MexIFace

Generated by Doxygen 1.8.13

Contents

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 Class Documentation

3.1 Handle< T > Class Template Reference

A class to represent and manipulate handles to C++ classes that can be wrapped as Matlab arrays, allowing C++ objects to persist between Mex calls.

```
#include </nfs/olah/home/mjo/github/MexIFace/src/Handle.h>
```

Public Types

- typedef uint64_t HandlePtrT

  The C++ datatype of corresponding to the array type a Handle pointer will be stored in for the Matlab side of things.

Public Member Functions

- Handle (T ∗obj)

  Make a new handle object to hold a pointer to given object of class T.
- ∼Handle ()

  Delete the object which was assumed to have been created with new.
- bool is_valid () const

  Check that this is a valid handle to a valid C++ object.
- T ∗ object () const

  Retrieve a pointer to the object stored by this handle.

Static Public Member Functions

- static mxArray ∗ makeHandle (T ∗obj)

  Given a pointer to a C++ object, make a new Handle object and save that as a uint64_t in a Matlab mxArray object.
- static Handle< T > ∗ getHandle (const mxArray ∗arr)

  Given a Matlab mxArray object pointer to data that represents a handle, return a Handle object pointer.
- static T ∗ getObject (const mxArray ∗in)

  Given a matlab mxArray object pointer to data that represents a handle, retrieve the object pointer for the C++ object.
- static void destroyObject (const mxArray ∗in)

  Given a matlab mxArray object pointer to data that represents a handle, delete the handle which also implies deleting object itself.

3.1.1 Detailed Description

template<class T>
class Handle< T >

A class to represent and manipulate handles to C++ classes that can be wrapped as Matlab arrays, allowing C++ objects to persist between Mex calls.

This allows Matlab to hold a handle to a C++ object allocated during one Mex call, but used during subsequent calls.

Definition at line 22 of file Handle.h.

3.1.2 Member Typedef Documentation

3.1.2.1 HandlePtrT

```
template<class T>
typedef uint64_t Handle< T >::HandlePtrT
```

The C++ datatype of corresponding to the array type a Handle pointer will be stored in for the Matlab side of things.

Definition at line 29 of file Handle.h.

3.1.3 Constructor & Destructor Documentation

3.1.3.1 Handle()

```
template<class T >
Handle< T >::Handle (
            T * obj )
```

Make a new handle object to hold a pointer to given object of class T.

Parameters

obj	The object to wrap in a handle and make persistent

Note: we assume ownership of obj and will free it with a call to delete when the handle itself is deleted. This assumes obj was created with a call to new.

Definition at line 61 of file Handle.h.

3.1.3.2 ∼Handle()

```
template<class T >
Handle< T >::∼Handle ( )
```

Delete the object which was assumed to have been created with new.

Definition at line 73 of file Handle.h.

3.1.4 Member Function Documentation

3.1.4.1 destroyObject()

```
template<class T >
void Handle< T >::destroyObject (
             const mxArray * arr )  [static]
```

Given a matlab mxArray object pointer to data that represents a handle, delete the handle which also implies deleting object itself.

Parameters

arr	The Matlab mxArray that contains the numerically encoded pointer to the handle we wish to destroy

The Handle object as well as the object that wrapped it are assumed to have been created with a call to new, and we are assuming that the Handle object now "owns" the memory of the wrapped object and thus is responsible for freeing it.

This also decrements the mexLock count, as we have freed one of the persistent object we previously created.

Definition at line 161 of file Handle.h.

3.1.4.2 getHandle()

```
template<class T >
Handle< T > * Handle< T >::getHandle (
             const mxArray * arr )  [static]
```

Given a Matlab mxArray object pointer to data that represents a handle, return a Handle object pointer.

Parameters

arr	A Matlab mxArray where the handle is stored as a uint64_t scalar.

Returns

A pointer to the Handle object

Definition at line 128 of file Handle.h.

3.1.4.3 getObject()

```
template<class T >
T * Handle< T >::getObject (
             const mxArray * arr )  [inline], [static]
```

Given a matlab mxArray object pointer to data that represents a handle, retrieve the object pointer for the C++ object.

Parameters

arr	A Matlab mxArray where the handle is stored as a uint64_t scalar.

Returns

A pointer to the actual object that was wrapped in the Handle object that was itself stored in the array numerically

Definition at line 145 of file Handle.h.

3.1.4.4 is_valid()

```
template<class T >
bool Handle< T >::is_valid ( ) const
```

Check that this is a valid handle to a valid C++ object.

Returns

True if valid

Definition at line 85 of file Handle.h.

3.1.4.5 makeHandle()

```
template<class T >
mxArray * Handle< T >::makeHandle (
              T * obj )  [static]
```

Given a pointer to a C++ object, make a new Handle object and save that as a uint64_t in a Matlab mxArray object.

Parameters

obj	The object to wrap.

Returns

A mxArray that contains the handle as a numeric scalar uint64_t

Definition at line 113 of file Handle.h.

3.1.4.6 object()

```
template<class T >
T * Handle< T >::object ( ) const [inline]
```

Retrieve a pointer to the object stored by this handle.

Returns

The pointer to the object

Definition at line 99 of file Handle.h.

The documentation for this class was generated from the following file:

- Handle.h

3.2 Hypercube< ElemT > Class Template Reference

A class to create a 4D armadillo array that can use externally allocated memory.

```
#include </nfs/olah/home/mjo/github/MexIFace/src/hypercube.h>
```

Public Member Functions

- Hypercube (int sX, int sY, int sZ, int sN)

  Create an empty hypercube of specified size.
- Hypercube (void ∗mem, int sX, int sY, int sZ, int sN)

  Create a hypercube of specified size using externally allocated 4D column-major array data.
- void zeros ()

  Zero out all cubes in this hypercube.
- const CubeT & slice (int i) const

  Get a subcube with index i.
- CubeT & slice (int i)

  Get a subcube with index i.
- ElemT & operator() (int iX, int iY, int iZ, int iN) const

  Access element at coords.
- int subcube_size () const

  Get the number of elements in each subcube.
- int size () const

  Get the number of elements in this hypercube.

Public Attributes

- const int sX
- const int sY
- const int sZ
- const int sN
- const unsigned n_slices

    This member variable matches the n_slices member of arma::Cube's and allows us to have a hypercube stand in for a cube in templated code that can work on 2D or 3D sub-slices.

3.2.1 Detailed Description

template<class ElemT>
class Hypercube< ElemT >

A class to create a 4D armadillo array that can use externally allocated memory.

This class provides a way to manipulate externally allocated 4D column-major arrays as a armadillo-like Array object. Really we just store a vector of arma::Cube's each of which has been initialized with the correct 3D chunk from the 4D external array. This allows us to work directly with Matlab allocated 4D arrays in C++. Unfortunately most of the armadillo functions won't work with this hypercube, but the slice method allows easy access to the actual armadillo Cubes that make up the Hypercube.

Definition at line 27 of file hypercube.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Hypercube() [1/2]

```
template<class ElemT>
Hypercube< ElemT >::Hypercube (
              int sX,
              int sY,
              int sZ,
              int sN )  [inline]
```

Create an empty hypercube of specified size.

Parameters

sX	The x coordinate (1st dim).
sY	The y coordinate (2nd dim).
sZ	The z coordinate (3rd dim).
sN	The n (hyperslice) coordinate (4th dim).

Definition at line 39 of file hypercube.h.

3.2.2.2 Hypercube() [2/2]

```
template<class ElemT>
Hypercube< ElemT >::Hypercube (
             void * mem,
             int sX,
             int sY,
             int sZ,
             int sN )  [inline]
```

Create a hypercube of specified size using externally allocated 4D column-major array data.

Parameters

mem	Pointer to external memory of a 4D column-major array, that this hypercube will give access to
sX	The x coordinate (1st dim).
sY	The y coordinate (2nd dim).
sZ	The z coordinate (3rd dim).
sN	The n (hyperslice) coordinate (4th dim).

Definition at line 56 of file hypercube.h.

3.2.3 Member Function Documentation

3.2.3.1 operator()()

```
template<class ElemT>
ElemT& Hypercube< ElemT >::operator() (
             int iX,
             int iY,
             int iZ,
             int iN ) const  [inline]
```

Access element at coords.

Parameters

iX	The x coordinate (1st dim).
iY	The y coordinate (2nd dim).
iZ	The z coordinate (3rd dim).
iN	The n (hyperslice) coordinate (4th dim).

Returns

A reference to the element

Definition at line 102 of file hypercube.h.

3.2.3.2 size()

```
template<class ElemT>
int Hypercube< ElemT >::size ( ) const  [inline]
```

Get the number of elements in this hypercube.

Definition at line 119 of file hypercube.h.

3.2.3.3 slice() [1/2]

```
template<class ElemT>
const CubeT& Hypercube< ElemT >::slice (
              int i ) const  [inline]
```

Get a subcube with index i.

Parameters

i	the sub-cube index, in the 4-th dim.

Returns

A constant reference to the subcube

Definition at line 77 of file hypercube.h.

3.2.3.4 slice() [2/2]

```
template<class ElemT>
CubeT& Hypercube< ElemT >::slice (
              int i )  [inline]
```

Get a subcube with index i.

Parameters

i	the sub-cube index, in the 4-th dim.

Returns

A reference to the subcube

Definition at line 88 of file hypercube.h.

3.2.3.5 subcube_size()

```
template<class ElemT>
int Hypercube< ElemT >::subcube_size ( ) const  [inline]
```

Get the number of elements in each subcube.

Definition at line 111 of file hypercube.h.

3.2.3.6 zeros()

```
template<class ElemT>
void Hypercube< ElemT >::zeros ( )  [inline]
```

Zero out all cubes in this hypercube.

Definition at line 70 of file hypercube.h.

3.2.4 Member Data Documentation

3.2.4.1 n_slices

```
template<class ElemT>
const unsigned Hypercube< ElemT >::n_slices
```

This member variable matches the n_slices member of arma::Cube's and allows us to have a hypercube stand in for a cube in templated code that can work on 2D or 3D sub-slices.

Definition at line 132 of file hypercube.h.

3.2.4.2 sN

```
template<class ElemT>
const int Hypercube< ElemT >::sN
```

Definition at line 126 of file hypercube.h.

3.2.4.3 sX

```
template<class ElemT>
const int Hypercube< ElemT >::sX
```

Definition at line 126 of file hypercube.h.

3.2.4.4 sY

```
template<class ElemT>
const int Hypercube< ElemT >::sY
```

Definition at line 126 of file hypercube.h.

3.2.4.5 sZ

```
template<class ElemT>
const int Hypercube< ElemT >::sZ
```

Definition at line 126 of file hypercube.h.

The documentation for this class was generated from the following file:

- hypercube.h

3.3 MexIFace Class Reference

Acts as a base class for implementing a C++ class <–> Matlab class interface.

```
#include </nfs/olah/home/mjo/github/MexIFace/src/MexIFace.h>
```

Public Types

- using IdxT = arma::uword
- using BoolT = uint16_t
- typedef std::map< std::string, double > StatsT
- typedef std::map< std::string, arma::Col< double > > VecStatsT
- template<class T >
  using VectorVector = std::vector< std::vector< T > >
- template<class T >
  using VectorList = std::vector< std::list< T > >
- template<class T >
  using VecField = arma::field< arma::Col< T > >
- template<class T >
  using MatField = arma::field< arma::Mat< T > >
- template<class T >
  using CubeField = arma::field< arma::Cube< T > >
- template<class T >
  using VecVector = std::vector< arma::Col< T > >
- template<class T >
  using MatVector = std::vector< arma::Mat< T > >
- template<class T >
  using CubeVector = std::vector< arma::Cube< T > >

Public Member Functions

- MexIFace (std::string name)
- void mexFunction (unsigned _nlhs, mxArray ∗_lhs[ ], unsigned _nrhs, const mxArray ∗_rhs[ ])

  The mexFunction that will be exposed as the entry point for the .mex file.

Public Attributes

- std::string mex_name

Protected Types

- typedef std::map< std::string, boost::function< void()> > MethodMap

Protected Member Functions

- virtual void objConstruct ()=0

  Called when the mexFunction gets the @new command, passing on the remaining input arguments.
- virtual void objDestroy ()=0

  Called when the mexFunction gets the @delete command, passing on the remaining input arguments.
- virtual void getObjectFromHandle (const mxArray ∗mxhandle)=0

  This is a helper method which saves a pointer to the wrapped class's object in an internal member variable called obj.
- void callMethod (std::string name)

  Calls a named member function on the instance of the wrapped class.
- void callStaticMethod (std::string name)

  Calls a named static member function of the wrapped class.
- void checkMinNumArgs (int min_nlhs, int min_nrhs) const

  Checks that the Iface mex function was called with a minimum number of input and output arguments.
- void checkMaxNumArgs (int max_nlhs, int max_nrhs) const

  Checks that the Iface mex function was called with a maximum number of input and output arguments.
- void checkNumArgs (int nlhs, int nrhs) const

  Checks that the Iface mex function was called with exactly the expected number of input and output arguments.
- void checkSameLastDim (const mxArray ∗m1, const mxArray ∗m2) const

  Checks that two matlab mxArray objects have the same sized last dimension.
- void checkDim (const mxArray ∗m, int rows, int cols) const

  Checks that a matlab mxArray object has the correct 2D dimensions.
- bool getBool (const mxArray ∗mxdata=nullptr)
- int getInt (const mxArray ∗mxdata=nullptr)

  Reads a mxArray as a scalar C++ int32_t type.
- unsigned getUnsigned (const mxArray ∗mxdata=nullptr)

  Reads a mxArray as a scalar C++ uint32_t type.
- float getFloat (const mxArray ∗mxdata=nullptr)

  Reads a mxArray as a scalar C++ float type.
- double getDouble (const mxArray ∗mxdata=nullptr)

  Reads a mxArray as a scalar C++ double type.

- template<class ElemT >
  ElemT getScalar (const mxArray ∗mxdata=nullptr)
- std::string getString (const mxArray ∗mxdata=nullptr)

  Reads a mxArray as a string.
- StatsT getDoubleStruct (const mxArray ∗mxdata=nullptr)

  Process a matlab structure returning a StatsT mapping from keys to double.
- VecStatsT getDoubleVecStruct (const mxArray ∗mxdata=nullptr)

  Process a matlab structure returning a VecStatsT mapping from keys to column vectors.
- arma::Col< uint16_t > getU16Vec (const mxArray ∗mxdata=nullptr)
- arma::Col< uint32_t > getUVec (const mxArray ∗mxdata=nullptr)
- arma::Col< int32_t > getIVec (const mxArray ∗mxdata=nullptr)
- arma::Col< float > getFVec (const mxArray ∗mxdata=nullptr)
- arma::Col< double > getDVec (const mxArray ∗mxdata=nullptr)
- template<class ElemT >
  arma::Col< ElemT > getVec (const mxArray ∗mxdata=nullptr)

  Create an armadillo Column vector to directly work with the Matlab data for a 1D array of arbitrary element type.
- arma::Mat< uint16_t > getU16Mat (const mxArray ∗mxdata=nullptr)
- arma::Mat< uint32_t > getUMat (const mxArray ∗mxdata=nullptr)
- arma::Mat< int32_t > getIMat (const mxArray ∗mxdata=nullptr)
- arma::Mat< float > getFMat (const mxArray ∗mxdata=nullptr)
- arma::Mat< double > getDMat (const mxArray ∗mxdata=nullptr)
- template<class ElemT >
  arma::Mat< ElemT > getMat (const mxArray ∗mxdata=nullptr)

  Create an armadillo Mat object to directly work with the Matlab data for a 2D array of arbitrary element type.
- arma::Cube< uint16_t > getU16Stack (const mxArray ∗mxdata=nullptr)
- arma::Cube< uint32_t > getUStack (const mxArray ∗mxdata=nullptr)
- arma::Cube< int32_t > getIStack (const mxArray ∗mxdata=nullptr)
- arma::Cube< float > getFStack (const mxArray ∗mxdata=nullptr)
- arma::Cube< double > getDStack (const mxArray ∗mxdata=nullptr)
- template<class ElemT >
  arma::Cube< ElemT > getStack (const mxArray ∗mxdata=nullptr)

  Create an armadillo Cube object to directly work with the Matlab data for a 3D array of arbitrary element type.
- Hypercube< uint16_t > getU16HyperStack (const mxArray ∗mxdata=nullptr)
- Hypercube< uint32_t > getUHyperStack (const mxArray ∗mxdata=nullptr)
- Hypercube< int32_t > getIHyperStack (const mxArray ∗mxdata=nullptr)
- Hypercube< float > getFHyperStack (const mxArray ∗mxdata=nullptr)
- Hypercube< double > getDHyperStack (const mxArray ∗mxdata=nullptr)
- template<class ElemT >
  Hypercube< ElemT > getHyperStack (const mxArray ∗mxdata=nullptr)

  Create an Hypercube object to directly work with the Matlab data for a 4D array of arbitrary element type.
- template<class ElemT >
  VecField< ElemT > getVecField (const mxArray ∗mxdata=nullptr)
- template<class ElemT >
  MatField< ElemT > getMatField (const mxArray ∗mxdata=nullptr)
- template<class ElemT >
  CubeField< ElemT > getCubeField (const mxArray ∗mxdata=nullptr)
- template<class ElemT >
  VecVector< ElemT > getVecVector (const mxArray ∗mxdata=nullptr)
- template<class ElemT >
  MatVector< ElemT > getMatVector (const mxArray ∗mxdata=nullptr)

- template<class ElemT >
  CubeVector< ElemT > getCubeVector (const mxArray ∗mxdata=nullptr)
- arma::Col< uint32_t > makeUVec (unsigned rows)
- arma::Col< int32_t > makeIVec (unsigned rows)
- arma::Col< float > makeFVec (unsigned rows)
- arma::Col< double > makeDVec (unsigned rows)
- template<class ElemT >
  arma::Col< ElemT > makeVec (unsigned nelem)
- arma::Mat< uint32_t > makeUMat (unsigned rows, unsigned cols)
- arma::Mat< int32_t > makeIMat (unsigned rows, unsigned cols)
- arma::Mat< float > makeFMat (unsigned rows, unsigned cols)
- arma::Mat< double > makeDMat (unsigned rows, unsigned cols)
- template<class ElemT >
  arma::Mat< ElemT > makeMat (unsigned rows, unsigned cols)
- arma::Cube< uint32_t > makeUStack (unsigned rows, unsigned cols, unsigned slices)
- arma::Cube< int32_t > makeIStack (unsigned rows, unsigned cols, unsigned slices)
- arma::Cube< float > makeFStack (unsigned rows, unsigned cols, unsigned slices)
- arma::Cube< double > makeDStack (unsigned rows, unsigned cols, unsigned slices)
- template<class ElemT >
  arma::Cube< ElemT > makeStack (unsigned rows, unsigned cols, unsigned slices)
- Hypercube< uint16_t > makeU16HyperStack (unsigned rows, unsigned cols, unsigned slices, unsigned hyperslices)
- Hypercube< uint32_t > makeUHyperStack (unsigned rows, unsigned cols, unsigned slices, unsigned hyperslices)
- Hypercube< int32_t > makeIHyperStack (unsigned rows, unsigned cols, unsigned slices, unsigned hyperslices)
- Hypercube< float > makeFHyperStack (unsigned rows, unsigned cols, unsigned slices, unsigned hyperslices)
- Hypercube< double > makeDHyperStack (unsigned rows, unsigned cols, unsigned slices, unsigned hyperslices)
- template<class ElemT >
  Hypercube< ElemT > makeHyperStack (unsigned rows, unsigned cols, unsigned slices, unsigned hyperslices)
- void outputMXArray (mxArray ∗m)
- template<class ElemT >
  void outputVec (const arma::Col< ElemT > &arr)
- void outputDouble (double val)
- void outputInt (int32_t val)
- void outputBool (bool val)
- void outputStatsToStruct (const StatsT &stats)

  Outputs a StatsT as a matlab structure appended to the method's return value.
- void outputStatsToDoubleVecStruct (const VecStatsT &stats)

  Outputs a VecStatsT as a matlab structure appended to the method's return value.
- void outputFVec (const arma::Col< float > &arr)
- template<int N>
  void outputDVec (const arma::Col< double >::fixed< N > &arr)
- void outputDVec (const arma::Col< double > &arr)
- template<class ElemT >
  void outputMat (const arma::Mat< ElemT > &arr)
- void outputIMat (const arma::Mat< int32_t > &arr)
- void outputDMat (const arma::Mat< double > &arr)
- template<class ElemT >
  void outputStack (const arma::Cube< ElemT > &arr)
- template<class ElemT >
  void outputSparse (const arma::SpMat< ElemT > &arr)

- template<class ElemT >
  void outputVecCellArray (const VectorVector< ElemT > &list)
- template<class ElemT >
  void outputVecCellArray (const VectorList< ElemT > &list)
- template<class ElemT >
  void outputVecCellArray (const VecVector< ElemT > &field)
- template<class ElemT >
  void outputMatCellArray (const MatVector< ElemT > &field)
- template<class ElemT >
  void outputCubeCellArray (const CubeVector< ElemT > &field)
- template<class ElemT >
  void outputVecCellArray (const VecField< ElemT > &field)
- template<class ElemT >
  void outputMatCellArray (const MatField< ElemT > &field)
- template<class ElemT >
  void outputCubeCellArray (const CubeField< ElemT > &field)
- void error (std::string condition, std::string message) const

  Reports an error condition to Matlab using the mexErrMsgIdAndTxt function.
- void component_error (std::string component, std::string condition, std::string message) const

  Reports an error condition in a specified component to Matlab using the mexErrMsgIdAndTxt function.
- void popRhs ()

  Remove the first right-hand-side (input) argument as it has already been used to find the correct command.
- void setArguments (unsigned _nlhs, mxArray ∗_lhs[ ], unsigned _nrhs, const mxArray ∗_rhs[ ])

  Helper function to set the internal copies of the left-hand-side and right-hand-side parameters as they were passed to the mexFunction.

Protected Attributes

- MethodMap methodmap
- MethodMap staticmethodmap
- unsigned nlhs
- mxArray ∗∗ lhs
- int lhs_idx =0
- unsigned nrhs
- const mxArray ∗∗ rhs
- int rhs_idx =0

3.3.1 Detailed Description

Acts as a base class for implementing a C++ class <–> Matlab class interface.

The MexIFace class provides a generic means of wrapping a C++ class as a Matlab MEX function, that can then be exposed as a Matlab class. This flexibility allows the code to be used in an object-oriented style either from other C++ code or from Matlab.

This type of interface is necessary because a Matlab .mex plug-in can only act as a Matlab function, not a Matlab class. The MexIFace class exposes a mexFunction method which takes in a variable number of arguments and returns a variable number of arguments. The first input argument is always a string that gives the command name. If it the special command "\@new" or "\@delete" a C++ instance is created or destroyed. The @new command returns a unique handle (number) which can be held onto by the Matlab IfaceMixin base class. This C++ object then remains in memory until the @delete command is called on the MexIFace, which then frees the underlying C++ class from memory.

The special command "\@static" allows static C++ methods to be called by the name passed as the second argument, and there is no need to have a existing object to call the method on because it is static.

Otherwise the command is interpreted as a named method which is registered in the methodmap, internal data structure which maps strings to callable member functions of the interface object which take in no arguments and return no arguments. The matlab arguments are passed to these functions through the internal storage of the MexIFace object's rhs and lhs member variables.

A C++ class is wrapped by creating a new Iface class that inherits from MexIFace. At a minimum the Iface class must define the pure virtual functions objConstruct(), objDestroy(), and getObjectFromHandle(). It also must implement the interface for any of the methods and static methods that are required. Each of these methods in the Iface class must process the passed matlab arguments in the rhs member variable and save outputs in the lhs member variable.

In general the Iface mex modules are not intended to be used directly, but rather are paired with a special Matlab class that inherits from the IfaceMixin.m base class.

Design decision: Because of the complexities of inheriting from a templated base class with regard to name lookups in superclasses, we chose to keep this MexIFace class non-templated. For this reason any methods and member variables which specifically mention the type of the wrapped class must be defined in the subclass of MexIFace.

Finally we provide many get∗ and make∗ which allow the lhs and rhs arguments to be interpreted as armadillo arrays on the C++ side. These methods are part of what makes this interface efficient as we don't need to create new storage and copy data, instead we just use the matlab memory directly, and matlab does all the memory management of parameters passed in and out.

Definition at line 70 of file MexIFace.h.

3.3.2 Member Typedef Documentation

3.3.2.1 BoolT

```
using MexIFace::BoolT = uint16_t
```

Definition at line 73 of file MexIFace.h.

3.3.2.2 CubeField

```
template<class T >
using MexIFace::CubeField = arma::field<arma::Cube<T> >
```

Definition at line 80 of file MexIFace.h.

3.3.2.3 CubeVector

```
template<class T >
using MexIFace::CubeVector = std::vector<arma::Cube<T> >
```

Definition at line 83 of file MexIFace.h.

3.3.2.4 IdxT

```
using MexIFace::IdxT = arma::uword
```

Definition at line 72 of file MexIFace.h.

3.3.2.5 MatField

```
template<class T >
using MexIFace::MatField = arma::field<arma::Mat<T> >
```

Definition at line 79 of file MexIFace.h.

3.3.2.6 MatVector

```
template<class T >
using MexIFace::MatVector = std::vector<arma::Mat<T> >
```

Definition at line 82 of file MexIFace.h.

3.3.2.7 MethodMap

```
typedef std::map<std::string, boost::function<void()> > MexIFace::MethodMap  [protected]
```

The type of mapping for mapping names to member functions to call

Definition at line 91 of file MexIFace.h.

3.3.2.8 StatsT

```
typedef std::map<std::string,double> MexIFace::StatsT
```

A convenient form for reporting dictionaries of named FP data to matlab

Definition at line 74 of file MexIFace.h.

3.3.2.9 VecField

```
template<class T >
using MexIFace::VecField = arma::field<arma::Col<T> >
```

Definition at line 78 of file MexIFace.h.

3.3.2.10 VecStatsT

```
typedef std::map<std::string,arma::Col<double> > MexIFace::VecStatsT
```

A convenient form for reporting dictionaries of named FP vector (or) scalar data to matlab

Definition at line 75 of file MexIFace.h.

3.3.2.11 VectorList

```
template<class T >
using MexIFace::VectorList = std::vector<std::list<T> >
```

Definition at line 77 of file MexIFace.h.

3.3.2.12 VectorVector

```
template<class T >
using MexIFace::VectorVector = std::vector<std::vector<T> >
```

Definition at line 76 of file MexIFace.h.

3.3.2.13 VecVector

```
template<class T >
using MexIFace::VecVector = std::vector<arma::Col<T> >
```

Definition at line 81 of file MexIFace.h.

3.3.3 Constructor & Destructor Documentation

3.3.3.1 MexIFace()

```
MexIFace::MexIFace (
            std::string name )
```

Definition at line 19 of file MexIFace.cpp.

3.3.4 Member Function Documentation

3.3.4.1 callMethod()

```
void MexIFace::callMethod (
            std::string name )  [protected]
```

Calls a named member function on the instance of the wrapped class.

Parameters

name	The name of the method to call, as given to the mexFunction call.

Throws an error if the name is not in the methodmap std::map data structure.

Definition at line 148 of file MexIFace.cpp.

3.3.4.2 callStaticMethod()

```
void MexIFace::callStaticMethod (
            std::string name )  [protected]
```

Calls a named static member function of the wrapped class.

Parameters

name	The name of the static method to call, as given to the mexFunction call.

Throws an error if the name is not in the staticmethodmap std::map data structure.

Definition at line 171 of file MexIFace.cpp.

3.3.4.3 checkDim()

```
void MexIFace::checkDim (
            const mxArray * m,
            int rows,
            int cols ) const  [protected]
```

Checks that a matlab mxArray object has the correct 2D dimensions.

Parameters

m	A pointer to the mxArray to check
rows	the expected number of rows should be >0
cols	the expected number of cols should be >0

Throws an exception if the number of rows or cols do not match

Definition at line 35 of file MexIFace.cpp.

3.3.4.4 checkMaxNumArgs()

```
void MexIFace::checkMaxNumArgs (
            int max_nlhs,
            int max_nrhs ) const  [protected]
```

Checks that the Iface mex function was called with a maximum number of input and output arguments.

Parameters

max_nlhs	The maximum number of Left-hand-side (output) arguments to allow.
max_nrhs	The maximum number of Right-hand-side (input) arguments to allow.

If max_nlhs or max_nrhs arguments are negative the respective side is not checked.

Throws an exception if there are too many lhs or rhs arguments

Definition at line 104 of file MexIFace.cpp.

3.3.4.5 checkMinNumArgs()

```
void MexIFace::checkMinNumArgs (
            int min_nlhs,
            int min_nrhs ) const  [protected]
```

Checks that the Iface mex function was called with a minimum number of input and output arguments.

Parameters

min_nlhs	The minimum number of Left-hand-side (output) arguments to require.
min_nrhs	The minimum number of Right-hand-side (input) arguments to require.

If min_nlhs or min_nrhs arguments are negative the respective side is not checked.

Throws an exception if there are not enough lhs or rhs arguments

Definition at line 80 of file MexIFace.cpp.

3.3.4.6 checkNumArgs()

```
void MexIFace::checkNumArgs (
             int expected_nlhs,
             int expected_nrhs ) const  [protected]
```

Checks that the Iface mex function was called with exactly the expected number of input and output arguments.

Parameters

expected_nlhs	The expected number of Left-hand-side (output) arguments to require.
expected_nrhs	The expected number of Right-hand-side (input) arguments to require.

If expected_nlhs or expected_nrhs arguments are negative the respective side is not checked.

Throws an exception if there are an incorrect number of lhs or rhs arguments.

Definition at line 128 of file MexIFace.cpp.

3.3.4.7 checkSameLastDim()

```
void MexIFace::checkSameLastDim (
             const mxArray * m1,
             const mxArray * m2 ) const  [protected]
```

Checks that two matlab mxArray objects have the same sized last dimension.

Parameters

m1	A pointer to the first mxArray to check
m2	A pointer to the second mxArray to check

Throws an exception if the last dimensions do not match.

Definition at line 57 of file MexIFace.cpp.

3.3.4.8 component_error()

```
void MexIFace::component_error (
             std::string component,
             std::string condition,
             std::string message ) const  [protected]
```

Reports an error condition in a specified component to Matlab using the mexErrMsgIdAndTxt function.

Parameters

component	A string describing the component in which the error was encountered.
condition	A string describing the error condition encountered.
message	An informative message to accompany the error.

Definition at line 334 of file MexIFace.cpp.

3.3.4.9 error()

```
void MexIFace::error (
             std::string condition,
             std::string message ) const  [protected]
```

Reports an error condition to Matlab using the mexErrMsgIdAndTxt function.

Parameters

condition	A string describing the error condition encountered.
message	An informative message to accompany the error.

Definition at line 322 of file MexIFace.cpp.

3.3.4.10 getBool()

```
bool MexIFace::getBool (
             const mxArray * mxdata = nullptr )  [protected]
```

Definition at line 188 of file MexIFace.cpp.

3.3.4.11 getCubeField()

```
template<class ElemT >
MexIFace::CubeField< ElemT > MexIFace::getCubeField (
             const mxArray * mxdata = nullptr )  [protected]
```

Definition at line 547 of file MexIFace.h.

3.3.4.12 getCubeVector()

```
template<class ElemT >
MexIFace::CubeVector< ElemT > MexIFace::getCubeVector (
             const mxArray * mxdata = nullptr )  [protected]
```

Definition at line 679 of file MexIFace.h.

3.3.4.13 getDHyperStack()

```
Hypercube< double > MexIFace::getDHyperStack (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 456 of file MexIFace.h.

3.3.4.14 getDMat()

```
arma::Mat< double > MexIFace::getDMat (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 363 of file MexIFace.h.

3.3.4.15 getDouble()

```
double MexIFace::getDouble (
             const mxArray * mxdata = nullptr )  [protected]
```

Reads a mxArray as a scalar C++ double type.

Parameters

mxdata	The pointer to the mxArray to interpret.

The mxArray must be a floating point type.

Throws an error if the conversion cannot be made.

Definition at line 281 of file MexIFace.cpp.

3.3.4.16 getDoubleStruct()

```
MexIFace::StatsT MexIFace::getDoubleStruct (
              const mxArray * mxdata = nullptr )  [protected]
```

Process a matlab structure returning a StatsT mapping from keys to double.

Parameters

mxdata	A matlab parameter to process. Must be a structure where all values are scalars.

Definition at line 419 of file MexIFace.cpp.

3.3.4.17 getDoubleVecStruct()

```
MexIFace::VecStatsT MexIFace::getDoubleVecStruct (
              const mxArray * mxdata = nullptr )  [protected]
```

Process a matlab structure returning a VecStatsT mapping from keys to column vectors.

Parameters

mxdata	A Matlabn pameter to process. Must be a structure where all values are 1D arrays.

Definition at line 440 of file MexIFace.cpp.

3.3.4.18 getDStack()

```
arma::Cube< double > MexIFace::getDStack (
              const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 411 of file MexIFace.h.

3.3.4.19 getDVec()

```
arma::Col< double > MexIFace::getDVec (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 325 of file MexIFace.h.

3.3.4.20 getFHyperStack()

```
Hypercube< float > MexIFace::getFHyperStack (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 455 of file MexIFace.h.

3.3.4.21 getFloat()

```
float MexIFace::getFloat (
             const mxArray * mxdata = nullptr )  [protected]
```

Reads a mxArray as a scalar C++ float type.

Parameters

mxdata	The pointer to the mxArray to interpret.

The mxArray must be a floating point type.

Throws an error if the conversion cannot be made.

Definition at line 257 of file MexIFace.cpp.

3.3.4.22 getFMat()

```
arma::Mat< float > MexIFace::getFMat (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 362 of file MexIFace.h.

3.3.4.23 getFStack()

```
arma::Cube< float > MexIFace::getFStack (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 410 of file MexIFace.h.

3.3.4.24 getFVec()

```
arma::Col< float > MexIFace::getFVec (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 324 of file MexIFace.h.

3.3.4.25 getHyperStack()

```
template<class ElemT >
Hypercube< ElemT > MexIFace::getHyperStack (
             const mxArray * mxdata = nullptr )  [protected]
```

Create an Hypercube object to directly work with the Matlab data for a 4D array of arbitrary element type.

Uses the ability of the armadillo arrays to interpret raw data passed to it as preallocated column major format. This allows us to open the array data in C++ using Matlab's memory directly instead of having to allocate a separate space and copy.

Parameters

mxdata	The pointer to the mxArray that is to be interpreted as an armadillo array.

Returns

A new Hypercube that interprets the data stored in the mxdata pointer.

Definition at line 425 of file MexIFace.h.

3.3.4.26 getIHyperStack()

```
Hypercube< int32_t > MexIFace::getIHyperStack (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 454 of file MexIFace.h.

3.3.4.27 getIMat()

```
arma::Mat< int32_t > MexIFace::getIMat (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 361 of file MexIFace.h.

3.3.4.28 getInt()

```
int32_t MexIFace::getInt (
             const mxArray * mxdata = nullptr )  [protected]
```

Reads a mxArray as a scalar C++ int32_t type.

Parameters

mxdata	The pointer to the mxArray to interpret.

The mxArray must be a signed 32 or 64 bit integer type.

Throws an error if the conversion cannot be made.

Definition at line 211 of file MexIFace.cpp.

3.3.4.29 getIStack()

```
arma::Cube< int32_t > MexIFace::getIStack (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 409 of file MexIFace.h.

3.3.4.30 getIVec()

```
arma::Col< int32_t > MexIFace::getIVec (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 323 of file MexIFace.h.

3.3.4.31 getMat()

```
template<class ElemT >
arma::Mat< ElemT > MexIFace::getMat (
             const mxArray * mxdata = nullptr )  [protected]
```

Create an armadillo Mat object to directly work with the Matlab data for a 2D array of arbitrary element type.

Uses the ability of the armadillo arrays to interpret raw data passed to it as preallocated column major format. This allows us to open the array data in C++ using Matlab's memory directly instead of having to allocate a separate space and copy.

Parameters

mxdata	The pointer to the mxArray that is to be interpreted as an armadillo array.

Returns

A new armadillo array that interprets the data stored in the mxdata pointer.

Definition at line 340 of file MexIFace.h.

3.3.4.32 getMatField()

```
template<class ElemT >
MexIFace::MatField< ElemT > MexIFace::getMatField (
              const mxArray * mxdata = nullptr )  [protected]
```

Definition at line 507 of file MexIFace.h.

3.3.4.33 getMatVector()

```
template<class ElemT >
MexIFace::MatVector< ElemT > MexIFace::getMatVector (
              const mxArray * mxdata = nullptr )  [protected]
```

Definition at line 639 of file MexIFace.h.

3.3.4.34 getObjectFromHandle()

```
virtual void MexIFace::getObjectFromHandle (
              const mxArray * mxhandle )  [protected], [pure virtual]
```

This is a helper method which saves a pointer to the wrapped class's object in an internal member variable called obj.

This is not templated on the wrapped class type, so it must be implemented by the IFace subclass.

Parameters

mxhandle	The mxArray where the handle is stored.

3.3.4.35 getScalar()

```
template<class ElemT >
ElemT MexIFace::getScalar (
              const mxArray * mxdata = nullptr )  [protected]
```

Definition at line 265 of file MexIFace.h.

3.3.4.36 getStack()

```
template<class ElemT >
arma::Cube< ElemT > MexIFace::getStack (
             const mxArray * mxdata = nullptr )  [protected]
```

Create an armadillo Cube object to directly work with the Matlab data for a 3D array of arbitrary element type.

Uses the ability of the armadillo arrays to interpret raw data passed to it as preallocated column major format. This allows us to open the array data in C++ using Matlab's memory directly instead of having to allocate a separate space and copy.

Parameters

mxdata	The pointer to the mxArray that is to be interpreted as an armadillo array.

Returns

A new armadillo array that interprets the data stored in the mxdata pointer.

Definition at line 380 of file MexIFace.h.

3.3.4.37 getString()

```
std::string MexIFace::getString (
             const mxArray * mxdata = nullptr )  [protected]
```

Reads a mxArray as a string.

Parameters

mxdata	The pointer to the mxArray to interpret.

Throws an error if the conversion cannot be made.

Definition at line 303 of file MexIFace.cpp.

3.3.4.38 getU16HyperStack()

```
Hypercube< uint16_t > MexIFace::getU16HyperStack (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 452 of file MexIFace.h.

3.3.4.39 getU16Mat()

```
arma::Mat< uint16_t > MexIFace::getU16Mat (
            const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 359 of file MexIFace.h.

3.3.4.40 getU16Stack()

```
arma::Cube< uint16_t > MexIFace::getU16Stack (
            const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 407 of file MexIFace.h.

3.3.4.41 getU16Vec()

```
arma::Col< uint16_t > MexIFace::getU16Vec (
            const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 321 of file MexIFace.h.

3.3.4.42 getUHyperStack()

```
Hypercube< uint32_t > MexIFace::getUHyperStack (
            const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 453 of file MexIFace.h.

3.3.4.43 getUMat()

```
arma::Mat< uint32_t > MexIFace::getUMat (
            const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 360 of file MexIFace.h.

3.3.4.44 getUnsigned()

```
uint32_t MexIFace::getUnsigned (
            const mxArray * mxdata = nullptr )  [protected]
```

Reads a mxArray as a scalar C++ uint32_t type.

Parameters

mxdata	The pointer to the mxArray to interpret.

The mxArray must be an unsigned 32 or 64 bit integer type.

Throws an error if the conversion cannot be made.

Definition at line 234 of file MexIFace.cpp.

3.3.4.45 getUStack()

```
arma::Cube< uint32_t > MexIFace::getUStack (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 408 of file MexIFace.h.

3.3.4.46 getUVec()

```
arma::Col< uint32_t > MexIFace::getUVec (
             const mxArray * mxdata = nullptr )  [inline], [protected]
```

Definition at line 322 of file MexIFace.h.

3.3.4.47 getVec()

```
template<class ElemT >
arma::Col< ElemT > MexIFace::getVec (
             const mxArray * mxdata = nullptr )  [protected]
```

Create an armadillo Column vector to directly work with the Matlab data for a 1D array of arbitrary element type.

Uses the ability of the armadillo arrays to interpret raw data passed to it as preallocated column major format. This allows us to open the array data in C++ using Matlab's memory directly instead of having to allocate a separate space and copy.

Parameters

mxdata	The pointer to the mxArray that is to be interpreted as an armadillo array.

Returns

A new armadillo array that interprets the data stored in the mxdata pointer.

Definition at line 299 of file MexIFace.h.

3.3.4.48 getVecField()

```
template<class ElemT >
MexIFace::VecField< ElemT > MexIFace::getVecField (
             const mxArray * mxdata = nullptr )  [protected]
```

Definition at line 464 of file MexIFace.h.

3.3.4.49 getVecVector()

```
template<class ElemT >
MexIFace::VecVector< ElemT > MexIFace::getVecVector (
             const mxArray * mxdata = nullptr )  [protected]
```

Definition at line 596 of file MexIFace.h.

3.3.4.50 makeDHyperStack()

```
Hypercube< double > MexIFace::makeDHyperStack (
             unsigned rows,
             unsigned cols,
             unsigned slices,
             unsigned hyperslices )  [inline], [protected]
```

Definition at line 817 of file MexIFace.h.

3.3.4.51 makeDMat()

```
arma::Mat< double > MexIFace::makeDMat (
             unsigned rows,
             unsigned cols )  [inline], [protected]
```

Definition at line 806 of file MexIFace.h.

3.3.4.52 makeDStack()

```
arma::Cube< double > MexIFace::makeDStack (
             unsigned rows,
             unsigned cols,
             unsigned slices )  [inline], [protected]
```

Definition at line 811 of file MexIFace.h.

3.3.4.53 makeDVec()

```
arma::Col< double > MexIFace::makeDVec (
             unsigned rows )  [inline], [protected]
```

Definition at line 801 of file MexIFace.h.

3.3.4.54 makeFHyperStack()

```
Hypercube< float > MexIFace::makeFHyperStack (
             unsigned rows,
             unsigned cols,
             unsigned slices,
             unsigned hyperslices )  [inline], [protected]
```

Definition at line 816 of file MexIFace.h.

3.3.4.55 makeFMat()

```
arma::Mat< float > MexIFace::makeFMat (
             unsigned rows,
             unsigned cols )  [inline], [protected]
```

Definition at line 805 of file MexIFace.h.

3.3.4.56 makeFStack()

```
arma::Cube< float > MexIFace::makeFStack (
             unsigned rows,
             unsigned cols,
             unsigned slices )  [inline], [protected]
```

Definition at line 810 of file MexIFace.h.

3.3.4.57 makeFVec()

```
arma::Col< float > MexIFace::makeFVec (
             unsigned rows )  [inline], [protected]
```

Definition at line 800 of file MexIFace.h.

3.3.4.58 makeHyperStack()

```
template<class ElemT >
Hypercube< ElemT > MexIFace::makeHyperStack (
             unsigned rows,
             unsigned cols,
             unsigned slices,
             unsigned hyperslices )  [inline], [protected]
```

Definition at line 789 of file MexIFace.h.

3.3.4.59 makeIHyperStack()

```
Hypercube< int32_t > MexIFace::makeIHyperStack (
             unsigned rows,
             unsigned cols,
             unsigned slices,
             unsigned hyperslices )  [inline], [protected]
```

Definition at line 815 of file MexIFace.h.

3.3.4.60 makeIMat()

```
arma::Mat< int32_t > MexIFace::makeIMat (
             unsigned rows,
             unsigned cols )  [inline], [protected]
```

Definition at line 804 of file MexIFace.h.

3.3.4.61 makeIStack()

```
arma::Cube< int32_t > MexIFace::makeIStack (
             unsigned rows,
             unsigned cols,
             unsigned slices )  [inline], [protected]
```

Definition at line 809 of file MexIFace.h.

3.3.4.62 makeIVec()

```
arma::Col< int32_t > MexIFace::makeIVec (
             unsigned rows )  [inline], [protected]
```

Definition at line 799 of file MexIFace.h.

3.3.4.63 makeMat()

```
template<class ElemT >
arma::Mat< ElemT > MexIFace::makeMat (
             unsigned rows,
             unsigned cols )  [inline], [protected]
```

Definition at line 767 of file MexIFace.h.

3.3.4.64 makeStack()

```
template<class ElemT >
arma::Cube< ElemT > MexIFace::makeStack (
             unsigned rows,
             unsigned cols,
             unsigned slices )  [inline], [protected]
```

Definition at line 777 of file MexIFace.h.

3.3.4.65 makeU16HyperStack()

```
Hypercube< uint16_t > MexIFace::makeU16HyperStack (
             unsigned rows,
             unsigned cols,
             unsigned slices,
             unsigned hyperslices )  [inline], [protected]
```

Definition at line 813 of file MexIFace.h.

3.3.4.66 makeUHyperStack()

```
Hypercube< uint32_t > MexIFace::makeUHyperStack (
             unsigned rows,
             unsigned cols,
             unsigned slices,
             unsigned hyperslices )  [inline], [protected]
```

Definition at line 814 of file MexIFace.h.

3.3.4.67 makeUMat()

```
arma::Mat< uint32_t > MexIFace::makeUMat (
             unsigned rows,
             unsigned cols )  [inline], [protected]
```

Definition at line 803 of file MexIFace.h.

3.3.4.68 makeUStack()

```
arma::Cube< uint32_t > MexIFace::makeUStack (
             unsigned rows,
             unsigned cols,
             unsigned slices )  [inline], [protected]
```

Definition at line 808 of file MexIFace.h.

3.3.4.69 makeUVec()

```
arma::Col< uint32_t > MexIFace::makeUVec (
             unsigned rows )  [inline], [protected]
```

Definition at line 798 of file MexIFace.h.

3.3.4.70 makeVec()

```
template<class ElemT >
arma::Col< ElemT > MexIFace::makeVec (
             unsigned nelem )  [inline], [protected]
```

Definition at line 756 of file MexIFace.h.

3.3.4.71 mexFunction()

```
void MexIFace::mexFunction (
             unsigned _nlhs,
             mxArray * _lhs[ ],
             unsigned _nrhs,
             const mxArray * _rhs[ ] )
```

The mexFunction that will be exposed as the entry point for the .mex file.

Parameters

in	*_nlhs*	The number of left-hand-side (input) arguments passed from the Matlab side of the Iface.
in	*_lhs*	The input arguments passed from the Matlab side of the Iface.
in	*_nrhs*	The number of right-hand-side (output) arguments requested from the Matlab side of the Iface.
in,out	*_rhs*	The output arguments requested from the Matlab side of the Iface to be filled in.

This command is the main entry point for the .mex file, and allows the mexFunction to act like a class interface. Special @new, @delete, @static strings allow objects to be created and destroyed and static functions to be called otherwise the command is interpreted as a member function to be called on the given object handle which is expected to be the second argument.

Definition at line 469 of file MexIFace.cpp.

3.3.4.72 objConstruct()

```
virtual void MexIFace::objConstruct ( )  [protected], [pure virtual]
```

Called when the mexFunction gets the @new command, passing on the remaining input arguments.

The rhs should have a single output argument which is the handle (number) which corresponds to the wrapped object.

This pure virtual function must be overloaded by the Iface subclass.

3.3.4.73 objDestroy()

```
virtual void MexIFace::objDestroy ( )  [protected], [pure virtual]
```

Called when the mexFunction gets the @delete command, passing on the remaining input arguments.

The rhs should be empty, and the lhs (input) should only be given the object handle that was created by a @new command.

This pure virtual function must be overloaded by the Iface subclass.

3.3.4.74 outputBool()

```
void MexIFace::outputBool (
             bool val )  [inline], [protected]
```

Definition at line 932 of file MexIFace.h.

3.3.4.75 outputCubeCellArray() [1/2]

```
template<class ElemT >
void MexIFace::outputCubeCellArray (
            const CubeVector< ElemT > & field )  [protected]
```

Definition at line 1057 of file MexIFace.h.

3.3.4.76 outputCubeCellArray() [2/2]

```
template<class ElemT >
void MexIFace::outputCubeCellArray (
            const CubeField< ElemT > & field )  [protected]
```

Definition at line 1072 of file MexIFace.h.

3.3.4.77 outputDMat()

```
void MexIFace::outputDMat (
            const arma::Mat< double > & arr )  [inline], [protected]
```

Definition at line 888 of file MexIFace.h.

3.3.4.78 outputDouble()

```
void MexIFace::outputDouble (
            double val )  [inline], [protected]
```

Definition at line 914 of file MexIFace.h.

3.3.4.79 outputDVec() [1/2]

```
template<int N>
void MexIFace::outputDVec (
            const arma::Col< double >::fixed< N > & arr )  [inline], [protected]
```

Definition at line 850 of file MexIFace.h.

3.3.4.80 outputDVec() [2/2]

```
void MexIFace::outputDVec (
            const arma::Col< double > & arr )  [inline], [protected]
```

Definition at line 859 of file MexIFace.h.

3.3.4.81 outputFVec()

```
void MexIFace::outputFVec (
            const arma::Col< float > & arr )  [inline], [protected]
```

Definition at line 839 of file MexIFace.h.

3.3.4.82 outputIMat()

```
void MexIFace::outputIMat (
            const arma::Mat< int32_t > & arr )  [inline], [protected]
```

Definition at line 878 of file MexIFace.h.

3.3.4.83 outputInt()

```
void MexIFace::outputInt (
            int32_t val )  [inline], [protected]
```

Definition at line 923 of file MexIFace.h.

3.3.4.84 outputMat()

```
template<class ElemT >
void MexIFace::outputMat (
            const arma::Mat< ElemT > & arr )  [protected]
```

Definition at line 868 of file MexIFace.h.

3.3.4.85 outputMatCellArray() [1/2]

```
template<class ElemT >
void MexIFace::outputMatCellArray (
            const MatVector< ElemT > & field )  [protected]
```

Definition at line 1029 of file MexIFace.h.

3.3.4.86 outputMatCellArray() [2/2]

```
template<class ElemT >
void MexIFace::outputMatCellArray (
            const MatField< ElemT > & field )  [protected]
```

Definition at line 1043 of file MexIFace.h.

3.3.4.87 outputMXArray()

```
void MexIFace::outputMXArray (
            mxArray * m )  [inline], [protected]
```

Definition at line 908 of file MexIFace.h.

3.3.4.88 outputSparse()

```
template<class ElemT >
void MexIFace::outputSparse (
            const arma::SpMat< ElemT > & arr )  [protected]
```

Definition at line 943 of file MexIFace.h.

3.3.4.89 outputStack()

```
template<class ElemT >
void MexIFace::outputStack (
            const arma::Cube< ElemT > & arr )  [protected]
```

Definition at line 898 of file MexIFace.h.

3.3.4.90 outputStatsToDoubleVecStruct()

```
void MexIFace::outputStatsToDoubleVecStruct (
            const VecStatsT & stats )  [protected]
```

Outputs a VecStatsT as a matlab structure appended to the method's return value.

Parameters

stats	A VecStatsT type mapping from strings to arrays of doubles.

Definition at line 392 of file MexIFace.cpp.

3.3.4.91 outputStatsToStruct()

```
void MexIFace::outputStatsToStruct (
             const StatsT & stats )  [protected]
```

Outputs a StatsT as a matlab structure appended to the method's return value.

A new matlab struct is created populated with the key/value pairs in stats and then is appended to the function outputs.

Parameters

stats	A StatsT type mapping from strings to scalar doubles.

Definition at line 372 of file MexIFace.cpp.

3.3.4.92 outputVec()

```
template<class ElemT >
void MexIFace::outputVec (
             const arma::Col< ElemT > & arr )  [protected]
```

Definition at line 830 of file MexIFace.h.

3.3.4.93 outputVecCellArray() [1/4]

```
template<class ElemT >
void MexIFace::outputVecCellArray (
             const VectorVector< ElemT > & list )  [protected]
```

Definition at line 969 of file MexIFace.h.

3.3.4.94 outputVecCellArray() [2/4]

```
template<class ElemT >
void MexIFace::outputVecCellArray (
             const VectorList< ElemT > & list )  [protected]
```

Definition at line 986 of file MexIFace.h.

3.3.4.95 outputVecCellArray() [3/4]

```
template<class ElemT >
void MexIFace::outputVecCellArray (
             const VecVector< ElemT > & field )  [protected]
```

Definition at line 1000 of file MexIFace.h.

3.3.4.96 outputVecCellArray() [4/4]

```
template<class ElemT >
void MexIFace::outputVecCellArray (
             const VecField< ElemT > & field )  [protected]
```

Definition at line 1015 of file MexIFace.h.

3.3.4.97 popRhs()

```
void MexIFace::popRhs ( )  [inline], [protected]
```

Remove the first right-hand-side (input) argument as it has already been used to find the correct command.

Definition at line 744 of file MexIFace.h.

3.3.4.98 setArguments()

```
void MexIFace::setArguments (
             unsigned _nlhs,
             mxArray * _lhs[ ],
             unsigned _nrhs,
             const mxArray * _rhs[ ] )  [inline], [protected]
```

Helper function to set the internal copies of the left-hand-side and right-hand-side parameters as they were passed to the mexFunction.

Definition at line 732 of file MexIFace.h.

3.3.5 Member Data Documentation

3.3.5.1 lhs

```
mxArray** MexIFace::lhs  [protected]
```

The lhs (output) arguments to be returned

Definition at line 96 of file MexIFace.h.

3.3.5.2 lhs_idx

```
int MexIFace::lhs_idx =0  [protected]
```

The index of the next lhs argument to write as output

Definition at line 97 of file MexIFace.h.

3.3.5.3 methodmap

```
MethodMap MexIFace::methodmap  [protected]
```

A map from names to wrapped member functions to be called

Definition at line 92 of file MexIFace.h.

3.3.5.4 mex_name

```
std::string MexIFace::mex_name
```

A name to use when reporting errors to Matlab

Definition at line 86 of file MexIFace.h.

3.3.5.5 nlhs

```
unsigned MexIFace::nlhs  [protected]
```

The number of lhs (output) arguments asked for

Definition at line 95 of file MexIFace.h.

3.3.5.6 nrhs

```
unsigned MexIFace::nrhs  [protected]
```

The number of rhs (input) arguments given

Definition at line 98 of file MexIFace.h.

3.3.5.7 rhs

```
const mxArray** MexIFace::rhs  [protected]
```

The rhs (input) arguments given

Definition at line 99 of file MexIFace.h.

3.3.5.8 rhs_idx

```
int MexIFace::rhs_idx =0  [protected]
```

The index of the next rhs argument to read as input

Definition at line 100 of file MexIFace.h.

3.3.5.9 staticmethodmap

```
MethodMap MexIFace::staticmethodmap  [protected]
```

A map from names to wrapped static member functions to be called

Definition at line 93 of file MexIFace.h.

The documentation for this class was generated from the following files:

- MexIFace.h
- MexIFace.cpp

4 File Documentation

4.1 explore.cpp File Reference

Code to analyze the data in an mxArray.

```
#include <cstdio>
#include <cstring>
#include "explore.h"
```

Macros

- #define XFMT_SIZE_T "z"

Functions

- void analyze_cell (const mxArray ∗cell_array_ptr)
- void analyze_structure (const mxArray ∗structure_array_ptr)
- void analyze_string (const mxArray ∗string_array_ptr)
- void analyze_sparse (const mxArray ∗array_ptr)
- void analyze_int8 (const mxArray ∗array_ptr)
- void analyze_uint8 (const mxArray ∗array_ptr)
- void analyze_int16 (const mxArray ∗array_ptr)
- void analyze_uint16 (const mxArray ∗array_ptr)
- void analyze_int32 (const mxArray ∗array_ptr)
- void analyze_uint32 (const mxArray ∗array_ptr)
- void analyze_int64 (const mxArray ∗array_ptr)
- void analyze_uint64 (const mxArray ∗array_ptr)
- void analyze_single (const mxArray ∗array_ptr)
- void analyze_double (const mxArray ∗array_ptr)
- void analyze_logical (const mxArray ∗array_ptr)
- void analyze_full (const mxArray ∗numeric_array_ptr)
- void display_subscript (const mxArray ∗array_ptr, mwSize index)
- void get_characteristics (const mxArray ∗array_ptr)
- mxClassID analyze_class (const mxArray ∗array_ptr)

4.1.1 Detailed Description

Code to analyze the data in an mxArray.

This code is slightly modified version of the example code provided by mathworks for an explore.cpp MEX module. Original Copyright notice attached below.

4.1.2 Macro Definition Documentation

4.1.2.1 XFMT_SIZE_T

```
#define XFMT_SIZE_T "z"
```

Definition at line 26 of file explore.cpp.

4.1.3 Function Documentation

4.1.3.1 analyze_cell()

```
void analyze_cell (
             const mxArray * cell_array_ptr )
```

Definition at line 33 of file explore.cpp.

4.1.3.2 analyze_class()

```
mxClassID analyze_class (
             const mxArray * array_ptr )
```

Definition at line 541 of file explore.cpp.

4.1.3.3 analyze_double()

```
void analyze_double (
             const mxArray * array_ptr )
```

Definition at line 395 of file explore.cpp.

4.1.3.4 analyze_full()

```
void analyze_full (
             const mxArray * numeric_array_ptr )
```

Definition at line 435 of file explore.cpp.

4.1.3.5 analyze_int16()

```
void analyze_int16 (
             const mxArray * array_ptr )
```

Definition at line 248 of file explore.cpp.

4.1.3.6 analyze_int32()

```
void analyze_int32 (
             const mxArray * array_ptr )
```

Definition at line 291 of file explore.cpp.

4.1.3.7 analyze_int64()

```
void analyze_int64 (
            const mxArray * array_ptr )
```

Definition at line 332 of file explore.cpp.

4.1.3.8 analyze_int8()

```
void analyze_int8 (
            const mxArray * array_ptr )
```

Definition at line 206 of file explore.cpp.

4.1.3.9 analyze_logical()

```
void analyze_logical (
            const mxArray * array_ptr )
```

Definition at line 415 of file explore.cpp.

4.1.3.10 analyze_single()

```
void analyze_single (
            const mxArray * array_ptr )
```

Definition at line 374 of file explore.cpp.

4.1.3.11 analyze_sparse()

```
void analyze_sparse (
            const mxArray * array_ptr )
```

Definition at line 173 of file explore.cpp.

4.1.3.12 analyze_string()

```
void analyze_string (
            const mxArray * string_array_ptr )
```

Definition at line 117 of file explore.cpp.

4.1.3.13 analyze_structure()

```
void analyze_structure (
             const mxArray * structure_array_ptr )
```

Definition at line 68 of file explore.cpp.

4.1.3.14 analyze_uint16()

```
void analyze_uint16 (
             const mxArray * array_ptr )
```

Definition at line 269 of file explore.cpp.

4.1.3.15 analyze_uint32()

```
void analyze_uint32 (
             const mxArray * array_ptr )
```

Definition at line 312 of file explore.cpp.

4.1.3.16 analyze_uint64()

```
void analyze_uint64 (
             const mxArray * array_ptr )
```

Definition at line 353 of file explore.cpp.

4.1.3.17 analyze_uint8()

```
void analyze_uint8 (
             const mxArray * array_ptr )
```

Definition at line 227 of file explore.cpp.

4.1.3.18 display_subscript()

```
void display_subscript (
             const mxArray * array_ptr,
             mwSize index )
```

Definition at line 456 of file explore.cpp.

4.1.3.19 get_characteristics()

```
void get_characteristics (
             const mxArray * array_ptr )
```

Definition at line 490 of file explore.cpp.

4.2 explore.h File Reference

```
#include "mex.h"
```

Functions

- void analyze_cell (const mxArray ∗cell_array_ptr)
- void analyze_structure (const mxArray ∗structure_array_ptr)
- void analyze_string (const mxArray ∗string_array_ptr)
- void analyze_sparse (const mxArray ∗array_ptr)
- void analyze_int8 (const mxArray ∗array_ptr)
- void analyze_uint8 (const mxArray ∗array_ptr)
- void analyze_int16 (const mxArray ∗array_ptr)
- void analyze_uint16 (const mxArray ∗array_ptr)
- void analyze_int32 (const mxArray ∗array_ptr)
- void analyze_uint32 (const mxArray ∗array_ptr)
- void analyze_int64 (const mxArray ∗array_ptr)
- void analyze_uint64 (const mxArray ∗array_ptr)
- void analyze_single (const mxArray ∗array_ptr)
- void analyze_double (const mxArray ∗array_ptr)
- void analyze_logical (const mxArray ∗array_ptr)
- void analyze_full (const mxArray ∗numeric_array_ptr)
- void display_subscript (const mxArray ∗array_ptr, mwSize index)
- void get_characteristics (const mxArray ∗array_ptr)
- mxClassID analyze_class (const mxArray ∗array_ptr)

4.2.1 Detailed Description

A header file for the functions provided by Matlab in explore.cpp. See: explore.cpp for copyright notice and documentation

4.2.2 Function Documentation

4.2.2.1 analyze_cell()

```
void analyze_cell (
            const mxArray * cell_array_ptr )
```

Definition at line 33 of file explore.cpp.

4.2.2.2 analyze_class()

```
mxClassID analyze_class (
            const mxArray * array_ptr )
```

Definition at line 541 of file explore.cpp.

4.2.2.3 analyze_double()

```
void analyze_double (
            const mxArray * array_ptr )
```

Definition at line 395 of file explore.cpp.

4.2.2.4 analyze_full()

```
void analyze_full (
            const mxArray * numeric_array_ptr )
```

Definition at line 435 of file explore.cpp.

4.2.2.5 analyze_int16()

```
void analyze_int16 (
            const mxArray * array_ptr )
```

Definition at line 248 of file explore.cpp.

4.2.2.6 analyze_int32()

```
void analyze_int32 (
            const mxArray * array_ptr )
```

Definition at line 291 of file explore.cpp.

4.2.2.7 analyze_int64()

```
void analyze_int64 (
            const mxArray * array_ptr )
```

Definition at line 332 of file explore.cpp.

4.2.2.8 analyze_int8()

```
void analyze_int8 (
            const mxArray * array_ptr )
```

Definition at line 206 of file explore.cpp.

4.2.2.9 analyze_logical()

```
void analyze_logical (
            const mxArray * array_ptr )
```

Definition at line 415 of file explore.cpp.

4.2.2.10 analyze_single()

```
void analyze_single (
            const mxArray * array_ptr )
```

Definition at line 374 of file explore.cpp.

4.2.2.11 analyze_sparse()

```
void analyze_sparse (
            const mxArray * array_ptr )
```

Definition at line 173 of file explore.cpp.

4.2.2.12 analyze_string()

```
void analyze_string (
            const mxArray * string_array_ptr )
```

Definition at line 117 of file explore.cpp.

4.2.2.13 analyze_structure()

```
void analyze_structure (
             const mxArray * structure_array_ptr )
```

Definition at line 68 of file explore.cpp.

4.2.2.14 analyze_uint16()

```
void analyze_uint16 (
             const mxArray * array_ptr )
```

Definition at line 269 of file explore.cpp.

4.2.2.15 analyze_uint32()

```
void analyze_uint32 (
             const mxArray * array_ptr )
```

Definition at line 312 of file explore.cpp.

4.2.2.16 analyze_uint64()

```
void analyze_uint64 (
             const mxArray * array_ptr )
```

Definition at line 353 of file explore.cpp.

4.2.2.17 analyze_uint8()

```
void analyze_uint8 (
             const mxArray * array_ptr )
```

Definition at line 227 of file explore.cpp.

4.2.2.18 display_subscript()

```
void display_subscript (
             const mxArray * array_ptr,
             mwSize index )
```

Definition at line 456 of file explore.cpp.

4.2.2.19 get_characteristics()

```
void get_characteristics (
             const mxArray * array_ptr )
```

Definition at line 490 of file explore.cpp.

4.3 Handle.h File Reference

Helper class and templated functions to represent and manipulate Handles to C++ objects as Matlab mxArrays.

```
#include "mex.h"
#include <cstdint>
#include <string>
#include <typeinfo>
```

Classes

- class Handle< T >

  A class to represent and manipulate handles to C++ classes that can be wrapped as Matlab arrays, allowing C++ objects to persist between Mex calls.

4.3.1 Detailed Description

Helper class and templated functions to represent and manipulate Handles to C++ objects as Matlab mxArrays.

Author

Mark J. Olah (mjo@cs.unm DOT edu)

Date

2013-2017

Copyright

See LICENSE file.

4.4 hypercube.h File Reference

The class declaration and inline and templated functions for hypercube.

```
#include <armadillo>
#include <memory>
#include <vector>
#include <stdexcept>
```

Classes

- class Hypercube< ElemT >

  A class to create a 4D armadillo array that can use externally allocated memory.

Typedefs

- typedef Hypercube< double > hypercube
- typedef Hypercube< float > fhypercube

4.4.1 Detailed Description

The class declaration and inline and templated functions for hypercube.

Author

Mark J. Olah (mjo@cs.unm DOT edu)

Date

2013-2017

Copyright

See LICENSE file.

4.4.2 Typedef Documentation

4.4.2.1 fhypercube

```
typedef Hypercube<float> fhypercube
```

Definition at line 139 of file hypercube.h.

4.4.2.2 hypercube

```
typedef Hypercube<double> hypercube
```

Definition at line 138 of file hypercube.h.

4.5 MexIFace.cpp File Reference

The class definition for MexIFace.

```
#include "MexIFace.h"
#include <algorithm>
#include <sstream>
#include <cstdint>
#include <cctype>
#include <string>
#include <thread>
#include "omp.h"
```

Functions

- bool isSubStruct (MexIFace::StatsT::value_type &p)

  Determines if the keyword p from a StatsT represents a sub-struct.

4.5.1 Detailed Description

The class definition for MexIFace.

Author

Mark J. Olah (mjo@cs.unm DOT edu)

Date

2013-2017

Copyright

See LICENSE file.

4.5.2 Function Documentation

4.5.2.1 isSubStruct()

```
bool isSubStruct (
            MexIFace::StatsT::value_type & p )
```

Determines if the keyword p from a StatsT represents a sub-struct.

Substructs are indicated by using the matlab syntax "subname.param1", "subname.param2", etc. All keys beginning with the same sub-struct name are grouped together into a matalab structure which is stored in the parent structure.

Parameters

p	is the parameter name in the to level StatsT map.

Definition at line 359 of file MexIFace.cpp.

4.6 MexIFace.h File Reference

The class declaration and inline and templated functions for MexIFace.

```
#include <sstream>
#include <map>
#include <vector>
#include <list>
#include <algorithm>
#include <armadillo>
#include <boost/function.hpp>
#include <boost/bind.hpp>
#include "mex.h"
#include "hypercube.h"
#include "Handle.h"
#include "explore.h"
#include "MexUtils.h"
```

Classes

- class MexIFace

  Acts as a base class for implementing a C++ class <-> Matlab class interface.

Macros

- #define MAX_STR_LEN 512

4.6.1 Detailed Description

The class declaration and inline and templated functions for MexIFace.

Author

Mark J. Olah (mjo@cs.unm DOT edu)

Date

2013-2017

Copyright

See LICENSE file.

4.6.2 Macro Definition Documentation

4.6.2.1 MAX_STR_LEN

```
#define MAX_STR_LEN 512
```

The maximum length of string we will accept from Matlab

Definition at line 29 of file MexIFace.h.

4.7 MexUtils.cpp File Reference

Helper functions for working with Matlab mxArrays and mxClassIDs.

```
#include "MexUtils.h"
#include "explore.h"
```

Functions

- const char ∗ get_mx_class_name (const mxArray ∗array)

  Returns a string representation of the mxArray class name.
- const char ∗ get_mx_class_name (mxClassID id)

  Returns a string representation of the class with given mxClassID.
- template<>
  mxClassID get_mx_class< double > ()
- template<>
  mxClassID get_mx_class< float > ()
- template<>
  mxClassID get_mx_class< int8_t > ()
- template<>
  mxClassID get_mx_class< int16_t > ()
- template<>
  mxClassID get_mx_class< int32_t > ()
- template<>
  mxClassID get_mx_class< int64_t > ()
- template<>
  mxClassID get_mx_class< uint8_t > ()
- template<>
  mxClassID get_mx_class< uint16_t > ()
- template<>
  mxClassID get_mx_class< uint32_t > ()
- template<>
  mxClassID get_mx_class< uint64_t > ()
- void exploreMexArgs (int nargs, const mxArray ∗args[ ])

  Given the arguments to a matlab mex function call, print out the details of each arguments.

4.7.1 Detailed Description

Helper functions for working with Matlab mxArrays and mxClassIDs.

Author

Mark J. Olah (mjo@cs.unm DOT edu)

Date

2013-2017

Copyright

See LICENSE file.

4.7.2 Function Documentation

4.7.2.1 exploreMexArgs()

```
void exploreMexArgs (
            int nargs,
            const mxArray * args[] )
```

Given the arguments to a matlab mex function call, print out the details of each arguments.

Parameters

nargs	Number of arguments
args	Array of pointers to mxArray's that were given as arguments to a function call

Uses the explore.cpp methods as provided my Matlab. This is just an interface that is callable without using it directlty as a mexFunction.

Definition at line 70 of file MexUtils.cpp.

4.7.2.2 get_mx_class< double >()

```
template<>
mxClassID get_mx_class< double > ( )
```

Definition at line 58 of file MexUtils.cpp.

4.7.2.3 get_mx_class< float >()

```
template<>
mxClassID get_mx_class< float > ( )
```

Definition at line 59 of file MexUtils.cpp.

4.7.2.4 get_mx_class< int16_t >()

```
template<>
mxClassID get_mx_class< int16_t > ( )
```

Definition at line 61 of file MexUtils.cpp.

4.7.2.5 get_mx_class< int32_t >()

```
template<>
mxClassID get_mx_class< int32_t > ( )
```

Definition at line 62 of file MexUtils.cpp.

4.7.2.6 get_mx_class< int64_t >()

```
template<>
mxClassID get_mx_class< int64_t > ( )
```

Definition at line 63 of file MexUtils.cpp.

4.7.2.7 get_mx_class< int8_t >()

```
template<>
mxClassID get_mx_class< int8_t > ( )
```

Definition at line 60 of file MexUtils.cpp.

4.7.2.8 get_mx_class< uint16_t >()

```
template<>
mxClassID get_mx_class< uint16_t > ( )
```

Definition at line 65 of file MexUtils.cpp.

4.7.2.9 get_mx_class< uint32_t >()

```
template<>
mxClassID get_mx_class< uint32_t > ( )
```

Definition at line 66 of file MexUtils.cpp.

4.7.2.10 get_mx_class< uint64_t >()

```
template<>
mxClassID get_mx_class< uint64_t > ( )
```

Definition at line 67 of file MexUtils.cpp.

4.7.2.11 get_mx_class< uint8_t >()

```
template<>
mxClassID get_mx_class< uint8_t > ( )
```

Definition at line 64 of file MexUtils.cpp.

4.7.2.12 get_mx_class_name() [1/2]

```
const char* get_mx_class_name (
             const mxArray * array )
```

Returns a string representation of the mxArray class name.

Parameters

array	The mxArray to analyze

Returns

String giving class name of array

Definition at line 12 of file MexUtils.cpp.

4.7.2.13 get_mx_class_name() [2/2]

```
const char* get_mx_class_name (
             mxClassID id )
```

Returns a string representation of the class with given mxClassID.

Parameters

id	The mxClassID to get name for

Returns

String giving name of matlab data class id.

Definition at line 37 of file MexUtils.cpp.

4.8 MexUtils.h File Reference

Helper functions for working with Matlab mxArrays and mxClassIDs.

```
#include <cstdint>
#include "mex.h"
```

Functions

- const char ∗ get_mx_class_name (const mxArray ∗array)

  Returns a string representation of the mxArray class name.
- const char ∗ get_mx_class_name (mxClassID id)

  Returns a string representation of the class with given mxClassID.
- template<class T >
  mxClassID get_mx_class ()

  Templated function to returns the matlab mxClassID for given C++ data type.
- template<>
  mxClassID get_mx_class< double > ()
- template<>
  mxClassID get_mx_class< float > ()
- template<>
  mxClassID get_mx_class< int8_t > ()
- template<>
  mxClassID get_mx_class< int16_t > ()
- template<>
  mxClassID get_mx_class< int32_t > ()
- template<>
  mxClassID get_mx_class< int64_t > ()
- template<>
  mxClassID get_mx_class< uint8_t > ()
- template<>
  mxClassID get_mx_class< uint16_t > ()
- template<>
  mxClassID get_mx_class< uint32_t > ()
- template<>
  mxClassID get_mx_class< uint64_t > ()
- void exploreMexArgs (int nargs, const mxArray ∗args[ ])

  Given the arguments to a matlab mex function call, print out the details of each arguments.

4.8.1 Detailed Description

Helper functions for working with Matlab mxArrays and mxClassIDs.

Author

Mark J. Olah (mjo@cs.unm DOT edu)

Date

2013-2017

Copyright

See LICENSE file.

4.8.2 Function Documentation

4.8.2.1 exploreMexArgs()

```
void exploreMexArgs (
             int nargs,
             const mxArray * args[] )
```

Given the arguments to a matlab mex function call, print out the details of each arguments.

Parameters

nargs	Number of arguments
args	Array of pointers to mxArray's that were given as arguments to a function call

Uses the explore.cpp methods as provided my Matlab. This is just an interface that is callable without using it directlty as a mexFunction.

Definition at line 70 of file MexUtils.cpp.

4.8.2.2 get_mx_class()

```
template<class T >
mxClassID get_mx_class ( )
```

Templated function to returns the matlab mxClassID for given C++ data type.

Returns

mxClassID for templated C++ class type.

Definition at line 47 of file MexUtils.h.

4.8.2.3 get_mx_class< double >()

```
template<>
mxClassID get_mx_class< double > ( )
```

Definition at line 58 of file MexUtils.cpp.

4.8.2.4 get_mx_class< float >()

```
template<>
mxClassID get_mx_class< float > ( )
```

Definition at line 59 of file MexUtils.cpp.

4.8.2.5 get_mx_class< int16_t >()

```
template<>
mxClassID get_mx_class< int16_t > ( )
```

Definition at line 61 of file MexUtils.cpp.

4.8.2.6 get_mx_class< int32_t >()

```
template<>
mxClassID get_mx_class< int32_t > ( )
```

Definition at line 62 of file MexUtils.cpp.

4.8.2.7 get_mx_class< int64_t >()

```
template<>
mxClassID get_mx_class< int64_t > ( )
```

Definition at line 63 of file MexUtils.cpp.

4.8.2.8 get_mx_class< int8_t >()

```
template<>
mxClassID get_mx_class< int8_t > ( )
```

Definition at line 60 of file MexUtils.cpp.

4.8.2.9 get_mx_class< uint16_t >()

```
template<>
mxClassID get_mx_class< uint16_t > ( )
```

Definition at line 65 of file MexUtils.cpp.

4.8.2.10 get_mx_class< uint32_t >()

```
template<>
mxClassID get_mx_class< uint32_t > ( )
```

Definition at line 66 of file MexUtils.cpp.

4.8.2.11 get_mx_class< uint64_t >()

```
template<>
mxClassID get_mx_class< uint64_t > ( )
```

Definition at line 67 of file MexUtils.cpp.

4.8.2.12 get_mx_class< uint8_t >()

```
template<>
mxClassID get_mx_class< uint8_t > ( )
```

Definition at line 64 of file MexUtils.cpp.

4.8.2.13 get_mx_class_name() [1/2]

```
const char* get_mx_class_name (
             const mxArray * array )
```

Returns a string representation of the mxArray class name.

Parameters

array	The mxArray to analyze

Returns

String giving class name of array

Definition at line 12 of file MexUtils.cpp.

4.8.2.14 get_mx_class_name() [2/2]

```
const char* get_mx_class_name (
             mxClassID id )
```

Returns a string representation of the class with given mxClassID.

Parameters

id	The mxClassID to get name for

Returns

String giving name of matlab data class id.

Definition at line 37 of file MexUtils.cpp.

Index