ArrayFire C++ API Reference Card

available here

Array Class Constructors

- array()
- array(const af_array handle)
- array(dim_type d0, af_dtype ty = f32)
- array(dim_type d0, dim_type d1, af_dtype ty = f32)
- array(dim_type d0, dim_type d1, dim_type d2, af_dtype ty = f32)
- array(dim_type d0, dim_type d1, dim_type d2, dim_type d3, af_dtype ty = f32)
- array(const dim4& dims, af_dtype ty = f32)
- array(const dim4& dims, const T *ptr, af_source_t src=afHost)
- array(dim_type d0, const T *ptr, af_source_t src=afHost)
- array(dim_type d0, dim_type d1, const T *ptr, af_source_t src=afHost)
- array(dim_type d0, dim_type d1, dim_type d2, const T *ptr, af_source_t src=afHost)
- array(dim_type d0, dim_type d1, dim_type d2, dim_type d3, const T *ptr, af_source_t src=afHost)

where T is one of the types mentioned in the Types section and ***dim_type*** is a typedef for ***long long***.

Enum af_dtype

- c64 - double precision complex type
- c32 - single precision complex type
- f64 - double
- f32 - float
- s32 - 32-bit int
- u32 - 32-bit unsigned int
- u8 - 8-bit unsigned char
- s8 - 8-bit signed char
- u8x4- quadruple of u8's
- s8x4- quadruple of s8's

Enum af_source_t

To distinguish host pointer from device pointer.

- afHost
- afDevice
- afPinned

Sequence Structure

It is used to index existing ***array***s to create/copy sub-arrays. The keyword ***span*** should be used to index all elements along a given dimension.

```
typedef struct af_seq {
    size_t begin, end;
    int    step;
} af_seq;
```

```
static const af_seq span = {1, 1, 0};
```

Dim4 Structure and its member functions

dim4 structure is used to store ***array*** meta-data.

- dim4(dim_type one, dim_type two=1, dim_type three=1, dim_type four=1)
- dim4(const dim4& other)
- dim_type elements()
- dim_type elements()const
- dim_type ndims()
- dim_type ndims()const
- bool operator==(const dim4& other)const
- bool operator!=(const dim4& other)const
- dim4& operator*=(const dim4& other)
- dim4& operator+=(const dim4& other)
- dim4& operator-=(const dim4& other)
- dim_type& operator[](const unsigned dim)
- const dim_type& operator[](const unsigned dim)const
- dim_type* get()
- const dim_type* get()const

array member functions

- dim4 dims()const
- dim_type dims(unsigned dim)const
- unsigned numdims()const
- size_t bytes()const
- array copy()const
- bool isempty()const
- bool isscalar()const
- bool isvector()const
- bool isrow()const
- bool iscolumn()const
- bool iscomplex()const
- inline bool isreal()const
- void eval()
- void unlock()const
- T scalar()const
- T* device()const
- static void *alloc(size_t, af_dtype)
- static void *pinned(size_t, af_dtype)
- static void free(const void *)
- array& operator=(const array& a)

Built-in Array Generators

```
array F(const dim4 &dims, af_dtype ty=f32);
array F(const dim_type, af_dtype ty=f32);
array F(const dim_type, const dim_type, af_dtype ty=f32);
array F(const dim_type, const dim_type, const dim_type, af_dtype ty=
      f32);
```

where ***F*** is one of the following.

- ***constant<double>*** - Constant value ***array***.
- ***randu*** - Uniform RNG values in the range $[0, 1]$.
- ***randn*** - Normalized RNG values in the range $[0, 1]$ with $\mu = 0$ and $\sigma = 1$.

Set Operations

- array setunique(const array& input)
- void setunique(array& values, array& indices, array& locations, const array& input)
- array setunion(const array& A, const array& B)
- void setunion(array& U, array& IA, array& IB, const array& A, const array& B)
- array setintersect(const array& A, const array& B)
- void setintersect(array& U, array& IA, array& IB, const array& A, const array& B)

Elementwise operations with Single Argument

```
array F(const array&);
```

where ***F*** is one of the following

sin	cos	tan	asin	acos	atan
sinh	cosh	tanh	asinh	acosh	atanh
ceil	floor	round	trunc	factorial	gamma
log2	log10	log1p	expm1	gammaln	epsilon
abs	phase	conjg	real	imag	cmplx
exp	pow2	gError	cmplxGError	invGError	invCmplxGError

Elementwise operations with Two Arguments

```
array F(const array& t0, const array& t1)}
array F(const array& t0, const double t1)}
array F(const double t0, const array& t1)}
```

where ***F*** is one of the following

- *root* - first argument is radicand.
- *hypot*- Euclidean distance.
- *rem* - remainder operation.
- *mod* - modulus operation.
- *cmplx* - create complex array with first argument as real values.

Other Elementwise operations

- array sqrt(const array&, bool isreal=false)
- array log(const array&, bool isreal=false)
- array min(const array&, const array&)
- array min(const double, const array&)
- array max(const array&, const array&)
- array max(const double, const array&)
- array pow(const array &base, const array &power, bool isreal = false)
- array pow(const array &base, const double power, bool isreal =false)
- array pow(const double base, const array &power, bool isreal = false)

Data Analysis

```
array F(const array &input, int dim=-1);
T F(const array &input);
```

where *F* is one of the following

anytrue	sum	min	mean	alltrue	stdev	mul	max	median

Additional data analysis functions are listed below:

- void min(array &values, array &indices, const array &input, int dim=-1)
- void min(T *value, int *index, const array &input)
- void max(array &values, array &indices, const array &input, int dim=-1)
- void max(T *value, int *index, const array &input)
- array minmax_packed(const array &input, int dim=-1)
- void minmax(array &minvalues, array &maxvalues, const array &input, int dim=-1)
- std::pair<T> minmax(const array &input)
- array where (const array &input)
- void where (array &row, array &column, const array &input)
- array segsum (const array &input, const array &segments, bool reduce=false)
- array accum (const array &input, int dim=-1, af_op_t op=sum_t, bool isupcast=false)
- array var (const array &input, bool isbiased=false, int dim=-1)
- array var (const array &input, const array weights, int dim=-1)
- T var (const array &input, bool isbiased=false)
- T var (const array &input, const array weights)
- array cov (const array &input, bool isbiased=false)
- array cov (const array &X, const array &Y, bool isbiased=false)
- T corrcoef (const array &x, const array &y)
- array diff1 (const array &in, int dim=-1)
- array diff2 (const array &in, int dim=-1)
- void grad(array &rows, array &cols, const array &in)
- array sort (const array &in, int dim=-1, bool isdescending=false)
- void sort (array &values, array &indices, const array &in, int dim=-1, bool isdescending=false)
- void sort (array &outkeys, array &outvalues, const array &keys, const array &values, bool isdescending=false)
- array histogram (const array &data, unsigned nbins)
- array histogram (const array &data, unsigned nbins, float min, float max)
- array histequal (const array &data, const array &histogram)

Signal Processing

- array convolve(const array& f1, const array& f2, const array& signal, bool expand=false)
- array convolve(unsigned n1, const T *h_f1, unsigned n2, const T *h_f2, const array& signal, bool expand=false)
- array convolve(const array& signal, const array & filter, bool expand=false)
- array convolve(const array& signal, unsigned ndims, unsigned *dims, const T *h_kernel, bool expand=false)
- array fir(int nb, const float *h_b, const array& x)
- array fir(const array& b, const array& x)
- array iir(int nb, const float *h_b, int na, const float *h_a, const array& x)
- array iir(const array& b, const array& a, const array& x)
- array fft(const array& , int pad=-1)
- array fft2(const array& , int nx=-1, int ny=-1)
- array fft3(const array& , const dim4& dims=dim4(0))
- array ifft(const array& , int pad=-1)
- array ifft2(const array& , int nx=-1, int ny=-1)
- array ifft3(const array& , const dim4& dims=dim4(0))
- array approx1(const array& Y, const array& Xi, char method='L', float offgrid=af::NaN)
- array approx2(const array& Z, const array& Xi, const array& Yi, char method='L', float offgrid=af::NaN)

Image Processing

- array gaussiankernel(int rows, int cols, double sigma_r=0, double sigma_c=0)
- array loadimage(const char *filename, bool iscolor=false)
- void saveimage(const char *filename, const af::array& img)
- array colorspace(const array& image, const af_cspace_t to, const af_cspace_t from)
- array resize(const float scale, const array& image, const char method='L')
- array resize(const float scale_x, const float scale_y, const array& image, const char method='L')
- array resize(const unsigned height, const unsigned width, const array& image, const char method='L')
- array transform(const array& image, const array& transform, const unsigned oh=0, const unsigned ow=0, const bool inverse=true)
- array rotate(const array& image, const float theta=0, const bool iscrop=true, const bool recenter=true)
- array translate(const array& image, const float trans_x, const float trans_y, const unsigned oh=0, const unsigned ow=0)
- array scale(const array& image, const float scale_x, const float scale_y, const unsigned oh=0, const unsigned ow=0)
- array skew(const array& image, const float skew_x, const float skew_y, const unsigned oh=0, const unsigned ow=0, const bool inverse=false)
- array erode(const array& image, const array& mask)
- array dilate(const array& image, const array& mask)
- array erode3d(const array& volume, const array& mask)
- array dilate3d(const array& volume, const array& mask)
- array medfilt(const array& image, const unsigned height=3, const unsigned width=3)
- array minfilt(const array& image, const unsigned height=3, const unsigned width=3)
- array maxfilt(const array& image, const unsigned height=3, const unsigned width=3)
- array bilateral(const array& image, const float space, const float color)
- array meanshift(const array& image, const float space, const float color, const int iter=5)
- array regions(const array& image, const unsigned connectiviT=8)
- array areas(const array& regions)
- array centroids(const array& regions)

Enum af_cspace_t

- AF_RGB - Image in RGB format
- AF_HSV - Image in HSV
- AF_GRAY - Grayscale Image
- AF_YCbCr - Image in YCRCB format

Feature Detection

- cv_features fastFeatureDetector(const array& image, const float thr=20.f, const unsigned arc_length=9, const bool nonmax=true, const float feature_ratio=0.05f);
- array matchTemplate(const array& search_image, const array& template_image, const af_match_t method=AF_SAD);

Structure cv_features

```
class cv_features {
  public:
      cv_features() {}
      ~cv_features() {}
      array x;            ///< feature x-coordinates
      array y;            ///< feature y-coordinates
      array score;        ///< feature score
      array orientation;  ///< feature orientation
      array size;         ///< feature size
};
```

Enum af_match_t

- AF_SAD - Sum of Absolute Differences
- AF_LSAD - Locally Scaled Sum of Absolute Differences
- AF_ZSAD - Zero-mean Sum of Absolute Differences
- AF_SSD - Sum of Squared Differences
- AF_LSSD - Locally Acaled Sum of Squared Differences
- AF_ZSSD - Zero-mean Sum of Squared Differences
- AF_NCC - Normalized Cross Correlation
- AF_ZNCC - Zero-mean Normalized Cross Correlation
- AF_SHD - Sum of Hamming Distances

Linear Algebra

- T dot(const array&, const array&)
- array matmul(const array &, const array &, af_matprop_t=af_none, af_matprop_t=af_none)
- array matmul(const array &, const array &, const array &)
- array matmul(const array &, const array &, const array &, const array &)
- array solve(const array&, const array&, af_solve_t options = af_solve_none)
- array lu(const array& in)
- void lu(array& lower, array& upper, const array& in)
- void lu(array& lower, array& upper, array& pivot, const array& in)
- array qr(const array& in)
- void qr(array& q, array& r, const array& in)
- void qr(array& q, array& r, array& tau, const array& in)
- array cholesky(const array& X, bool is_upper=true)
- array cholesky(unsigned& info, const array& X, bool is_upper=true)
- array hessenberg(const array& in)
- void hessenberg(array& h, array& q, const array& in)
- array eigen(const array& in, bool is_diag=false)
- void eigen(array& values, array& vectors, const array& in)
- array svd(const array& in, bool is_diag=false)
- void svd(array& s, array& u, array& v, const array& in)

Enum af_matprop_t

- AF_NONE - Default
- AF_CTRANS - Conjugate Transpose
- AF_TRANS - Transpose

Enum af_solve_t

- AF_SOLVE_NONE - Default
- AF_SOLVE_POSDEF - A is positive definite
- AF_SOLVE_NONPOSDEF - A is not positive definite
- AF_SOLVE_GAUSSIAN - Use Gaussian elimination (fast, cannot be combined with other options)
- AF_SOLVE_PSEUDO - Use pseudo inverse (fast, cannot be combined with other options)
- AF_SOLVE_CTRANS - Solve A.H() (conjugate transpose)
- AF_SOLVE_TRANS - Solve A.T() (non-conjugate transpose)
- AF_SOLVE_UPPERTRI - Solve upper triangular system
- AF_SOLVE_LOWERTRI - Solve lower triangular system
- AF_SOLVE_TRIDIAG
- AF_SOLVE_BLKDIAG

Sparse Matrix Support

- array sparse (const int *row_idx, const int *col_idx, const float *values, int nrows, int ncols, int nonzero, af_source_t src=afHost)
- array sparse (const int *row_idx, const int *col_idx, const double *values, int nrows, int ncols, int nonzero, af_source_t src=afHost)
- array sparse (const int *row_idx, const int *col_idx, const cfloat *values, int nrows, int ncols, int nonzero, af_source_t src=afHost)
- array sparse (const int *row_idx, const int *col_idx, const cdouble *values, int nrows, int ncols, int nonzero, af_source_t src=afHost)
- array dense(array sparse)
- afError where(array& I, array& J, array& Val, array sp, af_sparse_t format = af_coo)

Enum af_sparse_t

- AF_CSR
- AF_CSC
- AF_COO