_dim</i>	the block dimension of the BSR matrix. Between 1 and $\min(m, n)$
out	<i>bsr_row_ptr</i>	integer array containing $m_b+1$ elements that point to the start of each block row of the BSR matrix
out	<i>bsr_nnz</i>	total number of nonzero elements in device or host memory.

### Return values

<i>aoclsparse_status_</i>	the operation completed successfully.
---------------------------	---------------------------------------

<i>success</i>	
<i>aoclsparse_status_invalid_size</i>	m or n or block_dim is invalid.
<i>aoclsparse_status_invalid_pointer</i>	csr_row_ptr or csr_col_ind or bsr_row_ptr or bsr_nnz pointer is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_csr2dia\_ndiag (aoclsparse\_int m, aoclsparse\_int n, aoclsparse\_int nnz, const aoclsparse\_int \* csr\_row\_ptr, const aoclsparse\_int \* csr\_col\_ind, aoclsparse\_int \* dia\_num\_diag)**

Convert a sparse CSR matrix into a sparse DIA matrix.

`aoclsparse_csr2dia_ndiag` computes the number of the diagonals for a given CSR matrix.

#### Parameters

in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of cols of the sparse CSR matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of m+1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array containing the column indices of the sparse CSR matrix.
out	<i>dia_num_diag</i>	pointer to the number of diagonals with non-zeroes in DIA storage format.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	m is invalid.
<i>aoclsparse_status_invalid_pointer</i>	csr_row_ptr, or ell_width pointer is invalid.
<i>aoclsparse_status_internal_error</i>	an internal error occurred.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_csr2ell\_width (aoclsparse\_int m, aoclsparse\_int nnz, const aoclsparse\_int \* csr\_row\_ptr, aoclsparse\_int \* ell\_width)**

Convert a sparse CSR matrix into a sparse ELL matrix.

`aoclsparse_csr2ell_width` computes the maximum of the per row non-zero elements over all rows, the ELL width, for a given CSR matrix.

#### Parameters

in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of m+1 elements that point to the start of every row of the sparse CSR matrix.
out	<i>ell_width</i>	pointer to the number of non-zero elements per row in ELL storage format.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	m is invalid.
<i>aoclsparse_status_invalid_pointer</i>	csr_row_ptr, or ell_width pointer is invalid.
<i>aoclsparse_status_</i>	an internal error occurred.

<i>internal_error</i>	
-----------------------	--

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsr2bsr (aoclsparse\_int *m*, aoclsparse\_int *n*, const double \* *csr\_val*, const aoclsparse\_int \* *csr\_row\_ptr*, const aoclsparse\_int \* *csr\_col\_ind*, aoclsparse\_int *block\_dim*, double \* *bsr\_val*, aoclsparse\_int \* *bsr\_row\_ptr*, aoclsparse\_int \* *bsr\_col\_ind*)**

Convert a sparse CSR matrix into a sparse BSR matrix.

`aoclsparse_csr2bsr` converts a CSR matrix into a BSR matrix. It is assumed, that `bsr_val`, `bsr_col_ind` and `bsr_row_ptr` are allocated. Allocation size for `bsr_row_ptr` is computed as `mb+1` where `mb` is the number of block rows in the BSR matrix. Allocation size for `bsr_val` and `bsr_col_ind` is computed using `csr2bsr_nnz()` which also fills in `bsr_row_ptr`.

#### Parameters

in	<i>m</i>	number of rows in the sparse CSR matrix.
in	<i>n</i>	number of columns in the sparse CSR matrix.
in	<i>csr_val</i>	array of <code>nnz</code> elements containing the values of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <code>m+1</code> elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of <code>nnz</code> elements containing the column indices of the sparse CSR matrix.
in	<i>block_dim</i>	size of the blocks in the sparse BSR matrix.
out	<i>bsr_val</i>	array of <code>nnzb*block_dim*block_dim</code> containing the values of the sparse BSR matrix.
out	<i>bsr_row_ptr</i>	array of <code>mb+1</code> elements that point to the start of every block row of the sparse BSR matrix.
out	<i>bsr_col_ind</i>	array of <code>nnzb</code> elements containing the block column indices of the sparse BSR matrix.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<code>m</code> or <code>n</code> or <code>block_dim</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>bsr_val</code> , <code>bsr_row_ptr</code> , <code>bsr_col_ind</code> , <code>csr_val</code> , <code>csr_row_ptr</code> or <code>csr_col_ind</code> pointer is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsr2csc (aoclsparse\_int *m*, aoclsparse\_int *n*, aoclsparse\_int *nnz*, const aoclsparse\_int \* *csr\_row\_ptr*, const aoclsparse\_int \* *csr\_col\_ind*, const double \* *csr\_val*, aoclsparse\_int \* *csc\_row\_ind*, aoclsparse\_int \* *csc\_col\_ptr*, double \* *csc\_val*)**

Convert a sparse CSR matrix into a sparse CSC matrix.

`aoclsparse_csr2csc` converts a CSR matrix into a CSC matrix. `aoclsparse_csr2csc` can also be used to convert a CSC matrix into a CSR matrix.

#### Note

The resulting matrix can also be seen as the transpose of the input matrix.

#### Parameters

in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of columns of the sparse CSR matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_val</i>	array of <code>nnz</code> elements of the sparse CSR matrix.



in	<i>csr_row_ptr</i>	array of $m+1$ elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of $nnz$ elements containing the column indices of the sparse CSR matrix.
out	<i>csc_val</i>	array of $nnz$ elements of the sparse CSC matrix.
out	<i>csc_row_ind</i>	array of $nnz$ elements containing the row indices of the sparse CSC matrix.
out	<i>csc_col_ptr</i>	array of $n+1$ elements that point to the start of every column of the sparse CSC matrix. <code>aoclsparse_csr2csc_buffer_size()</code> .

**Return values**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m$ , $n$ or $nnz$ is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>csr_val</i> , <i>csr_row_ptr</i> , <i>csr_col_ind</i> , <i>csc_val</i> , <i>csc_row_ind</i> , <i>csc_col_ptr</i> is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsr2dense (aoclsparse\_int *m*, aoclsparse\_int *n*, const aoclsparse\_mat\_descr *descr*, const double \* *csr\_val*, const aoclsparse\_int \* *csr\_row\_ptr*, const aoclsparse\_int \* *csr\_col\_ind*, double \* *A*, aoclsparse\_int *ld*, aoclsparse\_order *order*)**

This function converts the sparse matrix in CSR format into a dense matrix.

**Parameters**

in	<i>m</i>	number of rows of the dense matrix <i>A</i> .
in	<i>n</i>	number of columns of the dense matrix <i>A</i> .
in	<i>descr</i>	the descriptor of the dense matrix <i>A</i> , the supported matrix type is <b>aoclsparse_matrix_type_general</b> and also any valid value of the <b>aoclsparse_index_base</b> .
in	<i>csr_val</i>	array of $nnz$ ( = <i>csr_row_ptr</i> [ $m$ ] - <i>csr_row_ptr</i> [0] ) nonzero elements of matrix <i>A</i> .
in	<i>csr_row_ptr</i>	integer array of $m+1$ elements that contains the start of every row and the end of the last row plus one.
in	<i>csr_col_ind</i>	integer array of $nnz$ ( = <i>csr_row_ptr</i> [ $m$ ] - <i>csr_row_ptr</i> [0] ) column indices of the non-zero elements of matrix <i>A</i> .
out	<i>A</i>	array of dimensions ( $ld$ , $n$ )
out	<i>ld</i>	leading dimension of dense array <i>A</i> .
in	<i>order</i>	memory layout of a dense matrix <i>A</i> .

**Return values**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m$ or $n$ or $ld$ is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>A</i> or <i>csr_val</i> <i>csr_row_ptr</i> or <i>csr_col_ind</i> pointer is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsr2dia (aoclsparse\_int *m*, aoclsparse\_int *n*, const aoclsparse\_int \* *csr\_row\_ptr*, const aoclsparse\_int \* *csr\_col\_ind*, const double \* *csr\_val*, aoclsparse\_int *dia\_num\_diag*, aoclsparse\_int \* *dia\_offset*, double \* *dia\_val*)**

Convert a sparse CSR matrix into a sparse DIA matrix.

`aoclsparse_csr2dia` converts a CSR matrix into an DIA matrix. It is assumed, that `dia_val` and `dia_offset` are allocated. Allocation size is computed by the number of rows times the number of diagonals. The number of DIA diagonals is obtained by `aoclsparse_csr2dia_ndiag()`.

#### Parameters

in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of cols of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of $m+1$ elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array containing the column indices of the sparse CSR matrix.
in	<i>csr_val</i>	array containing the values of the sparse CSR matrix.
in	<i>dia_num_diag</i>	number of diagonals in ELL storage format.
out	<i>dia_offset</i>	array of <code>dia_num_diag</code> elements containing the diagonal offsets from main diagonal.
out	<i>dia_val</i>	array of $m$ times <code>dia_num_diag</code> elements of the sparse DIA matrix.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_handle</i>	the library context was not initialized.
<i>aoclsparse_status_invalid_size</i>	$m$ or <code>ell_width</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>csr_val</code> , <code>csr_row_ptr</code> , <code>csr_col_ind</code> , <code>ell_val</code> or <code>ell_col_ind</code> pointer is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsr2ell (aoclsparse\_int *m*, const aoclsparse\_int \* *csr\_row\_ptr*, const aoclsparse\_int \* *csr\_col\_ind*, const double \* *csr\_val*, aoclsparse\_int \* *ell\_col\_ind*, double \* *ell\_val*, aoclsparse\_int *ell\_width*)**

Convert a sparse CSR matrix into a sparse ELLPACK matrix.

`aoclsparse_csr2ell` converts a CSR matrix into an ELL matrix. It is assumed, that `ell_val` and `ell_col_ind` are allocated. Allocation size is computed by the number of rows times the number of ELL non-zero elements per row, such that  $nnz_{ELL} = m \cdot ell\_width$ . The number of ELL non-zero elements per row is obtained by `aoclsparse_csr2ell_width()`.

#### Parameters

in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>csr_val</i>	array containing the values of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of $m+1$ elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array containing the column indices of the sparse CSR matrix.
in	<i>ell_width</i>	number of non-zero elements per row in ELL storage format.
out	<i>ell_val</i>	array of $m$ times <code>ell_width</code> elements of the sparse ELL matrix.
out	<i>ell_col_ind</i>	array of $m$ times <code>ell_width</code> elements containing the column indices of the sparse ELL matrix.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_handle</i>	the library context was not initialized.
<i>aoclsparse_status_invalid_size</i>	$m$ or <code>ell_width</code> is invalid.

<i>aoclsparse_status_</i> <i>invalid_pointer</i>	<i>csr_val</i> , <i>csr_row_ptr</i> , <i>csr_col_ind</i> , <i>ell_val</i> or <i>ell_col_ind</i> pointer is invalid.
---	--

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsr2bsr (aoclsparse\_int *m*, aoclsparse\_int *n*, const float \* *csr\_val*, const aoclsparse\_int \* *csr\_row\_ptr*, const aoclsparse\_int \* *csr\_col\_ind*, aoclsparse\_int *block\_dim*, float \* *bsr\_val*, aoclsparse\_int \* *bsr\_row\_ptr*, aoclsparse\_int \* *bsr\_col\_ind*)**

Convert a sparse CSR matrix into a sparse BSR matrix.

*aoclsparse\_scsr2bsr* converts a CSR matrix into a BSR matrix. It is assumed, that *bsr\_val*, *bsr\_col\_ind* and *bsr\_row\_ptr* are allocated. Allocation size for *bsr\_row\_ptr* is computed as *mb*+1 where *mb* is the number of block rows in the BSR matrix. Allocation size for *bsr\_val* and *bsr\_col\_ind* is computed using *csr2bsr\_nnz()* which also fills in *bsr\_row\_ptr*.

#### Parameters

in	<i>m</i>	number of rows in the sparse CSR matrix.
in	<i>n</i>	number of columns in the sparse CSR matrix.
in	<i>csr_val</i>	array of <i>nnz</i> elements containing the values of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <i>m</i> +1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of <i>nnz</i> elements containing the column indices of the sparse CSR matrix.
in	<i>block_dim</i>	size of the blocks in the sparse BSR matrix.
out	<i>bsr_val</i>	array of <i>nnzb</i> * <i>block_dim</i> * <i>block_dim</i> containing the values of the sparse BSR matrix.
out	<i>bsr_row_ptr</i>	array of <i>mb</i> +1 elements that point to the start of every block row of the sparse BSR matrix.
out	<i>bsr_col_ind</i>	array of <i>nnzb</i> elements containing the block column indices of the sparse BSR matrix.

#### Return values

<i>aoclsparse_status_</i> <i>success</i>	the operation completed successfully.
<i>aoclsparse_status_</i> <i>invalid_size</i>	<i>m</i> or <i>n</i> or <i>block_dim</i> is invalid.
<i>aoclsparse_status_</i> <i>invalid_pointer</i>	<i>bsr_val</i> , <i>bsr_row_ptr</i> , <i>bsr_col_ind</i> , <i>csr_val</i> , <i>csr_row_ptr</i> or <i>csr_col_ind</i> pointer is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsr2csc (aoclsparse\_int *m*, aoclsparse\_int *n*, aoclsparse\_int *nnz*, const aoclsparse\_int \* *csr\_row\_ptr*, const aoclsparse\_int \* *csr\_col\_ind*, const float \* *csr\_val*, aoclsparse\_int \* *csc\_row\_ind*, aoclsparse\_int \* *csc\_col\_ptr*, float \* *csc\_val*)**

Convert a sparse CSR matrix into a sparse CSC matrix.

*aoclsparse\_scsr2csc* converts a CSR matrix into a CSC matrix. *aoclsparse\_scsr2csc* can also be used to convert a CSC matrix into a CSR matrix.

#### Note

The resulting matrix can also be seen as the transpose of the input matrix.

#### Parameters

in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of columns of the sparse CSR matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse CSR matrix.

in	<i>csr_val</i>	array of nnz elements of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of m+1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of nnz elements containing the column indices of the sparse CSR matrix.
out	<i>csc_val</i>	array of nnz elements of the sparse CSC matrix.
out	<i>csc_row_ind</i>	array of nnz elements containing the row indices of the sparse CSC matrix.
out	<i>csc_col_ptr</i>	array of n+1 elements that point to the start of every column of the sparse CSC matrix. <code>aoclsparse_csr2csc_buffer_size()</code> .

**Return values**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	m, n or nnz is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>csr_val</i> , <i>csr_row_ptr</i> , <i>csr_col_ind</i> , <i>csc_val</i> , <i>csc_row_ind</i> , <i>csc_col_ptr</i> is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsr2dense (aoclsparse\_int m, aoclsparse\_int n, const aoclsparse\_mat\_descr descr, const float \* csr\_val, const aoclsparse\_int \* csr\_row\_ptr, const aoclsparse\_int \* csr\_col\_ind, float \* A, aoclsparse\_int ld, aoclsparse\_order order)**

This function converts the sparse matrix in CSR format into a dense matrix.

**Parameters**

in	<i>m</i>	number of rows of the dense matrix A .
in	<i>n</i>	number of columns of the dense matrix A .
in	<i>descr</i>	the descriptor of the dense matrix A , the supported matrix type is <b>aoclsparse_matrix_type_general</b> and also any valid value of the <b>aoclsparse_index_base</b> .
in	<i>csr_val</i>	array of nnz ( = <i>csr_row_ptr</i> [m] - <i>csr_row_ptr</i> [0] ) nonzero elements of matrix A .
in	<i>csr_row_ptr</i>	integer array of m+1 elements that contains the start of every row and the end of the last row plus one.
in	<i>csr_col_ind</i>	integer array of nnz ( = <i>csr_row_ptr</i> [m] - <i>csr_row_ptr</i> [0] ) column indices of the non-zero elements of matrix A .
out	<i>A</i>	array of dimensions (ld , n )
out	<i>ld</i>	leading dimension of dense array A .
in	<i>order</i>	memory layout of a dense matrix A .

**Return values**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	m or n or ld is invalid.
<i>aoclsparse_status_invalid_pointer</i>	A or <i>csr_val</i> <i>csr_row_ptr</i> or <i>csr_col_ind</i> pointer is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsr2dia (aoclsparse\_int m, aoclsparse\_int n, const aoclsparse\_int \* csr\_row\_ptr, const aoclsparse\_int \* csr\_col\_ind, const float \* csr\_val, aoclsparse\_int dia\_num\_diag, aoclsparse\_int \* dia\_offset, float \* dia\_val)**

Convert a sparse CSR matrix into a sparse DIA matrix.

`aoclsparse_csr2dia` converts a CSR matrix into an DIA matrix. It is assumed, that `dia_val` and `dia_offset` are allocated. Allocation size is computed by the number of rows times the number of diagonals. The number of DIA diagonals is obtained by `aoclsparse_csr2dia_ndiag()`.

#### Parameters

in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of cols of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of $m+1$ elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array containing the column indices of the sparse CSR matrix.
in	<i>csr_val</i>	array containing the values of the sparse CSR matrix.
in	<i>dia_num_diag</i>	number of diagonals in ELL storage format.
out	<i>dia_offset</i>	array of <code>dia_num_diag</code> elements containing the diagonal offsets from main diagonal.
out	<i>dia_val</i>	array of $m$ times <code>dia_num_diag</code> elements of the sparse DIA matrix.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_handle</i>	the library context was not initialized.
<i>aoclsparse_status_invalid_size</i>	$m$ or <code>ell_width</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>csr_val</code> , <code>csr_row_ptr</code> , <code>csr_col_ind</code> , <code>ell_val</code> or <code>ell_col_ind</code> pointer is invalid.

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsr2ell (aoclsparse\_int *m*, const aoclsparse\_int \* *csr\_row\_ptr*, const aoclsparse\_int \* *csr\_col\_ind*, const float \* *csr\_val*, aoclsparse\_int \* *ell\_col\_ind*, float \* *ell\_val*, aoclsparse\_int *ell\_width*)**

Convert a sparse CSR matrix into a sparse ELLPACK matrix.

`aoclsparse_csr2ell` converts a CSR matrix into an ELL matrix. It is assumed, that `ell_val` and `ell_col_ind` are allocated. Allocation size is computed by the number of rows times the number of ELL non-zero elements per row, such that  $nnz_{ELL} = m \cdot ell\_width$ . The number of ELL non-zero elements per row is obtained by `aoclsparse_csr2ell_width()`.

#### Parameters

in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>csr_val</i>	array containing the values of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of $m+1$ elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array containing the column indices of the sparse CSR matrix.
in	<i>ell_width</i>	number of non-zero elements per row in ELL storage format.
out	<i>ell_val</i>	array of $m$ times <code>ell_width</code> elements of the sparse ELL matrix.
out	<i>ell_col_ind</i>	array of $m$ times <code>ell_width</code> elements containing the column indices of the sparse ELL matrix.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_handle</i>	the library context was not initialized.
<i>aoclsparse_status_invalid_size</i>	$m$ or <code>ell_width</code> is invalid.

<i>aoclsparse_status_</i> <i>invalid_pointer</i>	csr_val, csr_row_ptr, csr_col_ind, ell_val or ell_col_ind pointer is invalid.
---	--

## Sparse Level 2 & 3 Functions

**aoclsparse\_functions.h** provides Sparse Linear Algebra Subprograms of Level 1, 2 and 3, for AMD CPU hardware.

### Functions

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsrmv** (**aoclsparse\_operation** trans, const float \*alpha, **aoclsparse\_int** m, **aoclsparse\_int** n, **aoclsparse\_int** nnz, const float \*csr\_val, const **aoclsparse\_int** \*csr\_col\_ind, const **aoclsparse\_int** \*csr\_row\_ptr, const **aoclsparse\_mat\_descr** descr, const float \*x, const float \*beta, float \*y)

*Single & Double precision sparse matrix vector multiplication using CSR storage format.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsmv** (**aoclsparse\_operation** trans, const double \*alpha, **aoclsparse\_int** m, **aoclsparse\_int** n, **aoclsparse\_int** nnz, const double \*csr\_val, const **aoclsparse\_int** \*csr\_col\_ind, const **aoclsparse\_int** \*csr\_row\_ptr, const **aoclsparse\_mat\_descr** descr, const double \*x, const double \*beta, double \*y)

*Single & Double precision sparse matrix vector multiplication using CSR storage format.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_sellmv** (**aoclsparse\_operation** trans, const float \*alpha, **aoclsparse\_int** m, **aoclsparse\_int** n, **aoclsparse\_int** nnz, const float \*ell\_val, const **aoclsparse\_int** \*ell\_col\_ind, **aoclsparse\_int** ell\_width, const **aoclsparse\_mat\_descr** descr, const float \*x, const float \*beta, float \*y)

*Single & Double precision sparse matrix vector multiplication using ELL storage format.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dellmv** (**aoclsparse\_operation** trans, const double \*alpha, **aoclsparse\_int** m, **aoclsparse\_int** n, **aoclsparse\_int** nnz, const double \*ell\_val, const **aoclsparse\_int** \*ell\_col\_ind, **aoclsparse\_int** ell\_width, const **aoclsparse\_mat\_descr** descr, const double \*x, const double \*beta, double \*y)

*Single & Double precision sparse matrix vector multiplication using ELL storage format.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_sdiamv** (**aoclsparse\_operation** trans, const float \*alpha, **aoclsparse\_int** m, **aoclsparse\_int** n, **aoclsparse\_int** nnz, const float \*dia\_val, const **aoclsparse\_int** \*dia\_offset, **aoclsparse\_int** dia\_num\_diag, const **aoclsparse\_mat\_descr** descr, const float \*x, const float \*beta, float \*y)

*Single & Double precision sparse matrix vector multiplication using DIA storage format.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_ddiamv** (**aoclsparse\_operation** trans, const double \*alpha, **aoclsparse\_int** m, **aoclsparse\_int** n, **aoclsparse\_int** nnz, const double \*dia\_val, const **aoclsparse\_int** \*dia\_offset, **aoclsparse\_int** dia\_num\_diag, const **aoclsparse\_mat\_descr** descr, const double \*x, const double \*beta, double \*y)

*Single & Double precision sparse matrix vector multiplication using DIA storage format.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_sbsrmv** (**aoclsparse\_operation** trans, const float \*alpha, **aoclsparse\_int** mb, **aoclsparse\_int** nb, **aoclsparse\_int** bsr\_dim, const float \*bsr\_val, const **aoclsparse\_int** \*bsr\_col\_ind, const **aoclsparse\_int** \*bsr\_row\_ptr, const **aoclsparse\_mat\_descr** descr, const float \*x, const float \*beta, float \*y)

*Single & Double precision Sparse matrix vector multiplication using BSR storage format.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dbarmv** (**aoclsparse\_operation** trans, const double \*alpha, **aoclsparse\_int** mb, **aoclsparse\_int** nb, **aoclsparse\_int** bsr\_dim, const double \*bsr\_val, const **aoclsparse\_int** \*bsr\_col\_ind, const **aoclsparse\_int** \*bsr\_row\_ptr, const **aoclsparse\_mat\_descr** descr, const double \*x, const double \*beta, double \*y)

*Single & Double precision Sparse matrix vector multiplication using BSR storage format.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsrv** (aoclsparse\_operation trans, const float \*alpha, aoclsparse\_int m, const float \*csr\_val, const aoclsparse\_int \*csr\_col\_ind, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_mat\_descr descr, const float \*x, float \*y)

*Sparse triangular solve using CSR storage format for single and double data precisions.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcscrsv** (aoclsparse\_operation trans, const double \*alpha, aoclsparse\_int m, const double \*csr\_val, const aoclsparse\_int \*csr\_col\_ind, const aoclsparse\_int \*csr\_row\_ptr, const aoclsparse\_mat\_descr descr, const double \*x, double \*y)

*Sparse triangular solve using CSR storage format for single and double data precisions.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsrmm** (aoclsparse\_operation trans\_A, const float \*alpha, const aoclsparse\_mat\_csr csr, const aoclsparse\_mat\_descr descr, aoclsparse\_order order, const float \*B, aoclsparse\_int n, aoclsparse\_int ldb, const float \*beta, float \*C, aoclsparse\_int ldc)

*Sparse matrix dense matrix multiplication using CSR storage format.*

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcscrmm** (aoclsparse\_operation trans\_A, const double \*alpha, const aoclsparse\_mat\_csr csr, const aoclsparse\_mat\_descr descr, aoclsparse\_order order, const double \*B, aoclsparse\_int n, aoclsparse\_int ldb, const double \*beta, double \*C, aoclsparse\_int ldc)

*Sparse matrix dense matrix multiplication using CSR storage format.*

---

## Detailed Description

**aoclsparse\_functions.h** provides Sparse Linear Algebra Subprograms of Level 1, 2 and 3, for AMD CPU hardware.

---

## Function Documentation

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dbstrmv** (aoclsparse\_operation trans, const double \* alpha, aoclsparse\_int mb, aoclsparse\_int nb, aoclsparse\_int bsr\_dim, const double \* bsr\_val, const aoclsparse\_int \* bsr\_col\_ind, const aoclsparse\_int \* bsr\_row\_ptr, const aoclsparse\_mat\_descr descr, const double \* x, const double \* beta, double \* y)

Single & Double precision Sparse matrix vector multiplication using BSR storage format.

**aoclsparse\_bstrmv** multiplies the scalar  $\alpha$  with a sparse  $(mb \cdot bsr\_dim) \times (nb \cdot bsr\_dim)$  matrix, defined in BSR storage format, and the dense vector  $x$  and adds the result to the dense vector  $y$  that is multiplied by the scalar  $\beta$ , such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if } trans == aoclsparse\_operation\_none \\ A^T, & \text{if } trans == aoclsparse\_operation\_transpose \\ A^H, & \text{if } trans == aoclsparse\_operation\_conjugate\_transpose \end{cases}$$



**Note**

Currently, only `trans == aoclsparse_operation_none` is supported.

**Parameters**

in	<i>trans</i>	matrix operation type.
in	<i>mb</i>	number of block rows of the sparse BSR matrix.
in	<i>nb</i>	number of block columns of the sparse BSR matrix.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>descr</i>	descriptor of the sparse BSR matrix. Currently, only <b>aoclsparse_matrix_type_general</b> is supported.
in	<i>bsr_val</i>	array of <code>nnzb</code> blocks of the sparse BSR matrix.
in	<i>bsr_row_ptr</i>	array of <code>mb+1</code> elements that point to the start of every block row of the sparse BSR matrix.
in	<i>bsr_col_ind</i>	array of <code>nnz</code> containing the block column indices of the sparse BSR matrix.
in	<i>bsr_dim</i>	block dimension of the sparse BSR matrix.
in	<i>x</i>	array of <code>nb*bsr_dim</code> elements ( $op(A) = A$ ) or <code>mb*bsr_dim</code> elements ( $op(A) = A^T$ or $op(A) = A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in,out	<i>y</i>	array of <code>mb*bsr_dim</code> elements ( $op(A) = A$ ) or <code>nb*bsr_dim</code> elements ( $op(A) = A^T$ or $op(A) = A^H$ ).

**Return values**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_handle</i>	the library context was not initialized.
<i>aoclsparse_status_invalid_size</i>	<code>mb</code> , <code>nb</code> , <code>nnzb</code> or <code>bsr_dim</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr</code> , <code>alpha</code> , <code>bsr_val</code> , <code>bsr_row_ind</code> , <code>bsr_col_ind</code> , <code>x</code> , <code>beta</code> or <code>y</code> pointer is invalid.
<i>aoclsparse_status_arch_mismatch</i>	the device is not supported.
<i>aoclsparse_status_not_implemented</i>	<code>trans != aoclsparse_operation_none</code> or <code>aoclsparse_matrix_type != aoclsparse_matrix_type_general</code> .

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsmm (aoclsparse\_operation *trans\_A*, const double \* *alpha*, const aoclsparse\_mat\_csr *csr*, const aoclsparse\_mat\_descr *descr*, aoclsparse\_order *order*, const double \* *B*, aoclsparse\_int *n*, aoclsparse\_int *ldb*, const double \* *beta*, double \* *C*, aoclsparse\_int *ldc*)**

Sparse matrix dense matrix multiplication using CSR storage format.

`aoclsparse_dcsmm` multiplies the scalar  $\alpha$  with a sparse  $m \times k$  matrix  $A$ , defined in CSR storage format, and the dense  $k \times n$  matrix  $B$  and adds the result to the dense  $m \times n$  matrix  $C$  that is multiplied by the scalar  $\beta$ , such that

$$C := \alpha \cdot op(A) \cdot B + \beta \cdot C,$$

with

$$op(A) = \begin{cases} A, & \text{if } trans_A == aoclsparse\_operation\_none \\ A^T, & \text{if } trans_A == aoclsparse\_operation\_transpose \\ A^H, & \text{if } trans_A == aoclsparse\_operation\_conjugate\_transpose \end{cases}$$

```
for(i = 0; i < ldc; ++i)
{
    for(j = 0; j < n; ++j)
    {
        C[i][j] = beta * C[i][j];
```

```

    for(k = csr_row_ptr[i]; k < csr_row_ptr[i + 1]; ++k)
    {
        C[i][j] += alpha * csr_val[k] * B[csr_col_ind[k]][j];
    }
}

```

### Parameters

in	<i>trans_A</i>	matrix <i>A</i> operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>csr</i>	sparse CSR matrix <i>A</i> structure.
in	<i>descr</i>	descriptor of the sparse CSR matrix <i>A</i> . Currently, only <b>aoclsparse_matrix_type_general</b> is supported.
in	<i>order</i>	aoclsparse_order_row/aoclsparse_order_column for dense matrix
in	<i>B</i>	array of dimension $ldb \times n$ or $ldb \times k$ .
in	<i>n</i>	number of columns of the dense matrix <i>B</i> and <i>C</i> .
in	<i>ldb</i>	leading dimension of <i>B</i> , must be at least $\max(1, k)$ ( $op(A) == A$ ) or $\max(1, m)$ ( $op(A) == A^T$ or $op(A) == A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in,out	<i>C</i>	array of dimension $ldc \times n$ .
in	<i>ldc</i>	leading dimension of <i>C</i> , must be at least $\max(1, m)$ ( $op(A) == A$ ) or $\max(1, k)$ ( $op(A) == A^T$ or $op(A) == A^H$ ).

### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<i>m</i> , <i>n</i> , <i>k</i> , <i>nnz</i> , <i>ldb</i> or <i>ldc</i> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> , <i>alpha</i> , <i>csr</i> , <i>B</i> , <i>beta</i> or <i>C</i> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<b>aoclsparse_matrix_type</b> != <b>aoclsparse_matrix_type_general</b> .

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dcsmv (aoclsparse\_operation *trans*, const double \* *alpha*, aoclsparse\_int *m*, aoclsparse\_int *n*, aoclsparse\_int *nnz*, const double \* *csr\_val*, const aoclsparse\_int \* *csr\_col\_ind*, const aoclsparse\_int \* *csr\_row\_ptr*, const aoclsparse\_mat\_descr *descr*, const double \* *x*, const double \* *beta*, double \* *y*)**

Single & Double precision sparse matrix vector multiplication using CSR storage format.

*aoclsparse\_dcsmv* multiplies the scalar  $\alpha$  with a sparse  $m \times n$  matrix, defined in CSR storage format, and the dense vector *x* and adds the result to the dense vector *y* that is multiplied by the scalar  $\beta$ , such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if } trans == aoclsparse\_operation\_none \\ A^T, & \text{if } trans == aoclsparse\_operation\_transpose \\ A^H, & \text{if } trans == aoclsparse\_operation\_conjugate\_transpose \end{cases}$$

```

for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];

    for(j = csr_row_ptr[i]; j < csr_row_ptr[i + 1]; ++j)
    {
        y[i] = y[i] + alpha * csr_val[j] * x[csr_col_ind[j]];
    }
}

```

}

**Note**

Currently, only `trans == aoclsparse_operation_none` is supported. Currently, for `aoclsparse_matrix_type == aoclsparse_matrix_type_symmetric`, only lower triangular matrices are supported.

**Parameters**

in	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of columns of the sparse CSR matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_val</i>	array of <code>nnz</code> elements of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of <code>nnz</code> elements containing the column indices of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <code>m+1</code> elements that point to the start of every row of the sparse CSR matrix.
in	<i>descr</i>	descriptor of the sparse CSR matrix. Currently, only <b><code>aoclsparse_matrix_type_general</code></b> and <b><code>aoclsparse_matrix_type_symmetric</code></b> is supported.
in	<i>x</i>	array of <code>n</code> elements ( $op(A) == A$ ) or <code>m</code> elements ( $op(A) == A^T$ or $op(A) == A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in,out	<i>y</i>	array of <code>m</code> elements ( $op(A) == A$ ) or <code>n</code> elements ( $op(A) == A^T$ or $op(A) == A^H$ ).

**Return values**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<code>m</code> , <code>n</code> or <code>nnz</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr</code> , <code>alpha</code> , <code>csr_val</code> , <code>csr_row_ptr</code> , <code>csr_col_ind</code> , <code>x</code> , <code>beta</code> or <code>y</code> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<code>trans != aoclsparse_operation_none</code> or <code>aoclsparse_matrix_type != aoclsparse_matrix_type_general</code> . <code>aoclsparse_matrix_type != aoclsparse_matrix_type_symmetric</code> .

**Example**

This example performs a sparse matrix vector multiplication in CSR format using additional meta data to improve performance.

```
// Compute y = Ax
aoclsparse_scsvmv(aoclsparse_operation_none,
                  &alpha,
                  m,
                  n,
                  nnz,
                  csr_val,
                  csr_col_ind,
                  csr_row_ptr,
                  descr,
                  x,
                  &beta,
                  y);

// Do more work
// ...
```

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcscsv(aoclsparse_operation trans,
const double * alpha, aoclsparse_int m, const double * csr_val, const aoclsparse_int
```

**\* *csr\_col\_ind*, const *aoclsparse\_int* \* *csr\_row\_ptr*, const *aoclsparse\_mat\_descr* *descr*, const double \* *x*, double \* *y*)**

Sparse triangular solve using CSR storage format for single and double data precisions.

*aoclsparse\_csrsv* solves a sparse triangular linear system of a sparse  $m \times m$  matrix, defined in CSR storage format, a dense solution vector  $y$  and the right-hand side  $x$  that is multiplied by  $\alpha$ , such that

$$op(A) \cdot y = \alpha \cdot x,$$

with

$$op(A) = \begin{cases} A, & \text{if } trans == aoclsparse\_operation\_none \\ A^T, & \text{if } trans == aoclsparse\_operation\_transpose \\ A^H, & \text{if } trans == aoclsparse\_operation\_conjugate\_transpose \end{cases}$$

### Note

Currently, only `trans == aoclsparse_operation_none` is supported.

The input matrix has to be sparse upper or lower triangular matrix with unit or non-unit main diagonal. Matrix has to be sorted. No diagonal element can be omitted from a sparse storage if the solver is called with the non-unit indicator.

### Parameters

in	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>csr_val</i>	array of <code>nnz</code> elements of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <code>m+1</code> elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of <code>nnz</code> elements containing the column indices of the sparse CSR matrix.
in	<i>descr</i>	descriptor of the sparse CSR matrix.
in	<i>x</i>	array of <code>m</code> elements, holding the right-hand side.
out	<i>y</i>	array of <code>m</code> elements, holding the solution.

### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<code>m</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr</code> , <code>alpha</code> , <code>csr_val</code> , <code>csr_row_ptr</code> , <code>csr_col_ind</code> , <code>x</code> or <code>y</code> pointer is invalid.
<i>aoclsparse_status_internal_error</i>	an internal error occurred.
<i>aoclsparse_status_not_implemented</i>	<code>trans == aoclsparse_operation_conjugate_transpose</code> or <code>trans == aoclsparse_operation_transpose</code> or <code>aoclsparse_matrix_type != aoclsparse_matrix_type_general</code> .

**DLL\_PUBLIC *aoclsparse\_status* *aoclsparse\_ddiamv* (*aoclsparse\_operation* *trans*, const double \* *alpha*, *aoclsparse\_int* *m*, *aoclsparse\_int* *n*, *aoclsparse\_int* *nnz*, const double \* *dia\_val*, const *aoclsparse\_int* \* *dia\_offset*, *aoclsparse\_int* *dia\_num\_diag*, const *aoclsparse\_mat\_descr* *descr*, const double \* *x*, const double \* *beta*, double \* *y*)**

Single & Double precision sparse matrix vector multiplication using DIA storage format.

`aoclsparse_diamv` multiplies the scalar  $\alpha$  with a sparse  $m \times n$  matrix, defined in DIA storage format, and the dense vector  $x$  and adds the result to the dense vector  $y$  that is multiplied by the scalar  $\beta$ , such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{iftrans} == \text{aoclsparse\_operation\_none} \\ A^T, & \text{iftrans} == \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{iftrans} == \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

### Note

Currently, only `trans == aoclsparse_operation_none` is supported.

### Parameters

in	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse DIA matrix.
in	<i>n</i>	number of columns of the sparse DIA matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse DIA matrix.
in	<i>descr</i>	descriptor of the sparse DIA matrix. Currently, only <b>aoclsparse_matrix_type_general</b> is supported.
in	<i>dia_val</i>	array that contains the elements of the sparse DIA matrix. Padded elements should be zero.
in	<i>dia_offset</i>	array that contains the offsets of each diagonal of the sparse DIA matrix.
in	<i>dia_num_diag</i>	number of diagonals in the sparse DIA matrix.
in	<i>x</i>	array of $n$ elements ( <code>op(A) == A</code> ) or $m$ elements ( <code>op(A) == A<sup>T</sup></code> or <code>op(A) == A<sup>H</sup></code> ).
in	<i>beta</i>	scalar $\beta$ .
in,out	<i>y</i>	array of $m$ elements ( <code>op(A) == A</code> ) or $n$ elements ( <code>op(A) == A<sup>T</sup></code> or <code>op(A) == A<sup>H</sup></code> ).

### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m, n$ or <code>ell_width</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr, alpha, ell_val, ell_col_ind, x, beta</code> or <code>y</code> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<code>trans != aoclsparse_operation_none</code> or <code>aoclsparse_matrix_type != aoclsparse_matrix_type_general</code> .

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_dellmv (aoclsparse\_operation *trans*, const double \* *alpha*, aoclsparse\_int *m*, aoclsparse\_int *n*, aoclsparse\_int *nnz*, const double \* *ell\_val*, const aoclsparse\_int \* *ell\_col\_ind*, aoclsparse\_int *ell\_width*, const aoclsparse\_mat\_descr *descr*, const double \* *x*, const double \* *beta*, double \* *y*)**

Single & Double precision sparse matrix vector multiplication using ELL storage format.

`aoclsparse_ellmv` multiplies the scalar  $\alpha$  with a sparse  $m \times n$  matrix, defined in ELL storage format, and the dense vector  $x$  and adds the result to the dense vector  $y$  that is multiplied by the scalar  $\beta$ , such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{iftrans} == \text{aoclsparse\_operation\_none} \\ A^T, & \text{iftrans} == \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{iftrans} == \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

```

for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];

    for(p = 0; p < ell_width; ++p)
    {
        idx = p * m + i;

        if((ell_col_ind[idx] >= 0) && (ell_col_ind[idx] < n))
        {
            y[i] = y[i] + alpha * ell_val[idx] * x[ell_col_ind[idx]];
        }
    }
}

```

### Note

Currently, only `trans == aoclsparse_operation_none` is supported.

### Parameters

in	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse ELL matrix.
in	<i>n</i>	number of columns of the sparse ELL matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse ELL matrix.
in	<i>descr</i>	descriptor of the sparse ELL matrix. Currently, only <b>aoclsparse_matrix_type_general</b> is supported.
in	<i>ell_val</i>	array that contains the elements of the sparse ELL matrix. Padded elements should be zero.
in	<i>ell_col_ind</i>	array that contains the column indices of the sparse ELL matrix. Padded column indices should be -1.
in	<i>ell_width</i>	number of non-zero elements per row of the sparse ELL matrix.
in	<i>x</i>	array of $n$ elements ( $op(A) == A$ ) or $m$ elements ( $op(A) == A^T$ or $op(A) == A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in,out	<i>y</i>	array of $m$ elements ( $op(A) == A$ ) or $n$ elements ( $op(A) == A^T$ or $op(A) == A^H$ ).

### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m, n$ or <code>ell_width</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr, alpha, ell_val, ell_col_ind, x, beta</code> or <code>y</code> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<code>trans != aoclsparse_operation_none</code> or <code>aoclsparse_matrix_type != aoclsparse_matrix_type_general</code> .

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_sbsrmv (aoclsparse\_operation *trans*, const float \* *alpha*, aoclsparse\_int *mb*, aoclsparse\_int *nb*, aoclsparse\_int *bsr\_dim*, const float \* *bsr\_val*, const aoclsparse\_int \* *bsr\_col\_ind*, const aoclsparse\_int \* *bsr\_row\_ptr*, const aoclsparse\_mat\_descr *descr*, const float \* *x*, const float \* *beta*, float \* *y*)**

Single & Double precision Sparse matrix vector multiplication using BSR storage format.

`aoclsparse_sbsrmv` multiplies the scalar  $\alpha$  with a sparse  $(mb \cdot bsr\_dim) \times (nb \cdot bsr\_dim)$  matrix, defined in BSR storage format, and the dense vector  $x$  and adds the result to the dense vector  $y$  that is multiplied by the scalar  $\beta$ , such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if } trans == aoclspase\_operation\_none \\ A^T, & \text{if } trans == aoclspase\_operation\_transpose \\ A^H, & \text{if } trans == aoclspase\_operation\_conjugate\_transpose \end{cases}$$

### Note

Currently, only `trans == aoclspase_operation_none` is supported.

### Parameters

in	<i>trans</i>	matrix operation type.
in	<i>mb</i>	number of block rows of the sparse BSR matrix.
in	<i>nb</i>	number of block columns of the sparse BSR matrix.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>descr</i>	descriptor of the sparse BSR matrix. Currently, only <b>aoclspase_matrix_type_general</b> is supported.
in	<i>bsr_val</i>	array of <code>nnzb</code> blocks of the sparse BSR matrix.
in	<i>bsr_row_ptr</i>	array of <code>mb+1</code> elements that point to the start of every block row of the sparse BSR matrix.
in	<i>bsr_col_ind</i>	array of <code>nnz</code> containing the block column indices of the sparse BSR matrix.
in	<i>bsr_dim</i>	block dimension of the sparse BSR matrix.
in	<i>x</i>	array of <code>nb*bsr_dim</code> elements ( $op(A) = A$ ) or <code>mb*bsr_dim</code> elements ( $op(A) = A^T$ or $op(A) = A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in,out	<i>y</i>	array of <code>mb*bsr_dim</code> elements ( $op(A) = A$ ) or <code>nb*bsr_dim</code> elements ( $op(A) = A^T$ or $op(A) = A^H$ ).

### Return values

<i>aoclspase_status_success</i>	the operation completed successfully.
<i>aoclspase_status_invalid_handle</i>	the library context was not initialized.
<i>aoclspase_status_invalid_size</i>	<code>mb</code> , <code>nb</code> , <code>nnzb</code> or <code>bsr_dim</code> is invalid.
<i>aoclspase_status_invalid_pointer</i>	<code>descr</code> , <code>alpha</code> , <code>bsr_val</code> , <code>bsr_row_ind</code> , <code>bsr_col_ind</code> , <code>x</code> , <code>beta</code> or <code>y</code> pointer is invalid.
<i>aoclspase_status_arch_mismatch</i>	the device is not supported.
<i>aoclspase_status_not_implemented</i>	<code>trans != aoclspase_operation_none</code> or <code>aoclspase_matrix_type != aoclspase_matrix_type_general</code> .

**DLL\_PUBLIC aoclspase\_status aoclspase\_scsrmm (aoclspase\_operation *trans\_A*, const float \* *alpha*, const aoclspase\_mat\_csr *csr*, const aoclspase\_mat\_descr *descr*, aoclspase\_order *order*, const float \* *B*, aoclspase\_int *n*, aoclspase\_int *ldb*, const float \* *beta*, float \* *C*, aoclspase\_int *ldc*)**

Sparse matrix dense matrix multiplication using CSR storage format.

`aoclspase_scsrmm` multiplies the scalar  $\alpha$  with a sparse  $m \times k$  matrix  $A$ , defined in CSR storage format, and the dense  $k \times n$  matrix  $B$  and adds the result to the dense  $m \times n$  matrix  $C$  that is multiplied by the scalar  $\beta$ , such that

$$C := \alpha \cdot op(A) \cdot B + \beta \cdot C,$$

with

$$op(A) = \begin{cases} A, & iftrans_A == aoclspase\_operation\_none \\ A^T, & iftrans_A == aoclspase\_operation\_transpose \\ A^H, & iftrans_A == aoclspase\_operation\_conjugate\_transpose \end{cases}$$

```

for(i = 0; i < ldc; ++i)
{
    for(j = 0; j < n; ++j)
    {
        C[i][j] = beta * C[i][j];

        for(k = csr_row_ptr[i]; k < csr_row_ptr[i + 1]; ++k)
        {
            C[i][j] += alpha * csr_val[k] * B[csr_col_ind[k]][j];
        }
    }
}

```

### Parameters

in	<i>trans_A</i>	matrix <i>A</i> operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>csr</i>	sparse CSR matrix <i>A</i> structure.
in	<i>descr</i>	descriptor of the sparse CSR matrix <i>A</i> . Currently, only <b>aoclspase_matrix_type_general</b> is supported.
in	<i>order</i>	aoclspase_order_row/aoclspase_order_column for dense matrix
in	<i>B</i>	array of dimension $ldb \times n$ or $ldb \times k$ .
in	<i>n</i>	number of columns of the dense matrix <i>B</i> and <i>C</i> .
in	<i>ldb</i>	leading dimension of <i>B</i> , must be at least $\max(1, k)$ ( $op(A) == A$ ) or $\max(1, m)$ ( $op(A) == A^T$ or $op(A) == A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in,out	<i>C</i>	array of dimension $ldb \times n$ .
in	<i>ldc</i>	leading dimension of <i>C</i> , must be at least $\max(1, m)$ ( $op(A) == A$ ) or $\max(1, k)$ ( $op(A) == A^T$ or $op(A) == A^H$ ).

### Return values

<i>aoclspase_status_success</i>	the operation completed successfully.
<i>aoclspase_status_invalid_size</i>	<i>m</i> , <i>n</i> , <i>k</i> , <i>nnz</i> , <i>ldb</i> or <i>ldc</i> is invalid.
<i>aoclspase_status_invalid_pointer</i>	<i>descr</i> , <i>alpha</i> , <i>csr</i> , <i>B</i> , <i>beta</i> or <i>C</i> pointer is invalid.
<i>aoclspase_status_not_implemented</i>	<b>aoclspase_matrix_type != aoclspase_matrix_type_general.</b>

**DLL\_PUBLIC aoclspase\_status aoclspase\_scsrmv (aoclspase\_operation *trans*, const float \* *alpha*, aoclspase\_int *m*, aoclspase\_int *n*, aoclspase\_int *nnz*, const float \* *csr\_val*, const aoclspase\_int \* *csr\_col\_ind*, const aoclspase\_int \* *csr\_row\_ptr*, const aoclspase\_mat\_descr *descr*, const float \* *x*, const float \* *beta*, float \* *y*)**

Single & Double precision sparse matrix vector multiplication using CSR storage format.

*aoclspase\_scsrmv* multiplies the scalar  $\alpha$  with a sparse  $m \times n$  matrix, defined in CSR storage format, and the dense vector *x* and adds the result to the dense vector *y* that is multiplied by the scalar  $\beta$ , such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with



$$op(A) = \begin{cases} A, & \text{iftrans} == \text{aoclsparse\_operation\_none} \\ A^T, & \text{iftrans} == \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{iftrans} == \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

```

for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];

    for(j = csr_row_ptr[i]; j < csr_row_ptr[i + 1]; ++j)
    {
        y[i] = y[i] + alpha * csr_val[j] * x[csr_col_ind[j]];
    }
}

```

### Note

Currently, only `trans == aoclsparse_operation_none` is supported. Currently, for `aoclsparse_matrix_type == aoclsparse_matrix_type_symmetric`, only lower triangular matrices are supported.

### Parameters

in	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of columns of the sparse CSR matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_val</i>	array of <code>nnz</code> elements of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of <code>nnz</code> elements containing the column indices of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <code>m+1</code> elements that point to the start of every row of the sparse CSR matrix.
in	<i>descr</i>	descriptor of the sparse CSR matrix. Currently, only <b>aoclsparse_matrix_type_general</b> and <b>aoclsparse_matrix_type_symmetric</b> is supported.
in	<i>x</i>	array of <code>n</code> elements ( $op(A) == A$ ) or <code>m</code> elements ( $op(A) == A^T$ or $op(A) == A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in,out	<i>y</i>	array of <code>m</code> elements ( $op(A) == A$ ) or <code>n</code> elements ( $op(A) == A^T$ or $op(A) == A^H$ ).

### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<code>m</code> , <code>n</code> or <code>nnz</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr</code> , <code>alpha</code> , <code>csr_val</code> , <code>csr_row_ptr</code> , <code>csr_col_ind</code> , <code>x</code> , <code>beta</code> or <code>y</code> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<code>trans != aoclsparse_operation_none</code> or <code>aoclsparse_matrix_type != aoclsparse_matrix_type_general</code> . <code>aoclsparse_matrix_type != aoclsparse_matrix_type_symmetric</code> .

### Example

This example performs a sparse matrix vector multiplication in CSR format using additional meta data to improve performance.

```

// Compute y = Ax
aoclsparse_scsrmmv(aoclsparse_operation_none,
                  &alpha,
                  m,
                  n,
                  nnz,
                  csr_val,

```

```

        csr_col_ind,
        csr_row_ptr,
        descr,
        x,
        &beta,
        y);

// Do more work
// ...

```

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_scsv (aoclsparse\_operation *trans*, const float \* *alpha*, aoclsparse\_int *m*, const float \* *csr\_val*, const aoclsparse\_int \* *csr\_col\_ind*, const aoclsparse\_int \* *csr\_row\_ptr*, const aoclsparse\_mat\_descr *descr*, const float \* *x*, float \* *y*)**

Sparse triangular solve using CSR storage format for single and double data precisions.

`aoclsparse_scsv` solves a sparse triangular linear system of a sparse  $m \times m$  matrix, defined in CSR storage format, a dense solution vector  $y$  and the right-hand side  $x$  that is multiplied by  $\alpha$ , such that

$$op(A) \cdot y = \alpha \cdot x,$$

with

$$op(A) = \begin{cases} A, & \text{if } trans == \text{aoclsparse\_operation\_none} \\ A^T, & \text{if } trans == \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if } trans == \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

#### Note

Currently, only `trans == aoclsparse_operation_none` is supported.

The input matrix has to be sparse upper or lower triangular matrix with unit or non-unit main diagonal. Matrix has to be sorted. No diagonal element can be omitted from a sparse storage if the solver is called with the non-unit indicator.

#### Parameters

in	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>csr_val</i>	array of nnz elements of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of m+1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of nnz elements containing the column indices of the sparse CSR matrix.
in	<i>descr</i>	descriptor of the sparse CSR matrix.
in	<i>x</i>	array of m elements, holding the right-hand side.
out	<i>y</i>	array of m elements, holding the solution.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	m is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> , <i>alpha</i> , <i>csr_val</i> , <i>csr_row_ptr</i> , <i>csr_col_ind</i> , <i>x</i> or <i>y</i> pointer is invalid.
<i>aoclsparse_status_internal_error</i>	an internal error occurred.
<i>aoclsparse_status_not_implemented</i>	<code>trans == aoclsparse_operation_conjugate_transpose</code> or <code>trans == aoclsparse_operation_transpose</code> or <code>aoclsparse_matrix_type != aoclsparse_matrix_type_general</code> .

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_sdiamv (aoclsparse\_operation *trans*, const float \* *alpha*, aoclsparse\_int *m*, aoclsparse\_int *n*, aoclsparse\_int *nnz*, const float \* *dia\_val*, const aoclsparse\_int \* *dia\_offset*, aoclsparse\_int *dia\_num\_diag*, const aoclsparse\_mat\_descr *descr*, const float \* *x*, const float \* *beta*, float \* *y*)**

Single & Double precision sparse matrix vector multiplication using DIA storage format.

*aoclsparse\_diamv* multiplies the scalar  $\alpha$  with a sparse  $m \times n$  matrix, defined in DIA storage format, and the dense vector  $x$  and adds the result to the dense vector  $y$  that is multiplied by the scalar  $\beta$ , such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if } trans == aoclsparse\_operation\_none \\ A^T, & \text{if } trans == aoclsparse\_operation\_transpose \\ A^H, & \text{if } trans == aoclsparse\_operation\_conjugate\_transpose \end{cases}$$

### Note

Currently, only *trans* == **aoclsparse\_operation\_none** is supported.

### Parameters

in	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse DIA matrix.
in	<i>n</i>	number of columns of the sparse DIA matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse DIA matrix.
in	<i>descr</i>	descriptor of the sparse DIA matrix. Currently, only <b>aoclsparse_matrix_type_general</b> is supported.
in	<i>dia_val</i>	array that contains the elements of the sparse DIA matrix. Padded elements should be zero.
in	<i>dia_offset</i>	array that contains the offsets of each diagonal of the sparse DIA matrix.
in	<i>dia_num_diag</i>	number of diagonals in the sparse DIA matrix.
in	<i>x</i>	array of $n$ elements ( $op(A) == A$ ) or $m$ elements ( $op(A) == A^T$ or $op(A) == A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in,out	<i>y</i>	array of $m$ elements ( $op(A) == A$ ) or $n$ elements ( $op(A) == A^T$ or $op(A) == A^H$ ).

### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m, n$ or <i>ell_width</i> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr, alpha, ell_val, ell_col_ind, x, beta</i> or <i>y</i> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<i>trans</i> != <b>aoclsparse_operation_none</b> or <b>aoclsparse_matrix_type</b> != <b>aoclsparse_matrix_type_general</b> .

**DLL\_PUBLIC aoclsparse\_status aoclsparse\_sellmv (aoclsparse\_operation *trans*, const float \* *alpha*, aoclsparse\_int *m*, aoclsparse\_int *n*, aoclsparse\_int *nnz*, const float \* *ell\_val*, const aoclsparse\_int \* *ell\_col\_ind*, aoclsparse\_int *ell\_width*, const aoclsparse\_mat\_descr *descr*, const float \* *x*, const float \* *beta*, float \* *y*)**

Single & Double precision sparse matrix vector multiplication using ELL storage format.

`aoclsparse_ellmv` multiplies the scalar  $\alpha$  with a sparse  $m \times n$  matrix, defined in ELL storage format, and the dense vector  $x$  and adds the result to the dense vector  $y$  that is multiplied by the scalar  $\beta$ , such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{iftrans} == \text{aoclsparse\_operation\_none} \\ A^T, & \text{iftrans} == \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{iftrans} == \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

```
for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];

    for(p = 0; p < ell_width; ++p)
    {
        idx = p * m + i;

        if((ell_col_ind[idx] >= 0) && (ell_col_ind[idx] < n))
        {
            y[i] = y[i] + alpha * ell_val[idx] * x[ell_col_ind[idx]];
        }
    }
}
```

#### Note

Currently, only `trans == aoclsparse_operation_none` is supported.

#### Parameters

in	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse ELL matrix.
in	<i>n</i>	number of columns of the sparse ELL matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse ELL matrix.
in	<i>descr</i>	descriptor of the sparse ELL matrix. Currently, only <b>aoclsparse_matrix_type_general</b> is supported.
in	<i>ell_val</i>	array that contains the elements of the sparse ELL matrix. Padded elements should be zero.
in	<i>ell_col_ind</i>	array that contains the column indices of the sparse ELL matrix. Padded column indices should be -1.
in	<i>ell_width</i>	number of non-zero elements per row of the sparse ELL matrix.
in	<i>x</i>	array of $n$ elements ( $op(A) == A$ ) or $m$ elements ( $op(A) == A^T$ or $op(A) == A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in,out	<i>y</i>	array of $m$ elements ( $op(A) == A$ ) or $n$ elements ( $op(A) == A^T$ or $op(A) == A^H$ ).

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m, n$ or <code>ell_width</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr, alpha, ell_val, ell_col_ind, x, beta</code> or <code>y</code> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<code>trans != aoclsparse_operation_none</code> or <code>aoclsparse_matrix_type != aoclsparse_matrix_type_general</code> .

## aoclsparse\_types.h File Reference

**aoclsparse\_types.h** defines data types used by aoclsparse

### Macros

**#define DLL\_PUBLIC** \_\_attribute\_\_((\_\_visibility\_\_("default")))  
*Macro for function attribute.*

### Typedefs

**typedef int32\_t aoclsparse\_int**  
*Specifies whether int32 or int64 is used.*

**typedef struct \_aoclsparse\_mat\_descr \* aoclsparse\_mat\_descr**  
*Descriptor of the matrix.*

**typedef struct \_aoclsparse\_mat\_csr \* aoclsparse\_mat\_csr**  
*CSR matrix storage format.*

**typedef enum aoclsparse\_operation\_ aoclsparse\_operation**  
*Specify whether the matrix is to be transposed or not.*

**typedef enum aoclsparse\_index\_base\_ aoclsparse\_index\_base**  
*Specify the matrix index base.*

**typedef enum aoclsparse\_matrix\_type\_ aoclsparse\_matrix\_type**  
*Specify the matrix type.*

**typedef enum aoclsparse\_diag\_type\_ aoclsparse\_diag\_type**  
*Indicates if the diagonal entries are unity.*

**typedef enum aoclsparse\_fill\_mode\_ aoclsparse\_fill\_mode**  
*Specify the matrix fill mode.*

**typedef enum aoclsparse\_order\_ aoclsparse\_order**  
*List of dense matrix ordering.*

**typedef enum aoclsparse\_status\_ aoclsparse\_status**  
*List of aoclsparse status codes definition.*

### Enumerations

**enum aoclsparse\_operation\_ { aoclsparse\_operation\_none = 111, aoclsparse\_operation\_transpose = 112, aoclsparse\_operation\_conjugate\_transpose = 113 }**  
*Specify whether the matrix is to be transposed or not.*

**enum aoclsparse\_index\_base\_ { aoclsparse\_index\_base\_zero = 0, aoclsparse\_index\_base\_one = 1 }**  
*Specify the matrix index base.*

```
enum aoclsparse_matrix_type_ { aoclsparse_matrix_type_general = 0,
    aoclsparse_matrix_type_symmetric = 1, aoclsparse_matrix_type_hermitian = 2,
    aoclsparse_matrix_type_triangular = 3 }
    Specify the matrix type.

enum aoclsparse_diag_type_ { aoclsparse_diag_type_non_unit = 0, aoclsparse_diag_type_unit = 1
    }
    Indicates if the diagonal entries are unity.

enum aoclsparse_fill_mode_ { aoclsparse_fill_mode_lower = 0, aoclsparse_fill_mode_upper = 1 }
    Specify the matrix fill mode.

enum aoclsparse_order_ { aoclsparse_order_row = 0, aoclsparse_order_column = 1 }
    List of dense matrix ordering.

enum aoclsparse_status_ { aoclsparse_status_success = 0, aoclsparse_status_not_implemented =
    1, aoclsparse_status_invalid_pointer = 2, aoclsparse_status_invalid_size = 3,
    aoclsparse_status_internal_error = 4, aoclsparse_status_invalid_value = 5 }
    List of aoclsparse status codes definition.
```

---

## Detailed Description

**aoclsparse\_types.h** defines data types used by aoclsparse

---

## Macro Definition Documentation

```
#define DLL_PUBLIC __attribute__((__visibility__("default")))
```

Macro for function attribute.

The macro specifies visibility attribute of public functions

---

## Typedef Documentation

```
typedef enum aoclsparse_diag_type_ aoclsparse_diag_type
```

Indicates if the diagonal entries are unity.

The **aoclsparse\_diag\_type** indicates whether the diagonal entries of a matrix are unity or not. If **aoclsparse\_diag\_type\_unit** is specified, all present diagonal values will be ignored. For a given **aoclsparse\_mat\_descr**, the **aoclsparse\_diag\_type** can be set using **aoclsparse\_set\_mat\_diag\_type()**. The current **aoclsparse\_diag\_type** of a matrix can be obtained by **aoclsparse\_get\_mat\_diag\_type()**.

```
typedef enum aoclsparse_fill_mode_ aoclsparse_fill_mode
```

Specify the matrix fill mode.

The **aoclsparse\_fill\_mode** indicates whether the lower or the upper part is stored in a sparse triangular matrix. For a given **aoclsparse\_mat\_descr**, the **aoclsparse\_fill\_mode** can be set using **aoclsparse\_set\_mat\_fill\_mode()**. The current **aoclsparse\_fill\_mode** of a matrix can be obtained by **aoclsparse\_get\_mat\_fill\_mode()**.

**typedef enum aoclsparse\_index\_base\_ aoclsparse\_index\_base**

Specify the matrix index base.

The **aoclsparse\_index\_base** indicates the index base of the indices. For a given **aoclsparse\_mat\_descr**, the **aoclsparse\_index\_base** can be set using **aoclsparse\_set\_mat\_index\_base()**. The current **aoclsparse\_index\_base** of a matrix can be obtained by **aoclsparse\_get\_mat\_index\_base()**.

**typedef struct \_aoclsparse\_mat\_csr\* aoclsparse\_mat\_csr**

CSR matrix storage format.

The aoclSPARSE CSR matrix structure holds the CSR matrix. It must be initialized using **aoclsparse\_create\_mat\_csr()** and the returned CSR matrix must be passed to all subsequent library calls that involve the matrix. It should be destroyed at the end using **aoclsparse\_destroy\_mat\_csr()**.

**typedef struct \_aoclsparse\_mat\_descr\* aoclsparse\_mat\_descr**

Descriptor of the matrix.

The aoclSPARSE matrix descriptor is a structure holding all properties of a matrix. It must be initialized using **aoclsparse\_create\_mat\_descr()** and the returned descriptor must be passed to all subsequent library calls that involve the matrix. It should be destroyed at the end using **aoclsparse\_destroy\_mat\_descr()**.

**typedef enum aoclsparse\_matrix\_type\_ aoclsparse\_matrix\_type**

Specify the matrix type.

The **aoclsparse\_matrix\_type** indicates the type of a matrix. For a given **aoclsparse\_mat\_descr**, the **aoclsparse\_matrix\_type** can be set using **aoclsparse\_set\_mat\_type()**. The current **aoclsparse\_matrix\_type** of a matrix can be obtained by **aoclsparse\_get\_mat\_type()**.

**typedef enum aoclsparse\_operation\_ aoclsparse\_operation**

Specify whether the matrix is to be transposed or not.

The **aoclsparse\_operation** indicates the operation performed with the given matrix.

**typedef enum aoclsparse\_order\_ aoclsparse\_order**

List of dense matrix ordering.

This is a list of supported **aoclsparse\_order** types that are used to describe the memory layout of a dense matrix

**typedef enum aoclsparse\_status\_ aoclsparse\_status**

List of aoclsparse status codes definition.

This is a list of the **aoclsparse\_status** types that are used by the aoclSPARSE library.

## Enumeration Type Documentation

### enum aoclsparse\_diag\_type\_

Indicates if the diagonal entries are unity.

The **aoclsparse\_diag\_type** indicates whether the diagonal entries of a matrix are unity or not. If **aoclsparse\_diag\_type\_unit** is specified, all present diagonal values will be ignored. For a given **aoclsparse\_mat\_descr**, the **aoclsparse\_diag\_type** can be set using **aoclsparse\_set\_mat\_diag\_type()**. The current **aoclsparse\_diag\_type** of a matrix can be obtained by **aoclsparse\_get\_mat\_diag\_type()**.

**Enumerator:**

aoclsparse_diag_type_non_unit	diagonal entries are non-unity.
aoclsparse_diag_type_unit	diagonal entries are unity

### enum aoclsparse\_fill\_mode\_

Specify the matrix fill mode.

The **aoclsparse\_fill\_mode** indicates whether the lower or the upper part is stored in a sparse triangular matrix. For a given **aoclsparse\_mat\_descr**, the **aoclsparse\_fill\_mode** can be set using **aoclsparse\_set\_mat\_fill\_mode()**. The current **aoclsparse\_fill\_mode** of a matrix can be obtained by **aoclsparse\_get\_mat\_fill\_mode()**.

**Enumerator:**

aoclsparse_fill_mode_lower	lower triangular part is stored.
aoclsparse_fill_mode_upper	upper triangular part is stored.

### enum aoclsparse\_index\_base\_

Specify the matrix index base.

The **aoclsparse\_index\_base** indicates the index base of the indices. For a given **aoclsparse\_mat\_descr**, the **aoclsparse\_index\_base** can be set using **aoclsparse\_set\_mat\_index\_base()**. The current **aoclsparse\_index\_base** of a matrix can be obtained by **aoclsparse\_get\_mat\_index\_base()**.

**Enumerator:**

aoclsparse_index_base_zero	zero based indexing.
aoclsparse_index_base_one	one based indexing.

### enum aoclsparse\_matrix\_type\_



Specify the matrix type.

The **aoclsparse\_matrix\_type** indices the type of a matrix. For a given **aoclsparse\_mat\_descr**, the **aoclsparse\_matrix\_type** can be set using **aoclsparse\_set\_mat\_type()**. The current **aoclsparse\_matrix\_type** of a matrix can be obtained by **aoclsparse\_get\_mat\_type()**.

**Enumerator:**

aoclsparse_matrix_type_general	general matrix type.
aoclsparse_matrix_type_symmetric	symmetric matrix type.
aoclsparse_matrix_type_hermitian	hermitian matrix type.
aoclsparse_matrix_type_triangular	triangular matrix type.

**enum aoclsparse\_operation\_**

Specify whether the matrix is to be transposed or not.

The **aoclsparse\_operation** indicates the operation performed with the given matrix.

**Enumerator:**

aoclsparse_operation_none	Operate with matrix.
aoclsparse_operation_transpose	Operate with transpose.
aoclsparse_operation_conjugate_transpose	Operate with conj. transpose.

**enum aoclsparse\_order\_**

List of dense matrix ordering.

This is a list of supported **aoclsparse\_order** types that are used to describe the memory layout of a dense matrix

**Enumerator:**

aoclsparse_order_row	Row major.
aoclsparse_order_column	Column major.

**enum aoclsparse\_status\_**

List of aoclsparse status codes definition.

This is a list of the **aoclsparse\_status** types that are used by the aoclSPARSE library.

**Enumerator:**

aoclsparse_status_success	success.
aoclsparse_status_not_implemented	function is not implemented.
aoclsparse_status_invalid_pointer	invalid pointer parameter.
aoclsparse_status_invalid_size	invalid size parameter.
aoclsparse_status_internal_error	other internal library failure.
aoclsparse_status_invalid_value	invalid value parameter.

