jplus-0.4.4

Generated by Doxygen 1.6.3

Sun Jul 15 19:53:35 2012

Contents

Chapter 1

Data Structure Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 jarray::header Struct Reference

J array header.

`#include <jarray.h>`

Data Fields

- **I offset**

  offset of data w.r.t header.

- **I flag**

  flags, jarray sets couple of its own.

- **I maxbytes**

  bytes allocated.

- **I type**

  type of array elements (one of T_XXX) .

- **I refcnt**

  reference count

- **I n**

  number of elements in ravel

- **I rank**

  array rank

- **I shape** [1]

  elements of shape

4.1.1 Detailed Description

J array header.

4.1.2 Field Documentation

4.1.2.1 I jarray::header::flag

flags, jarray sets couple of its own.

Referenced by jarray::getHeader().

4.1.2.2 I jarray::header::maxbytes

bytes allocated.

4.1.2.3 I jarray::header::n

number of elements in ravel

Referenced by jarray::get().

4.1.2.4 I jarray::header::offset

offset of data w.r.t header.

4.1.2.5 I jarray::header::rank

array rank

Referenced by jarray::rank().

4.1.2.6 I jarray::header::refcnt

reference count

Referenced by jarray::getRefcount().

4.1.2.7 I jarray::header::shape[1]

elements of shape

Referenced by jarray::shape(), and jarray::size().

4.1.2.8 I jarray::header::type

type of array elements (one of T_XXX) .

Referenced by jarray::get(), jarray::set(), and jarray::type().

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

- jarray.h

4.2 jarray Class Reference

C++ representation of J array.

`#include <jarray.h>`

Inheritance diagram for jarray:

Collaboration diagram for jarray:

Data Structures

- struct **header**
  J array header.
- struct **MS**
  Layout of two words before every array, responsible for J memory management.
- struct **Z**
  complex type, equivalent to J

Public Types

- enum **errorType** { **ERR_CONV** = 10, **ERR_SHAPE** = 11 }
  error codes
- enum **elementType** {
  T_B01 = 1, **T_LIT** = 2, **T_INT** = 4, **T_FL** = 8,
  T_CMPX = 16 }
  Constants for possible array element types.
- typedef char **B**
  byte type, equivalent to J
- typedef char **C**
  literal type, equivalent to J
- typedef short **S**
  short int type, equivalent to J
- typedef long **I**
  integer type, equivalent to J
- typedef double **D**
  floating point type, equivalent to J

Public Member Functions

- **I getRefcount** () const
  returns the array refcount

- template<class T >
  int **get** (T &v, const int i) const
  Obtains the value of a specified element of ravel.

- template<class T >
  int **set** (const int i, const T v)
  Assigns the value of the specified element of ravel.

- template<class T >
  int **get** (std::vector< T > &v) const
  Fits ravel of the array into the specified STL vector.

- template<class T >
  int **get** (T &v)
  Attempts to fit the whole array into the specified type.

- **jarray** ()
  Invalid array, a valid array can be assigned to it.

- **jarray** (**jengine** *je_, std::istream &in)
  Loads the array from binary representation in a stream.

- **jarray::I esize** () const
  Size (in bytes) of the single element of this array.

- void **addhash** (SHA1 &sha)
  Add this array (both shape and values) to the hash.

- bool **write** (std::ostream &out)
  Writes binary representation of this array into the specified output stream.

- bool **isValid** () const
  Returns true if the array is valid.

- const **I type** () const
  Returns array type.

- const **I rank** () const
  Rank (dimensionality) of the array.

- **I** * **shape** () const
  Returns pointer to the first element of the shape.

- int **extent** (int dimension) const
  Returns the extent of the specified dimension, element of shape.

- void **shape** (std::vector< **I** > &shape) const

  Copies array shape into STL vector.

- const int **size** () const

  Number of elements in ravel.

- **I** * **data** () const

  Returns pointer to the beginning of the array data.

- **jengine** * **getEngine** () const

  Returns pointer to jengine, managing memory of this array.

- **jarray** (**jengine** *je_, void *hdr_)

  Instantiates on top of an existing J array.

- **jarray** (**jengine** *je_, **elementType** type, **I** rank, **I** *shape)

  Creates new multidimensional array of the given J type.

- **jarray** (**jengine** *je_, **elementType** type, const std::vector< **I** > &shape)

  Creates new multidimensional array of the given J type.

- **jarray** (**jengine** *je_, const std::string &str)

  Creates T_LIT vector from C++ string.

- **jarray** (const **jarray** &other)

  Makes a copy of another array (increments refcount).

- **jarray** & **assign** (const **jarray** &other)

  Assigns another array to this one (increments refcount and frees memory, if necessary).

- **jarray** & **operator**= (const **jarray** &other)

  Shortcut to assign.

- bool **operator**== (const **jarray** &rhs) const

  Performs by-element comparison and return true if two arrays are the same.

- ~**jarray** ()

  Decrements refcount, frees array memory, if necessary.

Static Public Member Functions

- static **jarray::I** **esize** (**elementType** type)

  Size (in bytes) of the particular data type.

Protected Member Functions

- bool **allocate** (**jengine** *je_, **elementType** type, const **I** rank, const **I** *shape)

  Allocates new array in memory.

- void **grab** () const

  Increments refcount.

- void **release** ()

  Decrements refcount and frees memory, if necessary.

- **I getHeader** (bool give_up_ownership=true) const

Protected Attributes

- **header** * **hdr**

  Pointer to the array header, NULL for invalid array.

- **jengine** * **je**

  Pointer to jengine, managing the memory of this array.

Friends

- class **jengine**

4.2.1 Detailed Description

C++ representation of J array. This is the direct mapping of J array into C++ domain, for support of typed arrays (with automatic type conversion and convenient element indexing) see **jarray_of_type** (p. 23) template class. Only non-sparse array types are currently supported. This class may operate in standalone mode (managing its memory via malloc), or, if J engine is initalized, it will cooperate with J on memory allocation and make use of J garbage collection (AKA tempstack).

4.2.2 Member Typedef Documentation

4.2.2.1 typedef char jarray::B

byte type, equivalent to J

4.2.2.2 typedef char jarray::C

literal type, equivalent to J

4.2.2.3 typedef double jarray::D

floating point type, equivalent to J

4.2.2.4 typedef long jarray::I

integer type, equivalent to J

4.2.2.5 typedef short jarray::S

short int type, equivalent to J

4.2.3 Member Enumeration Documentation

4.2.3.1 enum jarray::elementType

Constants for possible array element types.

Enumerator:

T_B01 B boolean.
T_LIT C literal (character).
T_INT I integer.
T_FL D double (IEEE floating point).
T_CMPX **Z** (p. 52) complex.

4.2.3.2 enum jarray::errorType

error codes

Enumerator:

ERR_CONV data type conversion error
ERR_SHAPE shape mismatch

4.2.4 Constructor & Destructor Documentation

4.2.4.1 jarray::jarray ()

Invalid array, a valid array can be assigned to it.

4.2.4.2 jarray::jarray (jengine * *je_*, std::istream & *in*)

Loads the array from binary representation in a stream.
The representation can be created by the write method.

Parameters

je jengine to own the array's memory.
in stream to read the array from.

4.2.4.3 jarray::jarray (jengine * *je_*, void * *hdr_*)

Instantiates on top of an existing J array.

Parameters

hdr_ pointer to J array header.

4.2.4.4 jarray::jarray (jengine * *je_*, elementType *type*, I *rank*, I * *shape*)

Creates new multidimensional array of the given J type.

Parameters

je jengine to own the array memory.

type the type of new array (one of T_XXX).

rank rank of the new array.

shape pointer to elements of shape.

4.2.4.5 jarray::jarray (jengine * *je_*, elementType *type*, const std::vector< I > & *shape*)

Creates new multidimensional array of the given J type.

Parameters

je jengine to own the array memory.

type the array type (one of T_XXX).

shape the STL vector, holding elements of shape.

4.2.4.6 jarray::jarray (jengine * *je_*, const std::string & *str*)

Creates T_LIT vector from C++ string.

Parameters

je jengine to own the array memory.

str string the new array will contain.

4.2.4.7 jarray::jarray (const jarray & *other*)

Makes a copy of another array (increments refcount).

Parameters

other the array to copy.

4.2.4.8 jarray::~jarray ()

Decrements refcount, frees array memory, if necessary.

4.2.5 Member Function Documentation

4.2.5.1 void jarray::addhash (SHA1 & *sha*)

Add this array (both shape and values) to the hash.

Parameters

sha hash to add the array to.

4.2.5.2 bool jarray::allocate (jengine * *je_*, elementType *type*, const I *rank*, const I * *shape*) [protected]

Allocates new array in memory.

The memory if either malloc-ed (if the first argument is null), or, if passed non-null je pointer, allocated via jtga function of J engine.

Parameters

je_ jengine to manage the memory of this array.

type type of the new array (one of T_XXXX).

rank rank of the new array.

shape shape of the new array.

Returns

true if success.

Referenced by jarray_of_type< T >::jarray_of_type().

4.2.5.3 jarray& jarray::assign (const jarray & *other*)

Assigns another array to this one (increments refcount and frees memory, if necessary).

Parameters

other array to assign to this one.

Returns

reference to this array.

Referenced by jarray_of_type< T >::jarray_of_type(), and operator=().

4.2.5.4 I* jarray::data () const [inline]

Returns pointer to the beginning of the array data.

Returns

pointer to array data.

References hdr.

Referenced by jarray_of_type< T >::operator()(), and jarray_of_type< T >::operator[]().

4.2.5.5 jarray::I jarray::esize () const

Size (in bytes) of the single element of this array.

4.2.5.6 static jarray::I jarray::esize (elementType *type*) [static]

Size (in bytes) of the particular data type.

Parameters

type one of T_XXX constants).

Returns

size of the array element.

4.2.5.7 int jarray::extent (int *dimension*) const [inline]

Returns the extent of the specified dimension, element of shape.

Parameters

dimension the axis, whose size is requested.

Returns

dimension along the specified axis.

References rank(), and shape().

Referenced by jarray_of_type< T >::operator()().

4.2.5.8 template<class T > int jarray::get (T & *v*) [inline]

Attempts to fit the whole array into the specified type.

Currently, this works for literal arrays, which can be fitted into C++ strings.

Parameters

v place to store the array.

Returns

0 on success.

4.2.5.9 template<class T > int jarray::get (std::vector< T > & *v*) const [inline]

Fits ravel of the array into the specified STL vector.

The type must be convertible. Does copy.

Parameters

v the STL vector to hold the copy of the array.

Returns

0 on success.

References ERR_CONV, hdr, jarray::header::n, T_B01, T_CMPX, T_FL, T_INT, T_LIT, and jarray::header::type.

4.2.5.10 template<class T > int jarray::get (T & *v*, const int *i*) const [inline]

Obtains the value of a specified element of ravel.

The type must be convertible.

Parameters

v the place to store the value.

i zero-based index of the element in ravel.

Returns

0 on success.

References ERR_CONV, hdr, T_B01, T_CMPX, T_FL, T_INT, T_LIT, and jarray::header::type.

Referenced by jarray_of_type< T >::jarray_of_type().

4.2.5.11 jengine* jarray::getEngine () const [inline]

Returns pointer to jengine, managing memory of this array.

Returns

jengine, menaging the memory of this array, or NULL if the memory is managed autonomously (via malloc).

References je.

Referenced by jarray_of_type< T >::jarray_of_type().

4.2.5.12 I jarray::getHeader (bool *give_up_ownership* = true) const [inline, protected]

References jarray::header::flag, and hdr.

4.2.5.13 I jarray::getRefcount () const [inline]

returns the array refcount

Returns

the array refcount.

References hdr, and jarray::header::refcnt.

4.2.5.14 void jarray::grab () const [protected]

Increments refcount.

4.2.5.15 bool jarray::isValid () const [inline]

Returns true if the array is valid.

Returns

true if the array is valid.

References hdr.

4.2.5.16 jarray& jarray::operator= (const jarray & *other*) [inline]

Shortcut to assign.

Parameters

other array to assign to this one.

Returns

reference to this array.

References assign().

4.2.5.17 bool jarray::operator== (const jarray & *rhs*) const

Performs by-element comparison and return true if two arrays are the same.

Parameters

rhs array to compare to.

Returns

true if arrays are the same.

4.2.5.18 const I jarray::rank () const [inline]

Rank (dimensionality) of the array.

Returns

rank of the array.

References hdr, and jarray::header::rank.

Referenced by extent(), jarray_of_type< T >::jarray_of_type(), jarray_of_type< T >::operator()(), shape(), and size().

4.2.5.19 void jarray::release () [protected]

Decrements refcount and frees memory, if necessary.

4.2.5.20 template<class T > int jarray::set (const int *i*, const T *v*) [inline]

Assigns the value of the specified element of ravel.

The type must be convertible.

Parameters

i zero-based index of the element to assign.

v the value to put into the array.

Returns

0 on success.

References ERR_CONV, hdr, T_B01, T_CMPX, T_FL, T_INT, T_LIT, and jarray::header::type.

4.2.5.21 void jarray::shape (std::vector< I > & *shape*) const [inline]

Copies array shape into STL vector.

Parameters

shape the vector to hold the requested shape.

References hdr, rank(), and jarray::header::shape.

4.2.5.22 I* jarray::shape () const [inline]

Returns pointer to the first element of the shape.

Returns

pointer to the shape.

References hdr, and jarray::header::shape.

Referenced by extent(), and jarray_of_type< T >::jarray_of_type().

4.2.5.23 const int jarray::size () const [inline]

Number of elements in ravel.

Returns

number of elements in ravel.

References hdr, rank(), and jarray::header::shape.

Referenced by jarray_of_type< T >::jarray_of_type().

4.2.5.24 const I jarray::type () const [inline]

Returns array type.

One of T_XXX constants.

Returns

type of the array elements.

References hdr, and jarray::header::type.

Referenced by jarray_of_type< T >::jarray_of_type().

4.2.5.25 bool jarray::write (std::ostream & *out*)

Writes binary representation of this array into the specified output stream.

Parameters

out the stream to write the array to.

Returns

true if successfully written.

4.2.6 Friends And Related Function Documentation

4.2.6.1 friend class jengine [friend]

4.2.7 Field Documentation

4.2.7.1 header* jarray::hdr [protected]

Pointer to the array header, NULL for invalid array.

Referenced by data(), get(), getHeader(), getRefcount(), isValid(), rank(), set(), shape(), size(), and type().

4.2.7.2 jengine* jarray::je [protected]

Pointer to jengine, managing the memory of this array.

Referenced by getEngine().

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

- **jarray.h**

4.3 jarray_of_type< T > Class Template Reference

Typed variant of jarray, performs automatic type conversion on instantiation.

`#include <jarray.h>`

Inheritance diagram for jarray_of_type< T >:

Collaboration diagram for jarray_of_type< T >:

Public Member Functions

- **jarray_of_type** (const **jarray** &ja)
  Initialize with type conversion.
- T & **operator[]** (const **I** i)
  Directly access element of ravel.
- const T & **operator[]** (const **I** i) const
  Direct read-only access to an element of ravel.
- **jarray_of_type** (**jengine** *je_)
  Conveniently allocate scalar.
- **jarray_of_type** (**jengine** *je_, const **I** l0)
  Conveniently allocate 1-D array.
- T & **operator()** (const **I** i0)
  conveniently access 1-D array.
- const T & **operator()** (const **I** i0) const
  Convenient read-only access to 1-D array.
- **jarray_of_type** (**jengine** *je_, const **I** l0, const **I** l1)
  Conveniently allocate 2-D array.
- T & **operator()** (const **I** i0, const **I** i1)
  Conveniently access 2-D array.
- const T & **operator()** (const **I** i0, const **I** i1) const
  Convenient read-only access to 2-D array.
- **jarray_of_type** (**jengine** *je_, const **I** l0, const **I** l1, const **I** l2)
  Conveniently allocate 3-D array.
- T & **operator()** (const **I** i0, const **I** i1, const **I** i2)

Conveniently access 3-D array.

- const T & **operator()** (const **I** i0, const **I** i1, const **I** i2) const
  Convenient read-only access to 3-D array.
- **jarray_of_type** (**jengine** *je_, const **I** l0, const **I** l1, const **I** l2, const **I** l3)
  Conveniently allocate 4-D array.
- T & **operator()** (const **I** i0, const **I** i1, const **I** i2, const **I** i3)
  Conveniently access 4-D array.
- const T & **operator()** (const **I** i0, const **I** i1, const **I** i2, const **I** i3) const
  Convenient read-only access to 4-D array.
- **jarray_of_type** (**jengine** *je_, const **I** l0, const **I** l1, const **I** l2, const **I** l3, const **I** l4)
  Conveniently allocate 5-D array.
- T & **operator()** (const **I** i0, const **I** i1, const **I** i2, const **I** i3, const **I** i4)
  Conveniently access 5-D array.
- const T & **operator()** (const **I** i0, const **I** i1, const **I** i2, const **I** i3, const **I** i4) const
  Convenient read-only access to 5-D array.
- **jarray_of_type** (**jengine** *je_, const **I** rank, const **I** *shape)
  Conveniently allocate n-D array.
- **jarray_of_type** (**jengine** *je_, const std::vector< **I** > &shape)
  Conveniently allocate n-D array.
- T & **operator()** (const std::vector< **I** > &subscripts)
  Conveniently access n-D array.
- const T & **operator()** (const std::vector< **I** > &subscripts) const
  Convenient read-only access to n-D array.
- T & **operator()** (const **I** *subscripts)
  Conveniently access n-D array.
- const T & **operator()** (const **I** *subscripts) const
  Convenient read-only access to n-D array.

4.3.1 Detailed Description

template<class T> class jarray_of_type< T >

Typed variant of jarray, performs automatic type conversion on instantiation.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 template<class T > jarray_of_type< T >::jarray_of_type (const jarray & *ja*) [inline]

Initialize with type conversion.

In case no conversion is needed the original array is accessed directly, otherwise a copy is made. To access the array members (subscripting) round backets are used like in Blitz++ and not the square ones, like in Boost. Convenience methods are provided for ranks up to 5, for bigger ranks one has to construct and use for indexing an integer vector.

Parameters

ja jarray, holding the data.

References jarray::allocate(), jarray::assign(), jarray::get(), jarray::getEngine(), jarray::rank(), jarray::shape(), jarray::size(), and jarray::type().

4.3.2.2 template<class T > jarray_of_type< T >::jarray_of_type (jengine * *je_*) [inline]

Conveniently allocate scalar.

Parameters

*je_* jengine to manage memory of this array.

References jarray::allocate(), and jarray::shape().

4.3.2.3 template<class T > jarray_of_type< T >::jarray_of_type (jengine * *je_*, const I *l0*) [inline]

Conveniently allocate 1-D array.

Parameters

*je_* jengine to manage memory of this array.

l0 new array dimension

References jarray::allocate(), and jarray::shape().

4.3.2.4 template<class T > jarray_of_type< T >::jarray_of_type (jengine * *je_*, const I *l0*, const I *l1*) [inline]

Conveniently allocate 2-D array.

Parameters

*je_* jengine to manage memory of this array.

l0 new array 1-st dimension

l1 new array 2-nd dimension

References jarray::allocate(), and jarray::shape().

4.3.2.5 template<class T > jarray_of_type< T >::jarray_of_type (jengine * *je_*, const I *l0*, const I *l1*, const I *l2*) [inline]

Conveniently allocate 3-D array.

Parameters

*je_* jengine to manage memory of this array.

l0 new array 1-st dimension

l1 new array 2-nd dimension

l2 new array 3-rd dimension

References jarray::allocate(), and jarray::shape().

4.3.2.6 template<class T > jarray_of_type< T >::jarray_of_type (jengine * *je_*, const I *l0*, const I *l1*, const I *l2*, const I *l3*) [inline]

Conveniently allocate 4-D array.

Parameters

*je_* jengine to manage memory of this array.

l0 new array 1-st dimension

l1 new array 2-nd dimension

l2 new array 3-rd dimension

l3 new array 4-th dimension

References jarray::allocate(), and jarray::shape().

4.3.2.7 template<class T > jarray_of_type< T >::jarray_of_type (jengine * *je_*, const I *l0*, const I *l1*, const I *l2*, const I *l3*, const I *l4*) [inline]

Conveniently allocate 5-D array.

Parameters

*je_* jengine to manage memory of this array.

l0 new array 1-st dimension

l1 new array 2-nd dimension

l2 new array 3-rd dimension

l3 new array 4-th dimension

l4 new array 5-th dimension

References jarray::allocate(), and jarray::shape().

4.3.2.8 template<class T > jarray_of_type< T >::jarray_of_type (jengine * *je_*, const I *rank*, const I * *shape*) [inline]

Conveniently allocate n-D array.

Parameters

je_ jengine to manage memory of this array.

rank rank of new array.

shape shape of new array.

References jarray::allocate().

4.3.2.9 template<class T > jarray_of_type< T >::jarray_of_type (jengine * *je_*, const std::vector< I > & *shape*) [inline]

Conveniently allocate n-D array.

Parameters

je_ jengine to manage memory of this array.

shape shape of new array.

References jarray::allocate().

4.3.3 Member Function Documentation

4.3.3.1 template<class T > const T& jarray_of_type< T >::operator() (const I * *subscripts*) const [inline]

Convenient read-only access to n-D array.

Be careful, allocating enough subscripts.

Parameters

subscripts subscripts of the element to access.

Returns

const reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.2 template<class T > T& jarray_of_type< T >::operator() (const I * *subscripts*) [inline]

Conveniently access n-D array.

Be careful, allocating enough subscripts.

Parameters

subscripts subscripts of the element to access.

Returns

reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.3 template<class T > const T& jarray_of_type< T >::operator() (const std::vector< I > & *subscripts*) const [inline]

Convenient read-only access to n-D array.

Parameters

subscripts subscripts of the element to access.

Returns

const reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.4 template<class T > T& jarray_of_type< T >::operator() (const std::vector< I > & *subscripts*) [inline]

Conveniently access n-D array.

Parameters

je_ jengine to manage memory of this array.

subscripts subscripts of the element to access.

Returns

reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.5 template<class T > const T& jarray_of_type< T >::operator() (const I *i0*, const I *i1*, const I *i2*, const I *i3*, const I *i4*) const [inline]

Convenient read-only access to 5-D array.

Parameters

i0 zero-based 1-st index of the element to access.

i1 zero-based 2-nd index of the element to access.

i2 zero-based 3-rd index of the element to access.

i3 zero-based 4-th index of the element to access.

i4 zero-based 5-th index of the element to access.

Returns

const reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.6 template<class T > T& jarray_of_type< T >::operator() (const I *i0*, const I *i1*, const I *i2*, const I *i3*, const I *i4*) [inline]

Conveniently access 5-D array.

Parameters

i0 zero-based 1-st index of the element to access.

i1 zero-based 2-nd index of the element to access.

i2 zero-based 3-rd index of the element to access.

i3 zero-based 4-th index of the element to access.

i4 zero-based 5-th index of the element to access.

Returns

reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.7 template<class T > const T& jarray_of_type< T >::operator() (const I *i0*, const I *i1*, const I *i2*, const I *i3*) const [inline]

Convenient read-only access to 4-D array.

Parameters

i0 zero-based 1-st index of the element to access.

i1 zero-based 2-nd index of the element to access.

i2 zero-based 3-rd index of the element to access.

i3 zero-based 4-th index of the element to access.

Returns

const reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.8 template<class T > T& jarray_of_type< T >::operator() (const I *i0*, const I *i1*, const I *i2*, const I *i3*) [inline]

Conveniently access 4-D array.

Parameters

i0 zero-based 1-st index of the element to access.

i1 zero-based 2-nd index of the element to access.

i2 zero-based 3-rd index of the element to access.

i3 zero-based 4-th index of the element to access.

Returns

reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.9 template<class T > const T& jarray_of_type< T >::operator() (const I *i0*, const I *i1*, const I *i2*) const [inline]

Convenient read-only access to 3-D array.

Parameters

i0 zero-based 1-st index of the element to access.

i1 zero-based 2-nd index of the element to access.

i2 zero-based 3-rd index of the element to access.

Returns

const reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.10 template<class T > T& jarray_of_type< T >::operator() (const I *i0*, const I *i1*, const I *i2*) [inline]

Conveniently access 3-D array.

Parameters

i0 zero-based 1-st index of the element to access.

i1 zero-based 2-nd index of the element to access.

i2 zero-based 3-rd index of the element to access.

Returns

reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.11 template<class T > const T& jarray_of_type< T >::operator() (const I *i0*, const I *i1*) const [inline]

Convenient read-only access to 2-D array.

Parameters

i0 zero-based 1-st index of the element to access.

i1 zero-based 2-nd index of the element to access.

Returns

const reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.12 template<class T > T& jarray_of_type< T >::operator() (const I *i0*, const I *i1*) [inline]

Conveniently access 2-D array.

Parameters

i0 zero-based 1-st index of the element to access.

i1 zero-based 2-nd index of the element to access.

Returns

reference to the element.

References jarray::data(), jarray::extent(), and jarray::rank().

4.3.3.13 template<class T > const T& jarray_of_type< T >::operator() (const I *i0*) const [inline]

Convenient read-only access to 1-D array.

Parameters

i0 zero-based index of the element to access.

Returns

const reference to the element.

References jarray::data(), and jarray::rank().

4.3.3.14 template<class T > T& jarray_of_type< T >::operator() (const I *i0*) [inline]

conveniently access 1-D array.

Parameters

i0 zero-based index of the element to access.

Returns

reference to the element.

References jarray::data(), and jarray::rank().

4.3.3.15 template<class T > const T& jarray_of_type< T >::operator[] (const I *i*) const [inline]

Direct read-only access to an element of ravel.

Parameters

i index.

Returns

constant reference to the element.

References jarray::data().

4.3.3.16 template<class T > T& jarray_of_type< T >::operator[] (const I *i*) [inline]

Directly access element of ravel.

Parameters

i index.

Returns

reference to the element.

References jarray::data().

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

- **jarray.h**

4.4 jengine Class Reference

Interface to dynamically loaded J engine.

`#include <jengine.h>`

Inheritance diagram for jengine:

Public Types

- typedef **jarray**(∗ **monad**)(**jarray** y)
  Pointer to function, implementing monadic variant of a verb.
- typedef **jarray**(∗ **dyad**)(**jarray** x, **jarray** y)
  Pointer to function, implementing dyadic variant of a verb.
- typedef **jarray**(∗ **amonad**)(**monad** um, **dyad** ud, **jarray** y)
  Pointer to function, implementing monadic variant of an adverb.
- typedef **jarray**(∗ **adyad**)(**monad** um, **dyad** ud, **jarray** x, **jarray** y)
  Pointer to function, implementing dyadic variant of an adverb.

Public Member Functions

- **jengine** ()
  Initializes J engine.
- ∼**jengine** ()
  Frees J engine memory.
- bool **doJ** (const std::string s)
  Executes J sentence.
- const **jarray get** (const std::string name)
  Retrieves a named array from J.
- bool **set** (const std::string name, **jarray** &value)
  Assigns J name to the specified array.
- int **getError** ()
  J error code for the last unsuccessful operation.
- bool **ok** ()
  Conveniently check that there was no error.
- bool **defVerb** (std::string name, **monad** mf, **dyad** df, int mr=**RMAX**, int lr=**RMAX**, int rr=**RMAX**)

Defines J verb, calling one (or both) of specified C++ functions.

- bool **defAdverb** (std::string name, **amonad** mf, **adyad** df)
  defines J adverb, calling one (or both) of specified C++ functions.
- bool **defScript** (std::string name, int type, std::string code, int mr=**jengine::RMAX**, int lr=**jengine::RMAX**, int rr=**jengine::RMAX**)
  Defines a J script, given by a (possibly multi-libe) string.
- bool **isBuiltin** (std::string name) const
  Checks if given name was defined via defScript.
- std::set< std::string > **getBuiltins** () const
  Returns set of all names defined via defScript.
- **jarray::I PROLOG** ()
  Returns the top of J garbage collection stack.
- **jarray::I EPILOG** (**jarray::I** oldtop)
  Frees all memory, allocated since ttop was recorded.
- void * **EPILOG** (**jarray::I** oldtop, void *hdr)
  Frees all memory, allocated since ttop was recorded.

Static Public Member Functions

- static void **initJlibrary** (std::ostream &)
  Loads and links J dynamic library, must be called once before this class is instantiated.

Static Public Attributes

- static const int **RMAX** = 10000
  Constant to denote the infinite rank.

Protected Member Functions

- void * **GA** (const **jarray::I** t, const **jarray::I** n, const **jarray::I** r, const **jarray::I** *s)
  Allocates J array.
- void **FR** (void *hdr)
  Frees J array memory.

Friends

- class **jarray**

4.4.1 Detailed Description

Interface to dynamically loaded J engine.

4.4.2 Member Typedef Documentation

4.4.2.1 typedef jarray(* jengine::adyad)(monad um, dyad ud, jarray x, jarray y)

Pointer to function, implementing dyadic variant of an adverb.

4.4.2.2 typedef jarray(* jengine::amonad)(monad um, dyad ud, jarray y)

Pointer to function, implementing monadic variant of an adverb.

4.4.2.3 typedef jarray(* jengine::dyad)(jarray x, jarray y)

Pointer to function, implementing dyadic variant of a verb.

4.4.2.4 typedef jarray(* jengine::monad)(jarray y)

Pointer to function, implementing monadic variant of a verb.

4.4.3 Constructor & Destructor Documentation

4.4.3.1 jengine::jengine ()

Initializes J engine.

4.4.3.2 jengine::~jengine ()

Frees J engine memory.

4.4.4 Member Function Documentation

4.4.4.1 bool jengine::defAdverb (std::string *name*, amonad *mf*, adyad *df*)

defines J adverb, calling one (or both) of specified C++ functions.

Works similarly to verb definition, except the functions may now call the verb argument of an adverb monadically or dyadically. Adverbs always have infinite ranks.

Parameters

name name of the new adverb (may already be defined).

mf function to be called when adverb is invoked monadically.

df function to be called when adverb is invoked dyadically.

Returns

"true" on success.

4.4.4.2 bool jengine::defScript (std::string *name*, int *type*, std::string *code*, int *mr* = jengine::RMAX, int *lr* = jengine::RMAX, int *rr* = jengine::RMAX)

Defines a J script, given by a (possibly multi-libe) string.

Parameters

name name of the script.

type type of the script (the left argument of ":" conjunction).

code code of the script (possibly multi-line).

mr monadic rank (infinite by default).

lr left rank (infinite by default).

rr right rank (infinite by default).

Returns

"true" upon success.

4.4.4.3 bool jengine::defVerb (std::string *name*, monad *mf*, dyad *df*, int *mr* = RMAX, int *lr* = RMAX, int *rr* = RMAX)

Defines J verb, calling one (or both) of specified C++ functions.

Either of mf or df can be NULL, meaning the absence of corresponding (monadic/dyadic) case. Ranks, equal to RMAX denote "infinite" rank.

Parameters

name name of the new verb (may already be defined).

mf function to be called when verb is invoked monadically.

df function to be called when verb is invoked dyadically.

mr monadic rank (infinite by default).

lr left rank (infinite by default).

rr right rank (infinite by default).

Returns

"true" on success.

4.4.4.4 bool jengine::doJ (const std::string *s*)

Executes J sentence.

Parameters

s sentence to execute, must be single line.

Returns

0 upon success.

4.4.4.5 void* jengine::EPILOG (jarray::I *oldtop*, void * *hdr*)

Frees all memory, allocated since ttop was recorded.

This function, together with PROLOG allows for more explicit J memory management. Be careful that all jarrays, allocated between prolog and epilog are out of scope (so that their destructors have already been called), otherwise this function has potential to mess things up considerably.

Parameters

oldtop the top of J garbage collection stack, recorded by PROLOG.

pointer to the array header (already in the temp stack), which should not be freed, but lifted to the top of new stack (this array usually contains the result of a verb).

Returns

hdr.

4.4.4.6 jarray::I jengine::EPILOG (jarray::I *oldtop*)

Frees all memory, allocated since ttop was recorded.

This function, together with PROLOG allows for more explicit J memory management. Be careful that all jarrays, allocated between prolog and epilog are out of scope (so that their destructors have already been called), otherwise this function has potential to mess things up considerably.

Parameters

oldtop the top of J garbage collection stack, recorded by PROLOG.

Returns

oldtop.

4.4.4.7 void jengine::FR (void * *hdr*) [protected]

Frees J array memory.

Parameters

hdr pointer to the array header.

4.4.4.8 void* jengine::GA (const jarray::I *t*, const jarray::I *n*, const jarray::I *r*, const jarray::I * *s*) [protected]

Allocates J array.

Parameters

t type of the array.

n number of elements in the array.

r rank of the array.

s shape of the array.

Returns

pointer to header of newly allocated array.

4.4.4.9 const jarray jengine::get (const std::string *name*)

Retrieves a named array from J.

Parameters

name name of the array to retrieve

Returns

the J array with the specified name, invalid if name not defined.

Reimplemented in **jplus** (p. 41).

4.4.4.10 std::set<std::string> jengine::getBuiltins () const

Returns set of all names defined via defScript.

Returns

set of all names, defined via defScript

4.4.4.11 int jengine::getError ()

J error code for the last unsuccessful operation.

Returns

J error code or 0 if no error

4.4.4.12 static void jengine::initJlibrary (std::ostream &) [static]

Loads and links J dynamic library, must be called once before this class is instantiated.

4.4.4.13 bool jengine::isBuiltin (std::string *name*) const

Checks if given name was defined via defScript.

Parameters

name name to check

Returns

true if the name was defined via defScript

4.4.4.14 bool jengine::ok () [inline]

Conveniently check that there was no error.

Returns

true if there was no error.

4.4.4.15 jarray::I jengine::PROLOG ()

Returns the top of J garbage collection stack.

Returns

top of garbage collection stack.

4.4.4.16 bool jengine::set (const std::string *name*, jarray & *value*)

Assigns J name to the specified array.

Parameters

name is the name for the array (may already be defined).

value is the array to be known under the specified name.

Returns

true if success.

4.4.5 Friends And Related Function Documentation

4.4.5.1 friend class jarray [friend]

4.4.6 Field Documentation

4.4.6.1 const int jengine::RMAX = 10000 [static]

Constant to denote the infinite rank.

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

- **jengine.h**

4.5 jplus Class Reference

A J+ script.

`#include <jplus.h>`

Inheritance diagram for jplus:

Collaboration diagram for jplus:

Public Member Functions

- **jplus** ()

  initialize J+ engine with no script

- bool **init** (std::istream &script)

  Load and pre-parse J+ script.

- ~**jplus** ()

  Free allocated memory.

- bool **set** (std::string name, **jarray** data)

  Set value of the specified variable inside J+ environment, mark all dependent variables "dirty".

- **jarray get** (std::string name)

  Perform all the necessary calculations to computate the specified variable, return its value in an array.

- std::vector< std::string > **getProgram** (std::set< std::string > vars)

  Pull minimal set of J sentences for computing a given set of variables, assuming everything is "dirty".

- std::vector< std::string > **getProgram** ()

  Pull all the sentences in J+ script.

Protected Member Functions

- virtual void **libInit** ()

  Called by the constructor before parsing the J+ script.

4.5.1 Detailed Description

A J+ script.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 jplus::jplus ()

initialize J+ engine with no script

4.5.2.2 jplus::~jplus ()

Free allocated memory.

4.5.3 Member Function Documentation

4.5.3.1 jarray jplus::get (std::string *name*)

Perform all the necessary calculations to computate the specified variable, return its value in an array.

Parameters

name name of variable to compute.

Returns

the computed variable value.

Reimplemented from **jengine** (p. 38).

4.5.3.2 std::vector<std::string> jplus::getProgram ()

Pull all the sentences in J+ script.

This pulls all the sentences, assigning value of some variable.

Returns

vector of J sentences in J script.

4.5.3.3 std::vector<std::string> jplus::getProgram (std::set< std::string > *vars*)

Pull minimal set of J sentences for computing a given set of variables, assuming everything is "dirty".

Parameters

vars set of variable names to pull.

Returns

vector of J sentences for computing said variables.

4.5.3.4 bool jplus::init (std::istream & *script*)

Load and pre-parse J+ script.

Parameters

script stream, containing the script to load.

Returns

"true" upon success.

Reimplemented in **yacts** (p. 50).

4.5.3.5 virtual void jplus::libInit () [protected, virtual]

Called by the constructor before parsing the J+ script.

Does nothing by default, but can be overridden to prepend an additional J+ stream with some definitions and/or define a few verbs and adverbs to be used in J+ program.

Reimplemented in **yacts** (p. 50).

4.5.3.6 bool jplus::set (std::string *name*, jarray *data*)

Set value of the specified variable inside J+ environment, mark all dependent variables "dirty".

Parameters

name name of the variable.

data array for the variable to refer to.

Returns

"true" upon success.

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

- **jplus.h**

4.6 jarray::MS Struct Reference

Layout of two words before every array, responsible for J memory management.

`#include <jarray.h>`

Data Fields

- **I * a**
- **S j**
- **C mflag**
- **C unused**

4.6.1 Detailed Description

Layout of two words before every array, responsible for J memory management.

4.6.2 Field Documentation

4.6.2.1 I* jarray::MS::a

4.6.2.2 S jarray::MS::j

4.6.2.3 C jarray::MS::mflag

4.6.2.4 C jarray::MS::unused

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

- **jarray.h**

4.7 trjfile Class Reference

YACTS trajectory file.

`#include <trjfile.h>`

Inheritance diagram for trjfile:

Public Member Functions

- **trjfile** ()

  Instantiates unopened trajectory file.

- bool **open** (std::string path, bool create=true)

  Opens/creates trajectory and positions it before the last frame, or right after the header if trajectory was empty.

- bool **close** ()

  Closes the trajectory.

- bool **good** ()

  Checks that trajectory file is opened and properly positioned.

- bool **saveFrame** (const int **neq**, const double &T, const double *data)

  Saves frame to the file at the current pos, appends if necessary.

- bool **hasNextFrame** ()

  Checks if the file has next frame stored.

- int **getNEQ** ()

  Get number of equations.

- bool **loadFrame** (double &T, double *data)

  Load frame.

- int **size** ()

  Get number of frames in the file.

- bool **toFrame** (int frame)

  Position the file before the specified frame.

Protected Member Functions

- std::streampos **header_size** ()

  header size on disk

- std::streampos **frame_size** ()

frame size on disk

- void **setHeader** (int id, int **neq**)
  patch the header with new data

Protected Attributes

- int **neq**
  Number of equations in the system solved (frame size).
- std::fstream **trj**
  Opened stream to the underlying file.

4.7.1 Detailed Description

YACTS trajectory file. Holds the past and current states of integrated system.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 trjfile::trjfile ()

Instantiates unopened trajectory file.

4.7.3 Member Function Documentation

4.7.3.1 bool trjfile::close ()

Closes the trajectory.

4.7.3.2 std::streampos trjfile::frame_size () [protected]

frame size on disk

4.7.3.3 int trjfile::getNEQ ()

Get number of equations.

Returns

the number of frames or 0 if the file was just created and no frames were stored yet.

4.7.3.4 bool trjfile::good ()

Checks that trajectory file is opened and properly positioned.

Returns

true if everything is ok.

4.7.3.5 bool trjfile::hasNextFrame ()

Checks if the file has next frame stored.

Returns

true if the next frame can be read.

4.7.3.6 std::streampos trjfile::header_size () [protected]

header size on disk

4.7.3.7 bool trjfile::loadFrame (double & *T*, double * *data*)

Load frame.

Parameters

T the variable to hold the time of the frame.

data the array to hold the frame data (should have enough space for **getNEQ()** (p. 45) doubles).

4.7.3.8 bool trjfile::open (std::string *path*, bool *create* = true)

Opens/creates trajectory and positions it before the last frame, or right after the header if trajectory was empty.

Parameters

path filesystem path to the trajectory.

create when true an empty trajectory will be created when file is missing, otherwise this is failure.

Returns

true if everything is ok.

4.7.3.9 bool trjfile::saveFrame (const int *neq*, const double & *T*, const double * *data*)

Saves frame to the file at the current pos, appends if necessary.

If this is the first frame saved, the passed neq parameter _defines_ the number of frames in this trajectory file, then it becomes an error to save frames with different neq.

Parameters

neq number of equations in the system.

T time of the frame.

data frame data.

Returns

true if success.

4.7.3.10 void trjfile::setHeader (int *id*, int *neq*) [protected]

patch the header with new data

Parameters

id reserved.

neq number of equations in the system.

4.7.3.11 int trjfile::size ()

Get number of frames in the file.

Returns

the total number of frames currently in trajectory.

Reimplemented in **yacts** (p. 51).

4.7.3.12 bool trjfile::toFrame (int *frame*)

Position the file before the specified frame.

Parameters

frame position will be set before this frame, 0 positions before the first frame, -1 positions before the last frame.

4.7.4 Field Documentation

4.7.4.1 int trjfile::neq [protected]

Number of equations in the system solved (frame size).

4.7.4.2 std::fstream trjfile::trj [protected]

Opened stream to the underlying file.

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

- **trjfile.h**

4.8 yacts Class Reference

YACTS -- yet another continuous time simulator.

`#include <yacts.h>`

Inheritance diagram for yacts:

Collaboration diagram for yacts:

Public Member Functions

- **yacts** ()

  Initializes YACTS.

- bool **init** (std::istream &script)

  Load YACTS script.

- ~**yacts** ()

  Frees YACTS memory.

- virtual void **libInit** ()

  Initializes YACTS library functions and makes them avaliable to J.

- bool **setOut** (std::string newOut)

  Sets the name of output variable.

- int **REPL** ()

  Enters read(stdin)-eval-print(stdout) loop until the end of stdin.

- std::string **getTrajectoryFilenameBase** (std::string prefix)

  Computes trajectory filename base, using hash of "important" parts of yacts script.

- bool **initTrajectory** (std::string prefix)

  Initializes ODE solver and the state either from the start of the problem or from the last saved trajectory frame.

- int **size** ()

  Returns the number of computed frames in the current trajectory file.

- bool **setFrame** (int iframe)

  Sets specified frame as "current", loads it.

- bool **hasNextFrameStored** ()

  Returns true if the next frame is already in the trajectory file and does not have to be (and will not be) calculated.

- bool **nextFrame** ()

  Advances to the next trajectory frame by computing it (if necessary) and saving (if computed).

- std::string **process** ()

  Computes and returns the "OUT" variable, corresponding to the current frame.

Friends

- int **runTests** ()

4.8.1 Detailed Description

YACTS -- yet another continuous time simulator. It uses J+ to specify a system of ODEs and Sundials library to solve them.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 yacts::yacts ()

Initializes YACTS.

4.8.2.2 yacts::~yacts ()

Frees YACTS memory.

4.8.3 Member Function Documentation

4.8.3.1 std::string yacts::getTrajectoryFilenameBase (std::string *prefix*)

Computes trajectory filename base, using hash of "important" parts of yacts script.

Parameters

prefix trajectory filename prefix.

Returns

trajectory filename base (without extension).

4.8.3.2 bool yacts::hasNextFrameStored ()

Returns true if the next frame is already in the trajectory file and does not have to be (and will not be) calculated.

Returns

"true" if the next frame is in trajectory

4.8.3.3 bool yacts::init (std::istream & *script*)

Load YACTS script.

The script must define the following variables: T, double scalar -- initial(first assignment), current and next frame time; S, double array -- initial(first assignment) and current state; dSdT, double array -- time derivative of state; OUT, character vector -- textual rendering of the current state.

Parameters

script stream to load YACTS script from.

Reimplemented from **jplus** (p. 42).

4.8.3.4 bool yacts::initTrajectory (std::string *prefix*)

Initializes ODE solver and the state either from the start of the problem or from the last saved trajectory frame.

Parameters

prefix trajectory filename prefix.

Returns

true upon success.

4.8.3.5 virtual void yacts::libInit () [virtual]

Initializes YACTS library functions and makes them avaliable to J.

Reimplemented from **jplus** (p. 42).

4.8.3.6 bool yacts::nextFrame ()

Advances to the next trajectory frame by computing it (if necessary) and saving (if computed).

Returns

"true" if all is ok.

4.8.3.7 std::string yacts::process ()

Computes and returns the "OUT" variable, corresponding to the current frame.

Returns

the computed output.

4.8.3.8 int yacts::REPL ()

Enters read(stdin)-eval-print(stdout) loop until the end of stdin.

Returns

0.

4.8.3.9 bool yacts::setFrame (int *iframe*)

Sets specified frame as "current", loads it.

Parameters

iframe index of the frame to become current.

Returns

"true" upon success.

4.8.3.10 bool yacts::setOut (std::string *newOut*)

Sets the name of output variable.

Parameters

newOut new output variable name.

Returns

"true" if success (the name was OK)

4.8.3.11 int yacts::size ()

Returns the number of computed frames in the current trajectory file.

Returns

number of frames in the trajectory

Reimplemented from **trjfile** (p. 47).

4.8.4 Friends And Related Function Documentation

4.8.4.1 int runTests () [friend]

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

- **yacts.h**

4.9 jarray::Z Struct Reference

complex type, equivalent to J

`#include <jarray.h>`

Data Fields

- **D re**
  real part
- **D im**
  imaginary part

4.9.1 Detailed Description

complex type, equivalent to J

4.9.2 Field Documentation

4.9.2.1 D jarray::Z::im

imaginary part

4.9.2.2 D jarray::Z::re

real part

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

- **jarray.h**

Chapter 5

File Documentation

5.1 jarray.h File Reference

```
#include <vector>
#include <string>
#include <complex>
#include <istream>
#include <ostream>
#include <cassert>
#include "sha1.h"
```

Include dependency graph for jarray.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- class **jarray**
  C++ representation of J array.
- struct **jarray::Z**
  complex type, equivalent to J
- struct **jarray::header**
  J array header.
- struct **jarray::MS**
  Layout of two words before every array, responsible for J memory management.

- class **jarray_of_type< T >**
  Typed variant of jarray, performs automatic type conversion on instantiation.

Functions

- std::ostream & **operator<<** (std::ostream &stream, const **jarray** &array)
  prints ASCII representation of the array

5.1.1 Function Documentation

5.1.1.1 std::ostream& operator<< (std::ostream & *stream*, const jarray & *array*)

prints ASCII representation of the array

Parameters

stream stream to print the array to.

array the array to print.

Returns

the reference to stream.

5.2 jengine.h File Reference

```
#include <string>
#include <vector>
#include <set>
#include "util.h"
#include <complex>
#include "sha1.h"
#include <istream>
#include <ostream>
#include <cassert>
```

Include dependency graph for jengine.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- class **jengine**

  Interface to dynamically loaded J engine.

5.3 jplus.h File Reference

```
#include <istream>
#include <vector>
#include <set>
#include "jengine.h"
#include <string>
#include "util.h"
#include "jarray.h"
```

Include dependency graph for jplus.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- class **jplus**
  A J+ script.

5.4 trjfile.h File Reference

`#include <fstream>`

`#include <string>`

Include dependency graph for trjfile.h:

Data Structures

- class **trjfile**

  YACTS trajectory file.

5.5 util.h File Reference

```
#include <complex>
#include <string>
#include "sha1.h"
```

Include dependency graph for util.h:

This graph shows which files directly or indirectly include this file:

Defines

- #define **fftw**(name) fftw_ ## name
- #define **TYPE_STR** ""
- #define **_stdcall**

Typedefs

- typedef double **Real**
- typedef std::complex< double > **Cmplx**
- typedef long double **lReal**

Functions

- int **parseint** (char *str, int &res)
- int **parsepositiveint** (char *str, int &res)
- int **parsedouble** (char *str, double &res)
- int **parsepositivedouble** (char *str, double &res)
- std::string **sha1tostring** (SHA1 &sha1)

  convert sha1 to string

- bool **tol_eq** (double a, double b)

  tolerant comparison of two double numbers

5.5.1 Define Documentation

5.5.1.1 #define _stdcall

5.5.1.2 #define fftw(name) fftw_ ## name

5.5.1.3 #define TYPE_STR ""

5.5.2 Typedef Documentation

5.5.2.1 typedef std::complex<double> Cmplx

5.5.2.2 typedef long double lReal

5.5.2.3 typedef double Real

5.5.3 Function Documentation

5.5.3.1 int parsedouble (char * *str*, double & *res*)

5.5.3.2 int parseint (char * *str*, int & *res*)

5.5.3.3 int parsepositivedouble (char * *str*, double & *res*)

5.5.3.4 int parsepositiveint (char * *str*, int & *res*)

5.5.3.5 std::string sha1tostring (SHA1 & *sha1*)

convert sha1 to string

5.5.3.6 bool tol_eq (double *a*, double *b*)

tolerant comparison of two double numbers

5.6 yacts.h File Reference

```
#include <istream>
#include <fstream>
#include <string>
#include "jplus.h"
#include <vector>
#include <set>
#include "jengine.h"
```

Include dependency graph for yacts.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- class **yacts**

  YACTS -- yet another continuous time simulator.

Index