```
import Foundation

/// The data type of scalars in the multi-array.
@available macOS 10.13
public enum MLMultiArrayDataType   Int  @unchecked Sendable

    case double   65600

    @available macOS 11.0
    public static var float64  MLMultiArrayDataType   get

    case float32   65568

    @available macOS 12.0
    case float16   65552

    @available macOS 11.0
    public static var float  MLMultiArrayDataType   get

    case int32   131104


/// Use `MLMultiArray` to store a multi-dimensional value.
///
/// Unlike `MLShapedArray` or `MLTensor`, `MLMultiArray` can be used in Obj-C code. Unlike `MLTensor`, `MLMultiArray` is
/// always backed by a concrete storage.
///
/// The object has properties to define the interpretation of the storage.
///
/// `.dataType` defines the interpretation of raw bytes into a numeric scalar value. For example,
/// `MLMultiArrayDataTypeFloat32` means the backing storage uses IEEE 754 Float32 encoding.
///
/// `.shape` defines the multi-dimensional space. For example, 30 x 20 image with three color components (Red, Green,
/// Blue) could be defined using the shape `[3, 20, 30]`.
///
/// `.strides` defines the offset addressing of the scalar for a given coordinates. For example, the image above might
/// use `[640, 32, 1]` as the `strides`. Then, the scalar at (1, 10, 15) is stored at `640 * 1 + 32 * 10 + 1 * 15`, or
/// 975th scalar in the storage. In general, the scalar offset for coordinates `index` and strides `strides` is:
///
/// ```
/// scalarOffset = sum_d index[d]*strides[d]
/// ```
///
```

```
/// The backing storage can be a heap allocated buffer or CVPixelBuffer. Though CVPixelBuffer backing supports limited
/// data types, `MLModel` could share the storage with backend hardware such as Apple Neural Engine without copy.
@available macOS 10.13
open class MLMultiArray  NSObject  NSSecureCoding

    /// Unsafe pointer to underlying buffer holding the data
    @available  10.13  100000  "Use getBytesWithHandler or getMutableBytesWithHandler instead. For Swift, use withUnsafeBytes or withUnsafeMutableBytes."
    open var dataPointer  UnsafeMutableRawPointer  get

    /// Scalar's data type.
    open var dataType  MLMultiArrayDataType  get

    /// Shape of the multi-dimensional space that this instance represents.
    open var shape  NSNumber  get

    /// Strides.
    ///
    /// It defines the offset of the scalar of a given coordinate index in the storage, which is:
    /// ```
    /// scalarOffset = sum_d index[d]*strides[d]
    /// ```
    open var strides  NSNumber  get

    /// Count of total number of addressable scalars.
    ///
    /// The value is same as `product_d shape[d]`.
    open var count  Int  get

    /// Returns the backing pixel buffer if exists, otherwise nil.
    @available macOS 12.0
    open var pixelBuffer  CVPixelBuffer  get


@available macOS 10.15  iOS 13.0  watchOS 6.0  tvOS 13.0
extension MLMultiArray

    /// An MLMultiArray constructed with the FixedWidthInteger elements of the collection
    /// converted to Int32.
    /// ~~~~
    /// let v: [Int32] = [3, 2, 1]
    /// let m = try MLMultiArray(v)
    /// print(m)
    /// Int32 3 vector
```

```
    /// [3,2,1]
    /// ~~~~
    ///
    /// This initializer will trap if called with data containing FixedWidthInteger
elements
    /// that cannot be safely converted to Int32, but it is safe to use with
    /// wider types so long as the actual data is within range.
    /// ~~~~
    /// let a = try MLMultiArray([Int.max]) // trap!
    /// let b = try MLMultiArray([Int(Int32.max), Int(Int32.min)]) // This is fine.
    /// ~~~~
    public convenience init C  _       C  throws where C
Collection  C Element   FixedWidthInteger

    /// An MLMultiArray constructed with the elements of a collection
    /// whose elements are type Float.
    /// ~~~~
    /// let v:[Float] = [3.14159, 2.71828, 1.61803]
    /// let m = try MLMultiArray(v)
    /// print(m)
    /// Float32 3 vector
    /// [3.14159,2.71828,1.61803]
    /// ~~~~
    public convenience init C  _       C  throws where C
Collection  C Element    Float

    /// An MLMultiArray constructed with the elements of a collection
    /// whose elements are type Double.
    /// ~~~~
    /// let v:[Double] = [3.14159, 2.71828, 1.61803]
    /// let m = try MLMultiArray(v)
    /// print(m)
    /// Double 3 vector
    /// [3.14159,2.71828,1.61803]
    /// ~~~~
    public convenience init C  _       C  throws where C
Collection  C Element    Double

    /// Creates the object with specified strides.
    ///
    /// The contents of the object are left uninitialized; the client must initialize it.
    ///
    /// - Parameters:
    ///   - shape: The shape
    ///   - dataType: The data type
    ///   - strides: The strides.
    @available macOS 15.0  iOS 18.0  watchOS 11.0  tvOS 18.0
visionOS 2.0
    public convenience init        Int
```

```
MLMultiArrayDataType            Int

    /// An MLMultiArray constructed from MLShapedArray or its slice.
    /// ~~~~
    /// let s = MLShapedArray<Int32>(scalars: 0 ..< 6, shape: [3, 2])
    /// let m = try MLMultiArray(s)
    /// print(m)
    /// Int32 3 x 2 matrix
    /// [[0, 1]
    ///  [2, 3]
    ///  [4, 5]]
    /// ~~~~
    @available macOS 12.0  iOS 15.0  watchOS 8.0  tvOS 15.0

    public convenience init ShapedArray  _
ShapedArray  where ShapedArray  MLShapedArrayProtocol

    /// Calls a given closure with a raw pointer to the multi array's storage.
    ///
    /// ```
    /// let A = try MLMultiArray(shape:[3, 2],
dataType: .int32)
    /// A[[1, 2]] = 42
    /// A.withUnsafeBytes { bytes in
    ///     let scalarBuffer = bytes.bindMemory(to:
Int32.self)
    ///     let strideY = A.strides[0].intValue
    ///     // Print 42
    ///     print("Scalar at (1, 2): \(scalarBuffer[1 *
strideY + 2])")
    /// }
    /// ```
    ///
    /// - Parameter body: A closure with an
`UnsafeRawBufferPointer` parameter that points to the
    ///     storage for the multi array. If body has a return value, that value is also
used as the
    ///     return value for the `withUnsafeBytes(_:)` method. The pointer
argument is
    ///     valid only for the duration of the method's execution.
    /// - Parameter ptr: The pointer to the memory buffer of the multi array.
    @available macOS 12.3  iOS 15.4  watchOS 8.5  tvOS 15.4

    public func withUnsafeBytes R  _        _
UnsafeRawBufferPointer  throws    R  rethrows    R

    /// Calls a given closure with a raw pointer to the multi array's mutable
storage.
    ///
    /// Use strides passed to the closure to access the buffer. This is a mutating
```

```
function, which
    /// may change the underlying buffer layoout.
    ///
    /// ```
    /// let A = try MLMultiArray(shape:[3, 2],
dataType: .int32)
    /// A.withUnsafeMutableBytes { bytes, strides in
    ///     let scalarBuffer = bytes.bindMemory(to:
Int32.self)
    ///     let strideY = strides[0]
    ///     scalarBuffer[1 * strideY + 2] = 42  // Set 42 at
A[1, 2]
    /// }
    /// ```
    ///
    /// - Parameter body: A closure with an
`UnsafeRawBufferPointer` parameter that points to the
    ///     storage for the multi array. If body has a return value, that value is also
used as the
    ///     return value for the `withUnsafeMutableBytes(_:)` method. The
pointer argument is
    ///     valid only for the duration of the method's execution.
    /// - Parameter ptr: The pointer to the memory buffer of the multi array.
    /// - Parameter strides: The strides of the buffer in number of scalars,
not bytes.
    @available macOS 12.3  iOS 15.4  watchOS 8.5  tvOS 15.4

    public func withUnsafeMutableBytes R  _          _
UnsafeMutableRawBufferPointer  _          Int   throws    R
rethrows    R

    /// Calls a given closure with a raw pointer to the multi array's storage.
    ///
    /// ```
    /// let A = try MLMultiArray(shape:[3, 2],
dataType: .int32)
    /// A[[1, 2]] = 42
    /// A.withUnsafeBufferPointer(ofType: Int32.self)
{ scalarBuffer in
    ///     let strideY = A.strides[0].intValue
    ///     // Print 42
    ///     print("Scalar at (1, 2): \(scalarBuffer[1 *
strideY + 2])")
    /// }
    /// ```
    ///
    /// - Parameter body: A closure with an `UnsafeBufferPointer`
parameter that points to the
    ///     storage for the multi array. If body has a return value, that value is also
used as the
```

```
    ///     return value for the `withUnsafeBufferPointer(_:)` method.
The pointer argument is
    ///     valid only for the duration of the method's execution.
    /// - Parameter ptr: The pointer to the memory buffer of the multi array.
    @available macOS 12.3  iOS 15.4  watchOS 8.5  tvOS 15.4

    public func withUnsafeBufferPointer S  R
S       _           _       UnsafeBufferPointer S   throws    R
rethrows    R where S   MLShapedArrayScalar

    /// Calls a given closure with a buffer pointer to the multi array's mutable
storage.
    ///
    /// Use strides passed to the closure to access the buffer. This is a mutating
function, which
    /// may change the underlying buffer layoout.
    ///
    /// ```
    /// let A = try MLMultiArray(shape:[3, 2],
dataType: .int32)
    /// A.withUnsafeMutableBufferPointer(ofType: Int32.self) {
scalarBuffer, strides in
    ///     let strideY = strides[0]
    ///     scalarBuffer[1 * strideY + 2] = 42  // Set 42 at
A[1, 2]
    /// }
    /// ```
    ///
    /// - Parameter body: A closure with an
`UnsafeRawBufferPointer` parameter that points to the
    ///     storage for the multi array. If body has a return value, that value is also
used as the
    ///     return value for the `withUnsafeMutableBufferPointer(_:)`
method. The pointer argument is
    ///     valid only for the duration of the method's execution.
    /// - Parameter ptr: The pointer to the memory buffer of the multi array.
    /// - Parameter strides: The strides of the buffer in number of scalars,
not bytes.
    @available macOS 12.3  iOS 15.4  watchOS 8.5  tvOS 15.4

    public func withUnsafeMutableBufferPointer S  R
      S       _           _       UnsafeMutableBufferPointer S   _
        Int   throws    R  rethrows    R where S
MLShapedArrayScalar


extension MLMultiArray

    /// Creates the object.
    ///
```

```
    /// The contents of the object are left uninitialized; the client must initialize it.
    ///
    /// The scalars will use the first-major contiguous layout.
    ///
    /// - Parameters:
    ///   - shape: The shape
    ///   - dataType: The data type
    ///   - error: Filled with error information on error.
    public init(shape: [NSNumber], dataType: MLMultiArrayDataType) throws

    /// Creates the object with existing data without copy.
    ///
    /// Use this initializer to reference the existing buffer as the storage without copy.
    ///
    /// ```objc
    /// int32_t *buffer = malloc(sizeof(int32_t) * 2 * 3 * 4);
    /// MLMultiArray *multiArray = [[MLMultiArray alloc] initWithDataPointer:buffer
    ///                                                                shape:@[@2, @3, @4]
    ///                                                             dataType:MLMultiArrayDataTypeInt32
    ///                                                              strides:@[@12, @4, @1]
    ///                                                          deallocator:^(void *bytes) { free(bytes); }
    ///                                                                error:NULL];
    /// ```
    ///
    /// - Parameters:
    ///   - dataPointer: The pointer to the buffer.
    ///   - shape: The shape
    ///   - dataType: The data type
    ///   - strides: The strides.
    ///   - deallocator: Block to be called on the deallocation of the instance.
    ///   - error: Filled with error information on error.
    public init(dataPointer: UnsafeMutableRawPointer, shape: [NSNumber], dataType: MLMultiArrayDataType, strides: [NSNumber], deallocator: ((UnsafeMutableRawPointer) -> Void)?) throws

    /// Create by wrapping a pixel buffer.
    ///
    /// Use this initializer to create an IOSurface backed MLMultiArray, which can reduce the inference latency by avoiding the buffer copy.
    ///
```

```
    /// The instance will own the pixel buffer and release it on the deallocation.
    ///
    /// The pixel buffer's pixel format type must be OneComponent16Half. As such, MLMultiArray's data type will be MLMultiArrayDataTypeFloat16.
    ///
    /// ```objc
    /// CVPixelBufferRef pixelBuffer = NULL;
    /// NSDictionary* pixelBufferAttributes = @{
    ///     (id)kCVPixelBufferIOSurfacePropertiesKey: @{}
    /// };
    ///
    /// // Since shape == [2, 3, 4], width is 4 (= shape[2]) and height is 6 (= shape[0] * shape[1]).
    /// CVPixelBufferCreate(kCFAllocatorDefault, 4, 6, kCVPixelFormatType_OneComponent16Half, (__bridge CFDictionaryRef)pixelBufferAttributes, &pixelBuffer);
    /// MLMultiArray *multiArray = [[MLMultiArray alloc] initWithPixelBuffer:pixelBuffer shape:@[@2, @3, @4]];
    /// ```
    ///
    /// - Parameters:
    ///   - pixelBuffer: The pixel buffer to be owned by the instance.
    ///   - shape: The shape of the MLMultiArray. The last dimension of `shape` must match the pixel buffer's width. The product of the rest of the dimensions must match the height.
    @available macOS 12.0
    public init                CVPixelBuffer          NSNumber


extension MLMultiArray

    /// Concatenate MLMultiArrays to form a new MLMultiArray.
    ///
    /// All the source MLMultiArrays must have a same shape except the specified axis. The resultant
    /// MLMultiArray has the same shape as inputs except this axis, which dimension will be the sum of
    /// all the input dimensions of the axis.
    ///
    /// For example,
    ///
    /// ```swift
    /// // Swift
    /// let A = try MLMultiArray(shape: [2, 3], dataType: .int32)
    /// let B = try MLMultiArray(shape: [2, 2], dataType: .int32)
    /// let C = MLMultiArray(concatenating: [A, B], axis: 1, dataType: .int32)
    /// assert(C.shape == [2, 5])
```

```
    /// ```
    ///
    /// ```objc
    /// // Obj-C
    /// MLMultiArray *A = [[MLMultiArray alloc]
initWithShape:@[@2, @3] dataType:MLMultiArrayDataTypeInt32
error:NULL];
    /// MLMultiArray *B = [[MLMultiArray alloc]
initWithShape:@[@2, @2] dataType:MLMultiArrayDataTypeInt32
error:NULL];
    /// MLMultiArray *C = [MLMultiArray
multiArrayByConcatenatingMultiArrays:@[A, B] alongAxis:1
dataType:MLMultiArrayDataTypeInt32];
    /// assert(C.shape == @[@2, @5])
    /// ```
    ///
    /// Numeric data will be up or down casted as needed.
    ///
    /// The method raises NSInvalidArgumentException if the shapes of input
multi arrays are not
    /// compatible for concatenation.
    ///
    /// - Parameters:
    ///   - multiArrays: Array of MLMultiArray instances to be
concatenated.
    ///   - axis: Axis index with which the concatenation will performed. The
value is wrapped by the dimension of the axis. For example, -1 is the last axis.
    ///   - dataType: The data type of the resultant MLMultiArray.
    @available macOS 11.0
    public convenience init
 MLMultiArray         Int            MLMultiArrayDataType


extension MLMultiArray

    /// Get a value by its linear index (assumes C-style index ordering)
    open subscript      Int     NSNumber

    /// Get a value by its multidimensional index (NSArray<NSNumber *>)
    open subscript       NSNumber      NSNumber


extension MLMultiArray

    /// Transfer the contents to the destination multi-array.
    ///
    /// Numeric data will be up or down casted as needed. It can transfer to a
multi-array with different layout (strides).
    ///
    /// ```swift
```

```
    /// let sourceMultiArray: MLMultiArray = ... // shape is
[2, 3] and data type is Float64
    ///
    /// let newStrides = [4, 1]
    /// let destinationMultiArray = MLMultiArray(shape: [2,
3],
    ///
dataType: .float32,
    ///                                          strides:
newStrides)
    /// sourceMultiArray.transfer(to: destinationMultiArray)
    /// ```
    ///
    /// ```objc
    /// NSArray<NSNumber *> *shape = @[@2, @3];
    /// NSArray<NSNumber *> *sourceStrides = @[@3, @1];
    /// NSArray<NSNumber *> *destinationStrides = @[@4, @1];
    /// MLMultiArray *source = [[MLMultiArray alloc]
initWithShape:shape
    ///
dataType:MLMultiArrayDataTypeDouble
    ///
strides:sourceStrides];
    /// // Initialize source...
    ///
    /// MLMultiArray *destination = [[MLMultiArray alloc]
initWithShape:shape
    ///
dataType:MLMultiArrayDataTypeFloat32
    ///
strides:destinationStrides];
    /// [source transferToMultiArray:destination];
    /// ```
    ///
    /// - Parameters:
    ///   - destinationMultiArray: The transfer destination.
    @available macOS 15.0
    open func transfer                                MLMultiArray
```