

jplus-0.4.7

Generated on Fri Nov 29 2024 19:56:09 for jplus-0.4.7 by Doxygen 1.9.8

Fri Nov 29 2024 19:56:09



<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy . . . . .	1
<b>2 Data Structure Index</b>	<b>3</b>
2.1 Data Structures . . . . .	3
<b>3 File Index</b>	<b>5</b>
3.1 File List . . . . .	5
<b>4 Data Structure Documentation</b>	<b>7</b>
4.1 jarray::header Struct Reference . . . . .	7
4.1.1 Detailed Description . . . . .	7
4.1.2 Field Documentation . . . . .	8
4.1.2.1 flag . . . . .	8
4.1.2.2 maxbytes . . . . .	8
4.1.2.3 n . . . . .	8
4.1.2.4 offset . . . . .	8
4.1.2.5 rank . . . . .	8
4.1.2.6 refcnt . . . . .	8
4.1.2.7 shape . . . . .	9
4.1.2.8 type . . . . .	9
4.2 jarray Class Reference . . . . .	9
4.2.1 Detailed Description . . . . .	12
4.2.2 Member Typedef Documentation . . . . .	12
4.2.2.1 B . . . . .	12
4.2.2.2 C . . . . .	12
4.2.2.3 D . . . . .	13
4.2.2.4 I . . . . .	13
4.2.2.5 S . . . . .	13
4.2.3 Member Enumeration Documentation . . . . .	13
4.2.3.1 elementType . . . . .	13
4.2.3.2 errorType . . . . .	13
4.2.4 Constructor & Destructor Documentation . . . . .	14
4.2.4.1 jarray() [1/7] . . . . .	14
4.2.4.2 jarray() [2/7] . . . . .	14
4.2.4.3 jarray() [3/7] . . . . .	14
4.2.4.4 jarray() [4/7] . . . . .	14
4.2.4.5 jarray() [5/7] . . . . .	15
4.2.4.6 jarray() [6/7] . . . . .	15
4.2.4.7 jarray() [7/7] . . . . .	15
4.2.4.8 ~jarray() . . . . .	16
4.2.5 Member Function Documentation . . . . .	16
4.2.5.1 addhash() . . . . .	16

4.2.5.2 allocate()	16
4.2.5.3 assign()	17
4.2.5.4 data()	17
4.2.5.5 esize() [1/2]	17
4.2.5.6 esize() [2/2]	17
4.2.5.7 extent()	18
4.2.5.8 get() [1/3]	18
4.2.5.9 get() [2/3]	19
4.2.5.10 get() [3/3]	19
4.2.5.11 getEngine()	19
4.2.5.12 getHeader()	20
4.2.5.13 getRefCount()	20
4.2.5.14 grab()	20
4.2.5.15 isValid()	20
4.2.5.16 operator=()	20
4.2.5.17 operator==()	21
4.2.5.18 rank()	21
4.2.5.19 release()	21
4.2.5.20 set()	22
4.2.5.21 shape() [1/2]	22
4.2.5.22 shape() [2/2]	22
4.2.5.23 size()	23
4.2.5.24 type()	23
4.2.5.25 write()	23
4.2.6 Friends And Related Symbol Documentation	24
4.2.6.1 jengine	24
4.2.7 Field Documentation	24
4.2.7.1 hdr	24
4.2.7.2 je	24
4.3 jarray_of_type< T > Class Template Reference	24
4.3.1 Detailed Description	28
4.3.2 Constructor & Destructor Documentation	28
4.3.2.1 jarray_of_type() [1/9]	28
4.3.2.2 jarray_of_type() [2/9]	29
4.3.2.3 jarray_of_type() [3/9]	29
4.3.2.4 jarray_of_type() [4/9]	29
4.3.2.5 jarray_of_type() [5/9]	30
4.3.2.6 jarray_of_type() [6/9]	30
4.3.2.7 jarray_of_type() [7/9]	31
4.3.2.8 jarray_of_type() [8/9]	31
4.3.2.9 jarray_of_type() [9/9]	31
4.3.3 Member Function Documentation	32

4.3.3.1 operator>() [1/14]	32
4.3.3.2 operator>() [2/14]	32
4.3.3.3 operator>() [3/14]	33
4.3.3.4 operator>() [4/14]	33
4.3.3.5 operator>() [5/14]	33
4.3.3.6 operator>() [6/14]	34
4.3.3.7 operator>() [7/14]	34
4.3.3.8 operator>() [8/14]	35
4.3.3.9 operator>() [9/14]	35
4.3.3.10 operator>() [10/14]	36
4.3.3.11 operator>() [11/14]	36
4.3.3.12 operator>() [12/14]	37
4.3.3.13 operator>() [13/14]	37
4.3.3.14 operator>() [14/14]	37
4.3.3.15 operator[]() [1/2]	38
4.3.3.16 operator[]() [2/2]	38
4.4 jengine Class Reference	39
4.4.1 Detailed Description	40
4.4.2 Member Typedef Documentation	41
4.4.2.1 adyad	41
4.4.2.2 amonad	41
4.4.2.3 dyad	41
4.4.2.4 monad	41
4.4.3 Constructor & Destructor Documentation	41
4.4.3.1 jengine()	41
4.4.3.2 ~jengine()	41
4.4.4 Member Function Documentation	41
4.4.4.1 defAdverb()	41
4.4.4.2 defScript()	42
4.4.4.3 defVerb()	42
4.4.4.4 doJ()	43
4.4.4.5 EPILOG() [1/2]	43
4.4.4.6 EPILOG() [2/2]	44
4.4.4.7 FR()	44
4.4.4.8 GA()	44
4.4.4.9 get()	45
4.4.4.10 getBuiltins()	45
4.4.4.11 getError()	45
4.4.4.12 initJlibrary()	45
4.4.4.13 isBuiltin()	45
4.4.4.14 ok()	46
4.4.4.15 PROLOG()	46

4.4.4.16 set()	46
4.4.5 Friends And Related Symbol Documentation	47
4.4.5.1 jarray	47
4.4.6 Field Documentation	47
4.4.6.1 RMAX	47
4.5 jplus Class Reference	47
4.5.1 Detailed Description	50
4.5.2 Constructor & Destructor Documentation	50
4.5.2.1 jplus()	50
4.5.2.2 ~jplus()	50
4.5.3 Member Function Documentation	50
4.5.3.1 get()	50
4.5.3.2 getProgram() [1/2]	50
4.5.3.3 getProgram() [2/2]	51
4.5.3.4 init()	51
4.5.3.5 libInit()	51
4.5.3.6 set()	51
4.6 jarray::MS Struct Reference	52
4.6.1 Detailed Description	52
4.6.2 Field Documentation	52
4.6.2.1 a	52
4.6.2.2 j	52
4.6.2.3 mflag	52
4.6.2.4 unused	53
4.7 trfile Class Reference	53
4.7.1 Detailed Description	54
4.7.2 Constructor & Destructor Documentation	54
4.7.2.1 trfile()	54
4.7.3 Member Function Documentation	54
4.7.3.1 close()	54
4.7.3.2 frame_size()	54
4.7.3.3 getNEQ()	55
4.7.3.4 good()	55
4.7.3.5 hasNextFrame()	55
4.7.3.6 header_size()	55
4.7.3.7 loadFrame()	55
4.7.3.8 open()	56
4.7.3.9 saveFrame()	56
4.7.3.10 setHeader()	56
4.7.3.11 size()	57
4.7.3.12 toFrame()	57
4.7.4 Field Documentation	57

4.7.4.1 neq . . . . .	57
4.7.4.2 trj . . . . .	57
4.8 yacts Class Reference . . . . .	58
4.8.1 Detailed Description . . . . .	61
4.8.2 Constructor & Destructor Documentation . . . . .	62
4.8.2.1 yacts() . . . . .	62
4.8.2.2 ~yacts() . . . . .	62
4.8.3 Member Function Documentation . . . . .	62
4.8.3.1 getTrajectoryFilenameBase() . . . . .	62
4.8.3.2 hasNextFrameStored() . . . . .	62
4.8.3.3 init() . . . . .	62
4.8.3.4 initTrajectory() . . . . .	63
4.8.3.5 libInit() . . . . .	63
4.8.3.6 nextFrame() . . . . .	63
4.8.3.7 process() . . . . .	63
4.8.3.8 REPL() . . . . .	64
4.8.3.9 setFrame() . . . . .	64
4.8.3.10 setOut() . . . . .	64
4.8.3.11 size() . . . . .	64
4.8.4 Friends And Related Symbol Documentation . . . . .	65
4.8.4.1 runTests . . . . .	65
4.9 jarray::Z Struct Reference . . . . .	65
4.9.1 Detailed Description . . . . .	65
4.9.2 Field Documentation . . . . .	65
4.9.2.1 im . . . . .	65
4.9.2.2 re . . . . .	65
<b>5 File Documentation</b>	<b>67</b>
5.1 src/jarray.h File Reference . . . . .	67
5.1.1 Function Documentation . . . . .	68
5.1.1.1 operator<<() . . . . .	68
5.2 jarray.h . . . . .	69
5.3 src/jengine.h File Reference . . . . .	74
5.4 jengine.h . . . . .	75
5.5 src/jplus.h File Reference . . . . .	76
5.6 jplus.h . . . . .	77
5.7 src/trjfile.h File Reference . . . . .	78
5.8 trjfile.h . . . . .	78
5.9 src/util.h File Reference . . . . .	79
5.9.1 Macro Definition Documentation . . . . .	80
5.9.1.1 _stdcall . . . . .	80
5.9.1.2 fftw . . . . .	80

---

5.9.1.3 TYPE_STR . . . . .	80
5.9.2 Typedef Documentation . . . . .	80
5.9.2.1 Cmplx . . . . .	80
5.9.2.2 IReal . . . . .	81
5.9.2.3 Real . . . . .	81
5.9.3 Function Documentation . . . . .	81
5.9.3.1 parsedouble() . . . . .	81
5.9.3.2 parseint() . . . . .	81
5.9.3.3 parsepositivedouble() . . . . .	81
5.9.3.4 parsepositiveint() . . . . .	81
5.9.3.5 sha1tostring() . . . . .	81
5.9.3.6 tol_eq() . . . . .	81
5.10 util.h . . . . .	82
5.11 src/yacts.h File Reference . . . . .	82
5.12 yacts.h . . . . .	83
<b>Index</b>	<b>85</b>



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

jarray::header . . . . .	7
jarray . . . . .	9
jarray_of_type< T > . . . . .	24
jengine . . . . .	39
jplus . . . . .	47
yacts . . . . .	58
jarray::MS . . . . .	52
trjfile . . . . .	53
yacts . . . . .	58
jarray::Z . . . . .	65



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<b>jarray::header</b>	J array header . . . . .	7
<b>jarray</b>	C++ representation of J array . . . . .	9
<b>jarray_of_type&lt; T &gt;</b>	Typed variant of jarray, performs automatic type conversion on instantiation . . . . .	24
<b>jengine</b>	Interface to dynamically loaded J engine . . . . .	39
<b>jplus</b>	A J+ script . . . . .	47
<b>jarray::MS</b>	Layout of two words before every array, responsible for J memory management . . . . .	52
<b>trjfile</b>	YACTS trajectory file . . . . .	53
<b>yacts</b>	YACTS – yet another continuous time simulator . . . . .	58
<b>jarray::Z</b>	Complex type, equivalent to J . . . . .	65



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

src/ <b>jarray.h</b> . . . . .	67
src/ <b>jengine.h</b> . . . . .	74
src/ <b>jplus.h</b> . . . . .	76
src/ <b>trjfile.h</b> . . . . .	78
src/ <b>util.h</b> . . . . .	79
src/ <b>yacts.h</b> . . . . .	82



## Chapter 4

# Data Structure Documentation

### 4.1 jarray::header Struct Reference

J array header.

```
#include <src/jarray.h>
```

#### Data Fields

- **offset**  
*offset of data w.r.t header.*
- **flag**  
*flags, jarray sets couple of its own.*
- **maxbytes**  
*bytes allocated.*
- **type**  
*type of array elements (one of T\_XXX) .*
- **refcnt**  
*reference count*
- **n**  
*number of elements in ravel*
- **rank**  
*array rank*
- **shape [1]**  
*elements of shape*

#### 4.1.1 Detailed Description

J array header.

### 4.1.2 Field Documentation

#### 4.1.2.1 flag

`I jarray::header::flag`

flags, jarray sets couple of its own.

Referenced by `jarray::getHeader()`.

#### 4.1.2.2 maxbytes

`I jarray::header::maxbytes`

bytes allocated.

#### 4.1.2.3 n

`I jarray::header::n`

number of elements in ravel

Referenced by `jarray::get()`.

#### 4.1.2.4 offset

`I jarray::header::offset`

offset of data w.r.t header.

Referenced by `jarray::data()`, `jarray::get()`, `jarray::get()`, and `jarray::set()`.

#### 4.1.2.5 rank

`I jarray::header::rank`

array rank

Referenced by `jarray::rank()`.

#### 4.1.2.6 refcnt

`I jarray::header::refcnt`

reference count

Referenced by `jarray::getRefCount()`.



## 4.1.2.7 shape

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

elements of shape

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

## 4.1.2.8 type

```
I jarray::header::type
```

type of array elements (one of T\_XXX) .

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

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

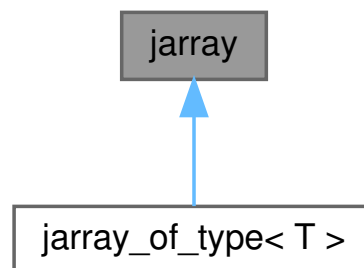
- `src/ jarray.h`

## 4.2 jarray Class Reference

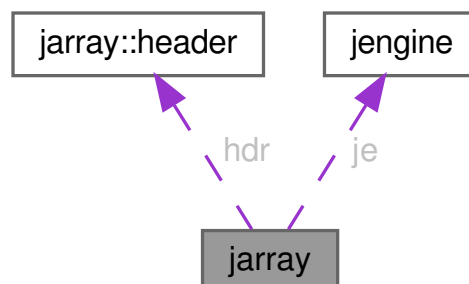
C++ representation of J array.

```
#include <src/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::l 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.24) 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 initialized, 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 B

```
typedef char jarray::B
```

byte type, equivalent to J

### 4.2.2.2 C

```
typedef char jarray::C
```

literal type, equivalent to J

#### 4.2.2.3 D

```
typedef double jarray::D
```

floating point type, equivalent to J

#### 4.2.2.4 I

```
typedef long jarray::I
```

integer type, equivalent to J

#### 4.2.2.5 S

```
typedef short jarray::S
```

short int type, equivalent to J

### 4.2.3 Member Enumeration Documentation

#### 4.2.3.1 elementType

```
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	<b>Z</b> (p. 65) complex

#### 4.2.3.2 errorType

```
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()` [1/7]

```
jarray::jarray ( )
```

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

### 4.2.4.2 `jarray()` [2/7]

```
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

<code>je_</code>	jengine to own the array's memory.
<code>in</code>	stream to read the array from.

### 4.2.4.3 `jarray()` [3/7]

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

Instantiates on top of an existing J array.

#### Parameters

<code>je_</code>	jengine to own the array memory.
<code>hdr_</code>	pointer to J array header.

### 4.2.4.4 `jarray()` [4/7]

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

Creates new multidimensional array of the given J type.

## Parameters

<i>je_</i>	jengine to own the array memory.
<i>type</i>	the type of new array (one of T_XXX).
<i>rank</i>	rank of the new array.
<i>shape</i>	pointer to elements of shape.

## 4.2.4.5 jarray() [5/7]

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

Creates new multidimensional array of the given J type.

## Parameters

<i>je_</i>	jengine to own the array memory.
<i>type</i>	the array type (one of T_XXX).
<i>shape</i>	the STL vector, holding elements of shape.

## 4.2.4.6 jarray() [6/7]

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

Creates T\_LIT vector from C++ string.

## Parameters

<i>je_↔ _↔</i>	jengine to own the array memory.
<i>str</i>	string the new array will contain.

## 4.2.4.7 jarray() [7/7]

```
jarray::jarray (
    const jarray & other )
```

Makes a copy of another array (increments refcount).

## Parameters

<i>other</i>	the array to copy.
--------------	--------------------

#### 4.2.4.8 ~jarray()

```
jarray::~~jarray ( )
```

Decrements refcount, frees array memory, if necessary.

### 4.2.5 Member Function Documentation

#### 4.2.5.1 addhash()

```
void jarray::addhash (
    SHA1 & sha )
```

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

##### Parameters

<i>sha</i>	hash to add the array to.
------------	---------------------------

#### 4.2.5.2 allocate()

```
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

<i>je_</i>	jengine to manage the memory of this array.
<i>type</i>	type of the new array (one of T_XXXX).
<i>rank</i>	rank of the new array.
<i>shape</i>	shape of the new array.

##### Returns

true if success.

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





**Parameters**

<i>type</i>	one of T_XXX constants).
-------------	--------------------------

**Returns**

size of the array element.

**4.2.5.7 extent()**

```
int jarray::extent (
    int dimension ) const [inline]
```

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

**Parameters**

<i>dimension</i>	the axis, whose size is requested.
------------------	------------------------------------

**Returns**

dimension along the specified axis.

References **rank()**, and **shape()**.

Referenced by **jarray\_of\_type< T >::operator()()**, **jarray\_of\_type< T >::operator()()**, **jarray\_of\_type< T >::operator()()**, **jarray\_of\_type< T >::operator()()**, **jarray\_of\_type< T >::operator()()**, **jarray\_of\_type< T >::operator()()**, **jarray\_of\_type< T >::operator()()**, **jarray\_of\_type< T >::operator()()**, **jarray\_of\_type< T >::operator()()**, **jarray\_of\_type< T >::operator()()**, **jarray\_of\_type< T >::operator()()**, and **jarray\_of\_type< T >::operator()()**.

**4.2.5.8 get() [1/3]**

```
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**

<i>v</i>	the STL vector to hold the copy of the array.
----------	---

**Returns**

0 on success.

References **ERR\_CONV**, **hdr**, **jarray::header::n**, **jarray::header::offset**, **T\_B01**, **T\_CMPX**, **T\_FL**, **T\_INT**, **T\_LIT**, and **jarray::header::type**.

#### 4.2.5.9 get() [2/3]

```
template< class T >
int jarray::get (
    T & v )
```

Attempts to fit the whole array into the specified type.

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

##### Parameters

<i>v</i>	place to store the array.
----------	---------------------------

##### Returns

0 on success.

#### 4.2.5.10 get() [3/3]

```
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

<i>v</i>	the place to store the value.
<i>i</i>	zero-based index of the element in ravel.

##### Returns

0 on success.

References **ERR\_CONV**, **hdr**, **jarray::header::offset**, **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 getEngine()

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

Returns pointer to jengine, managing memory of this array.

**Returns**

jengine, managing 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 getHeader()**

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

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

**4.2.5.13 getRefCount()**

```
I jarray::getRefCount ( ) const [inline]
```

returns the array refcount

**Returns**

the array refcount.

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

**4.2.5.14 grab()**

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

Increments refcount.

**4.2.5.15 isValid()**

```
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 operator=()**

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

Shortcut to assign.

## Parameters

<i>other</i>	array to assign to this one.
--------------	------------------------------

## Returns

reference to this array.

References **assign()**.

## 4.2.5.17 operator==()

```
bool jarray::operator== (
    const jarray & rhs ) const
```

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

## Parameters

<i>rhs</i>	array to compare to.
------------	----------------------

## Returns

true if arrays are the same.

## 4.2.5.18 rank()

```
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()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **jarray\_of\_type< T >::operator()**, **shape()**, and **size()**.

## 4.2.5.19 release()

```
void jarray::release ( ) [protected]
```

Decrements refcount and frees memory, if necessary.

#### 4.2.5.20 set()

```
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>i</i>	zero-based index of the element to assign.
<i>v</i>	the value to put into the array.

##### Returns

0 on success.

References **ERR\_CONV**, **hdr**, **jarray::header::offset**, **T\_B01**, **T\_CMPX**, **T\_FL**, **T\_INT**, **T\_LIT**, and **jarray↵  
::header::type**.

Referenced by **jarray\_of\_type< T >::jarray\_of\_type()**.

#### 4.2.5.21 shape() [1/2]

```
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()**, **jarray\_of\_type< T >::jarray\_of\_type()**, **jarray\_of\_type< T >::jarray\_of\_type()**, **jarray\_of\_type< T >::jarray\_of\_type()**, **jarray\_of\_type< T >↵  
::jarray\_of\_type()**, **jarray\_of\_type< T >::jarray\_of\_type()**, **jarray\_of\_type< T >::jarray\_of\_type()**, **jarray\_of\_type< T >↵  
of\_type< T >::jarray\_of\_type()**, **jarray\_of\_type< T >::jarray\_of\_type()**, and **shape()**.

#### 4.2.5.22 shape() [2/2]

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

Copies array shape into STL vector.

## Parameters

<i>shape</i>	the vector to hold the requested shape.
--------------	---

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

**4.2.5.23 size()**

```
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 type()**

```
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 write()**

```
bool jarray::write (
    std::ostream & out )
```

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

## Parameters

<i>out</i>	the stream to write the array to.
------------	-----------------------------------

**Returns**

true if successfully written.

**4.2.6 Friends And Related Symbol Documentation****4.2.6.1 jengine**

```
friend class jengine [friend]
```

**4.2.7 Field Documentation****4.2.7.1 hdr**

```
header* jarray::hdr [protected]
```

Pointer to the array header, NULL for invalid array.

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

**4.2.7.2 je**

```
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:

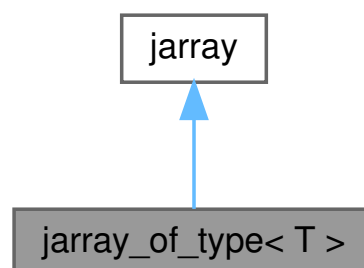
- src/ **jarray.h**

**4.3 jarray\_of\_type< T > Class Template Reference**

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

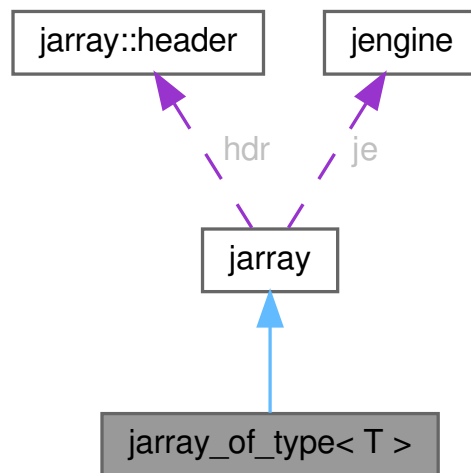
```
#include <src/jarray.h>
```

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** **i0**, **const I** **i1**, **const I** **i2**, **const I** **i3**, **const I** **i4**)

*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.*

## Public Member Functions inherited from jarray

- **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::l 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.*

### Additional Inherited Members

### Public Types inherited from jarray

- **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*

## Static Public Member Functions inherited from `jarray`

- **static `jarray::l esize ( elementType type)`**

*Size (in bytes) of the particular data type.*

## Protected Member Functions inherited from `jarray`

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

*Allocates new array in memory.*

- **`void grab () const`**

*Increments refcount.*

- **`void release ()`**

*Decrements refcount and frees memory, if necessary.*

- **`l getHeader ( bool give_up_ownership= true) const`**

## Protected Attributes inherited from `jarray`

- **`header * hdr`**

*Pointer to the array header, NULL for invalid array.*

- **`jengine * je`**

*Pointer to jengine, managing the memory of this 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 `jarray_of_type()` [1/9]

```
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 brackets 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

<code>ja</code>	jarray, holding the data.
-----------------	---------------------------

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

#### 4.3.2.2 `jarray_of_type()` [2/9]

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

Conveniently allocate scalar.

##### Parameters

<code>je_</code>	jengine to manage memory of this array.
------------------	---

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

#### 4.3.2.3 `jarray_of_type()` [3/9]

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

Conveniently allocate 1-D array.

##### Parameters

<code>je_</code>	jengine to manage memory of this array.
<code>l0</code>	new array dimension

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

#### 4.3.2.4 `jarray_of_type()` [4/9]

```
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**

<i>je</i> ↔ _↔	jengine to manage memory of this array.
<i>l0</i>	new array 1-st dimension
<i>l1</i>	new array 2-nd dimension

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

**4.3.2.5 jarray\_of\_type() [5/9]**

```
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**

<i>je</i> ↔ _↔	jengine to manage memory of this array.
<i>l0</i>	new array 1-st dimension
<i>l1</i>	new array 2-nd dimension
<i>l2</i>	new array 3-rd dimension

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

**4.3.2.6 jarray\_of\_type() [6/9]**

```
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**

<i>je</i> ↔ _↔	jengine to manage memory of this array.
<i>l0</i>	new array 1-st dimension
<i>l1</i>	new array 2-nd dimension
<i>l2</i>	new array 3-rd dimension
<i>l3</i>	new array 4-th dimension

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

#### 4.3.2.7 `jarray_of_type()` [7/9]

```
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

<code>je_</code> <code>_</code>	jengine to manage memory of this array.
<code>l0</code>	new array 1-st dimension
<code>l1</code>	new array 2-nd dimension
<code>l2</code>	new array 3-rd dimension
<code>l3</code>	new array 4-th dimension
<code>l4</code>	new array 5-th dimension

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

#### 4.3.2.8 `jarray_of_type()` [8/9]

```
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

<code>je_</code>	jengine to manage memory of this array.
<code>rank</code>	rank of new array.
<code>shape</code>	shape of new array.

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

#### 4.3.2.9 `jarray_of_type()` [9/9]

```
template< class T >
jarray_of_type< T > ::jarray_of_type (
```

```

jengine * je_,
const std::vector< I > & shape ) [inline]

```

Conveniently allocate n-D array.

#### Parameters

<i>je_</i>	jengine to manage memory of this array.
<i>shape</i>	shape of new array.

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

## 4.3.3 Member Function Documentation

### 4.3.3.1 `operator>()` [1/14]

```

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

```

Conveniently access n-D array.

Be careful, allocating enough subscripts.

#### Parameters

<i>subscripts</i>	subscripts of the element to access.
-------------------	--------------------------------------

#### Returns

reference to the element.

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

### 4.3.3.2 `operator>()` [2/14]

```

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

<i>subscripts</i>	subscripts of the element to access.
-------------------	--------------------------------------



**Returns**

const reference to the element.

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

**4.3.3.3 operator>() [3/14]**

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

conveniently access 1-D array.

**Parameters**

<i>i0</i>	zero-based index of the element to access.
-----------	--

**Returns**

reference to the element.

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

**4.3.3.4 operator>() [4/14]**

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

Convenient read-only access to 1-D array.

**Parameters**

<i>i0</i>	zero-based index of the element to access.
-----------	--

**Returns**

const reference to the element.

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

**4.3.3.5 operator>() [5/14]**

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

Conveniently access 2-D array.

**Parameters**

<i>i0</i>	zero-based 1-st index of the element to access.
<i>i1</i>	zero-based 2-nd index of the element to access.

**Returns**

reference to the element.

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

**4.3.3.6 operator() [6/14]**

```
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**

<i>i0</i>	zero-based 1-st index of the element to access.
<i>i1</i>	zero-based 2-nd index of the element to access.

**Returns**

const reference to the element.

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

**4.3.3.7 operator() [7/14]**

```
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**

<i>i0</i>	zero-based 1-st index of the element to access.
<i>i1</i>	zero-based 2-nd index of the element to access.
<i>i2</i>	zero-based 3-rd index of the element to access.

**Returns**

reference to the element.

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

**4.3.3.8 operator>() [8/14]**

```
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**

<i>i0</i>	zero-based 1-st index of the element to access.
<i>i1</i>	zero-based 2-nd index of the element to access.
<i>i2</i>	zero-based 3-rd index of the element to access.

**Returns**

const reference to the element.

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

**4.3.3.9 operator>() [9/14]**

```
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**

<i>i0</i>	zero-based 1-st index of the element to access.
<i>i1</i>	zero-based 2-nd index of the element to access.
<i>i2</i>	zero-based 3-rd index of the element to access.
<i>i3</i>	zero-based 4-th index of the element to access.

**Returns**

reference to the element.

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

**4.3.3.10 operator>() [10/14]**

```
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**

<i>i0</i>	zero-based 1-st index of the element to access.
<i>i1</i>	zero-based 2-nd index of the element to access.
<i>i2</i>	zero-based 3-rd index of the element to access.
<i>i3</i>	zero-based 4-th index of the element to access.

**Returns**

const reference to the element.

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

**4.3.3.11 operator>() [11/14]**

```
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**

<i>i0</i>	zero-based 1-st index of the element to access.
<i>i1</i>	zero-based 2-nd index of the element to access.
<i>i2</i>	zero-based 3-rd index of the element to access.
<i>i3</i>	zero-based 4-th index of the element to access.
<i>i4</i>	zero-based 5-th index of the element to access.

**Returns**

reference to the element.

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

**4.3.3.12 operator>() [12/14]**

```
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**

<i>i0</i>	zero-based 1-st index of the element to access.
<i>i1</i>	zero-based 2-nd index of the element to access.
<i>i2</i>	zero-based 3-rd index of the element to access.
<i>i3</i>	zero-based 4-th index of the element to access.
<i>i4</i>	zero-based 5-th index of the element to access.

**Returns**

const reference to the element.

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

**4.3.3.13 operator>() [13/14]**

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

Conveniently access n-D array.

**Parameters**

<i>subscripts</i>	subscripts of the element to access.
-------------------	--------------------------------------

**Returns**

reference to the element.

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

**4.3.3.14 operator>() [14/14]**

```
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

<i>subscripts</i>	subscripts of the element to access.
-------------------	--------------------------------------

## Returns

const reference to the element.

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

4.3.3.15 `operator[]()` [1/2]

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

Directly access element of ravel.

## Parameters

<i>i</i>	index.
----------	--------

## Returns

reference to the element.

References `jarray::data()`.

4.3.3.16 `operator[]()` [2/2]

```
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>i</i>	index.
----------	--------

## Returns

constant reference to the element.

References `jarray::data()`.

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

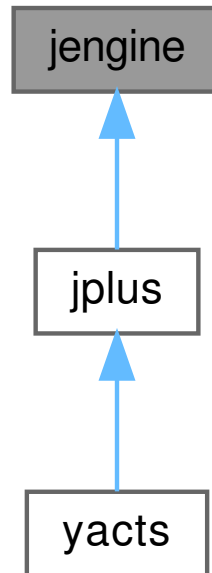
- `src/jarray.h`

## 4.4 jengine Class Reference

Interface to dynamically loaded J engine.

```
#include <src/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-line) 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::l PROLOG ()**  
*Returns the top of J garbage collection stack.*
- **jarray::l EPILOG ( jarray::l oldtop)**  
*Frees all memory, allocated since ttop was recorded.*
- **void \* EPILOG ( jarray::l 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::l t, const jarray::l n, const jarray::l r, const jarray::l 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 adyad

```
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 amonad

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

Pointer to function, implementing monadic variant of an adverb.

### 4.4.2.3 dyad

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

Pointer to function, implementing dyadic variant of a verb.

### 4.4.2.4 monad

```
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::jengine ( )
```

Initializes J engine.

### 4.4.3.2 ~jengine()

```
jengine::~~jengine ( )
```

Frees J engine memory.

## 4.4.4 Member Function Documentation

### 4.4.4.1 defAdverb()

```
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

<i>name</i>	name of the new adverb (may already be defined).
<i>mf</i>	function to be called when adverb is invoked monadically.
<i>df</i>	function to be called when adverb is invoked dyadically.

## Returns

"true" on success.

## 4.4.4.2 defScript()

```
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-line) string.

## Parameters

<i>name</i>	name of the script.
<i>type</i>	type of the script (the left argument of ":" conjunction).
<i>code</i>	code of the script (possibly multi-line).
<i>mr</i>	monadic rank (infinite by default).
<i>lr</i>	left rank (infinite by default).
<i>rr</i>	right rank (infinite by default).

## Returns

"true" upon success.

## 4.4.4.3 defVerb()

```
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

<i>name</i>	name of the new verb (may already be defined).
<i>mf</i>	function to be called when verb is invoked monadically.
<i>df</i>	function to be called when verb is invoked dyadically.
<i>mr</i>	monadic rank (infinite by default).
<i>lr</i>	left rank (infinite by default).
<i>rr</i>	right rank (infinite by default).

## Returns

"true" on success.

## 4.4.4.4 doJ()

```
bool jengine::doJ (
    const std::string s )
```

Executes J sentence.

## Parameters

<i>s</i>	sentence to execute, must be single line.
----------	---

## Returns

0 upon success.

## 4.4.4.5 EPILOG() [1/2]

```
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 epilg are out of scope (so that their destructors have already been called), otherwise this function has potential to mess things up considerably.

## Parameters

<i>oldtop</i>	the top of J garbage collection stack, recorded by PROLOG.
---------------	--

## Returns

oldtop.

#### 4.4.4.6 EPILOG() [2/2]

```
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

<i>oldtop</i>	the top of J garbage collection stack, recorded by PROLOG.
<i>hdr</i>	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*.

#### 4.4.4.7 FR()

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

Frees J array memory.

##### Parameters

<i>hdr</i>	pointer to the array header.
------------	------------------------------

#### 4.4.4.8 GA()

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

Allocates J array.

##### Parameters

<i>t</i>	type of the array.
<i>n</i>	number of elements in the array.
<i>r</i>	rank of the array.
<i>s</i>	shape of the array.

**Returns**

pointer to header of newly allocated array.

**4.4.4.9 get()**

```
const jarray jengine::get (
    const std::string name )
```

Retrieves a named array from J.

**Parameters**

<i>name</i>	name of the array to retrieve
-------------	-------------------------------

**Returns**

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

**4.4.4.10 getBuiltins()**

```
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 getError()**

```
int jengine::getError ( )
```

J error code for the last unsuccessful operation.

**Returns**

J error code or 0 if no error

Referenced by **ok()**.

**4.4.4.12 initJlibrary()**

```
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 isBuiltin()**

```
bool jengine::isBuiltin (
    std::string name ) const
```

Checks if given name was defined via defScript.

**Parameters**

<i>name</i>	name to check
-------------	---------------

**Returns**

true if the name was defined via defScript

**4.4.4.14 ok()**

```
bool jengine::ok ( ) [inline]
```

Conveniently check that there was no error.

**Returns**

true if there was no error.

References **getError()**.

**4.4.4.15 PROLOG()**

```
jarray::I jengine::PROLOG ( )
```

Returns the top of J garbage collection stack.

**Returns**

top of garbage collection stack.

**4.4.4.16 set()**

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

Assigns J name to the specified array.

**Parameters**

<i>name</i>	is the name for the array (may already be defined).
<i>value</i>	is the array to be known under the specified name.

**Returns**

true if success.

## 4.4.5 Friends And Related Symbol Documentation

### 4.4.5.1 jarray

```
friend class jarray [friend]
```

## 4.4.6 Field Documentation

### 4.4.6.1 RMAX

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

Constant to denote the infinite rank.

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

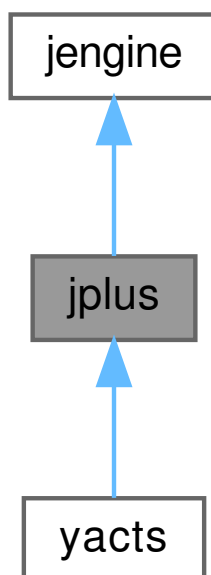
- src/ **jengine.h**

## 4.5 jplus Class Reference

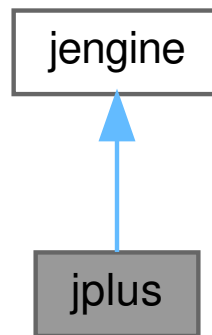
A J+ script.

```
#include <src/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 compute 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.*

### Public Member Functions inherited from jengine

- **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-line) 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::l PROLOG** ()

*Returns the top of J garbage collection stack.*

- **jarray::l EPILOG** ( jarray::l oldtop)

*Frees all memory, allocated since ttop was recorded.*

- **void \* EPILOG** ( jarray::l oldtop, void \*hdr)

*Frees all memory, allocated since ttop was recorded.*

### Protected Member Functions

- **virtual void libInit** ()

*Called by the constructor before parsing the J+ script.*

### Protected Member Functions inherited from jengine

- **void \* GA** ( const jarray::l t, const jarray::l n, const jarray::l r, const jarray::l \* s)

*Allocates J array.*

- **void FR** ( void \*hdr)

*Frees J array memory.*

### Additional Inherited Members

### Public Types inherited from jengine

- **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.*

### Static Public Member Functions inherited from jengine

- **static void initJlibrary** (std::ostream &)

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

## Static Public Attributes inherited from `jengine`

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

### 4.5.1 Detailed Description

A J+ script.

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 `jplus()`

```
jplus::jplus ( )
```

initialize J+ engine with no script

#### 4.5.2.2 `~jplus()`

```
jplus::~~jplus ( )
```

Free allocated memory.

### 4.5.3 Member Function Documentation

#### 4.5.3.1 `get()`

```
jarray jplus::get (
    std::string name )
```

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

##### Parameters

<i>name</i>	name of variable to compute.
-------------	------------------------------

##### Returns

the computed variable value.

#### 4.5.3.2 `getProgram()` [1/2]

```
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 getProgram() [2/2]**

```
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**

<i>vars</i>	set of variable names to pull.
-------------	--------------------------------

**Returns**

vector of J sentences for computing said variables.

**4.5.3.4 init()**

```
bool jplus::init (
    std::istream & script )
```

Load and pre-parse J+ script.

**Parameters**

<i>script</i>	stream, containing the script to load.
---------------	--

**Returns**

"true" upon success.

**4.5.3.5 libInit()**

```
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. 63).

**4.5.3.6 set()**

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

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

**Parameters**

<i>name</i>	name of the variable.
<i>data</i>	array for the variable to refer to.

**Returns**

"true" upon success.

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

- `src/jplus.h`

## 4.6 jarray::MS Struct Reference

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

```
#include <src/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 a

```
I* jarray::MS::a
```

#### 4.6.2.2 j

```
S jarray::MS::j
```

#### 4.6.2.3 mflag

```
C jarray::MS::mflag
```

## 4.6.2.4 unused

```
C jarray::MS::unused
```

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

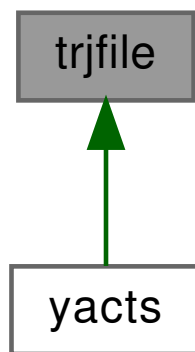
- src/ **jarray.h**

## 4.7 trjfile Class Reference

YACTS trajectory file.

```
#include <src/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::trjfile ( )
```

Instantiates unopened trajectory file.

## 4.7.3 Member Function Documentation

### 4.7.3.1 close()

```
bool trjfile::close ( )
```

Closes the trajectory.

### 4.7.3.2 frame\_size()

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

frame size on disk

#### 4.7.3.3 getNEQ()

```
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 good()

```
bool trjfile::good ( )
```

Checks that trajectory file is opened and properly positioned.

##### Returns

true if everything is ok.

#### 4.7.3.5 hasNextFrame()

```
bool trjfile::hasNextFrame ( )
```

Checks if the file has next frame stored.

##### Returns

true if the next frame can be read.

#### 4.7.3.6 header\_size()

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

header size on disk

#### 4.7.3.7 loadFrame()

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

Load frame.

##### Parameters

<i>T</i>	the variable to hold the time of the frame.
<i>data</i>	the array to hold the frame data (should have enough space for <b>getNEQ()</b> (p. 54) doubles).

#### 4.7.3.8 open()

```
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

<i>path</i>	filesystem path to the trajectory.
<i>create</i>	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 saveFrame()

```
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

<i>neq</i>	number of equations in the system.
<i>T</i>	time of the frame.
<i>data</i>	frame data.

##### Returns

true if success.

#### 4.7.3.10 setHeader()

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

patch the header with new data



## Parameters

<i>id</i>	reserved.
<i>neq</i>	number of equations in the system.

**4.7.3.11 size()**

```
int trjfile::size ( )
```

Get number of frames in the file.

## Returns

the total number of frames currently in trajectory.

**4.7.3.12 toFrame()**

```
bool trjfile::toFrame (
    int frame )
```

Position the file before the specified frame.

## Parameters

<i>frame</i>	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 neq**

```
int trjfile::neq [protected]
```

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

**4.7.4.2 trj**

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

Opened stream to the underlying file.

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

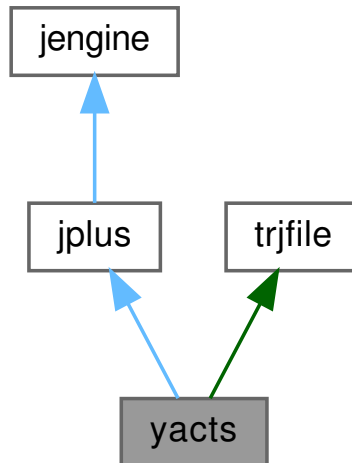
- src/ **trjfile.h**

## 4.8 yacts Class Reference

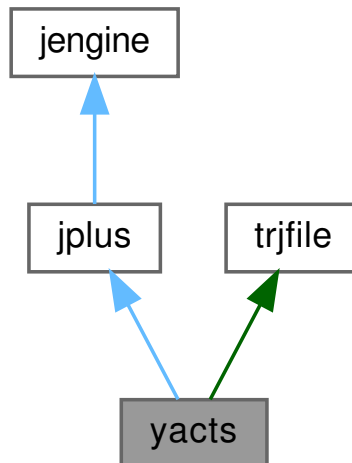
YACTS – yet another continuous time simulator.

```
#include <src/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 available 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.*

## Public Member Functions inherited from **jplus**

- **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 compute 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.*

## Public Member Functions inherited from **jengine**

- **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-line) 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::l PROLOG ()**  
*Returns the top of J garbage collection stack.*
- **jarray::l EPILOG ( jarray::l oldtop)**  
*Frees all memory, allocated since ttop was recorded.*
- **void \* EPILOG ( jarray::l oldtop, void \*hdr)**  
*Frees all memory, allocated since ttop was recorded.*

## Friends

- **int runTests ()**

## Additional Inherited Members

### Public Types inherited from jengine

- **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.*

### Static Public Member Functions inherited from jengine

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

### Static Public Attributes inherited from jengine

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

## Protected Member Functions inherited from `jengine`

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

## Protected Member Functions inherited from `trjfile`

- 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*
- **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 Attributes inherited from `trjfile`

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

### 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::yacts ( )
```

Initializes YACTS.

### 4.8.2.2 ~yacts()

```
yacts::~~yacts ( )
```

Frees YACTS memory.

## 4.8.3 Member Function Documentation

### 4.8.3.1 getTrajectoryFilenameBase()

```
std::string yacts::getTrajectoryFilenameBase (
    std::string prefix )
```

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

#### Parameters

<i>prefix</i>	trajectory filename prefix.
---------------	-----------------------------

#### Returns

trajectory filename base (without extension).

### 4.8.3.2 hasNextFrameStored()

```
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 init()

```
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

<i>script</i>	stream to load YACTS script from.
---------------	-----------------------------------

#### 4.8.3.4 initTrajectory()

```
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

<i>prefix</i>	trajectory filename prefix.
---------------	-----------------------------

## Returns

true upon success.

#### 4.8.3.5 libInit()

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

Initializes YACTS library functions and makes them available to J.

Reimplemented from **jplus** (p. 51).

#### 4.8.3.6 nextFrame()

```
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 process()

```
std::string yacts::process ( )
```

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

## Returns

the computed output.

#### 4.8.3.8 REPL()

```
int yacts::REPL ( )
```

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

Returns

0.

#### 4.8.3.9 setFrame()

```
bool yacts::setFrame (
    int iframe )
```

Sets specified frame as "current", loads it.

Parameters

<i>iframe</i>	index of the frame to become current.
---------------	---------------------------------------

Returns

"true" upon success.

#### 4.8.3.10 setOut()

```
bool yacts::setOut (
    std::string newOut )
```

Sets the name of output variable.

Parameters

<i>newOut</i>	new output variable name.
---------------	---------------------------

Returns

"true" if success (the name was OK)

#### 4.8.3.11 size()

```
int yacts::size ( )
```

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

Returns

number of frames in the trajectory



## 4.8.4 Friends And Related Symbol Documentation

### 4.8.4.1 runTests

```
int runTests ( ) [friend]
```

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

- src/ **yacts.h**

## 4.9 jarray::Z Struct Reference

complex type, equivalent to J

```
#include <src/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 im

```
D jarray::Z::im
```

imaginary part

#### 4.9.2.2 re

```
D jarray::Z::re
```

real part

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

- src/ **jarray.h**



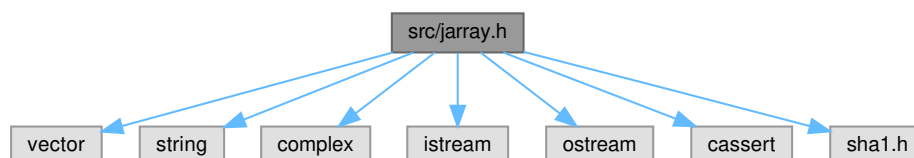
## Chapter 5

# File Documentation

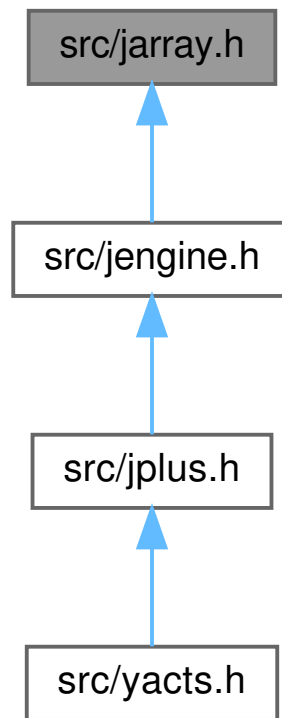
### 5.1 src/jarray.h File Reference

```
#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 operator<<()

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

prints ASCII representation of the array

## Parameters

<i>stream</i>	stream to print the array to.
<i>array</i>	the array to print.

## Returns

the reference to stream.

## 5.2 jarray.h

### Go to the documentation of this file.

```

00001 /* -*- mode: c++; indent-tabs-mode:nil -*- */
00002 #ifndef JARRAY_H
00003 #define JARRAY_H
00004
00005 #include <vector>
00006 #include <string>
00007 #include <complex>
00008 #include <istream>
00009 #include <ostream>
00010 #include <cassert>
00011 #include "shal.h"
00012
00013 class jengine;
00014
00025 class jarray {
00026 public:
00027     // C types, corresponding to J types
00028     // TODO: these are only for 32 bit linux, need ifdefs to cover
00029     // additional platforms
00030     typedef char B;
00031     typedef char C;
00032     typedef short S;
00033     #if _UNIX64
00034     typedef long long I;
00035     #else
00036     typedef long I;
00037     #endif
00038     typedef double D;
00040     typedef struct {
00041         D re,
00042         im;
00043     } Z;
00044
00045     friend class jengine;
00046
00047 public:
00049     enum errorType {
00050         ERR_CONV=10,
00051         ERR_SHAPE=11
00052     };
00053
00055     enum elementType {
00056         T_B01 =1,
00057         T_LIT =2,
00058         T_INT =4,
00059         T_FL =8,
00060         T_CMPX =16
00061     };
00062     // numeric constants, corresponding to J 7.01 types.
00063     // Not all of J types are supported, but the supported ones have
00064     // the full set of automatic widening conversions.
00065     //const static I T_B01 =1;          /**< B boolean          */
00066     //const static I T_LIT =2;          /**< C literal (character) */
00067     //const static I T_INT =4;          /**< I integer            */
00068     //const static I T_FL =8;           /**< D double (IEEE floating point) */
00069     //const static I T_CMPX =16;        /**< Z complex            */
00070     //const static I T_BOX =32;         /**< A boxed              */
00071     //const static I T_XNUM =64;        /**< X extended precision integer */
00072     //const static I T_RAT =128;        /**< Q rational number      */
00073     //const static I T_SB01 =1024;      /**< P sparse boolean      */
00074     //const static I T_SLIT =2048;      /**< P sparse literal (character) */
00075     //const static I T_SINT =4096;      /**< P sparse integer       */
00076     //const static I T_SFL =8192;       /**< P sparse floating point */
00077     //const static I T_SCMPX=16384;     /**< P sparse complex       */

```

```

00078 //const static I T_SBOX =32768;          /**< P sparse boxed          */
00079 //const static I T_SBT  =65536;          /**< SB symbol          */
00080 //const static I T_C2T  =131072;         /**< C2 unicode (2-byte characters) */
00081
00082 protected:
00083     typedef struct {
00084         I offset,
00085         flag,
00086         maxbytes,
00087         type,
00088         refcnt,
00089         n,
00090         rank,
00091         shape[1];
00092     } header;
00093     header *hdr;
00094     jengine *je;
00095     typedef struct {I*a;S j;C mflag,unused;} MS;
00096
00097     bool allocate(jengine *je_, elementType type, const I rank, const I* shape);
00098
00099     // conversions: T=bool
00100     inline static int fromB01(bool& v, const B d) {v=(d!=0); return 0;};
00101     inline static int fromLIT(bool& v, const C d) {return ERR_CONV;};
00102     inline static int fromINT(bool& v, const I d) {return ERR_CONV;};
00103     inline static int fromFL(bool& v, const D d) {return ERR_CONV;};
00104     inline static int fromCMPX(bool& v, const Z d) {return ERR_CONV;};
00105     inline static int toB01(const bool v, B& d) {d=v; return 0;};
00106     inline static int toLIT(const bool v, C& d) {return ERR_CONV;};
00107     inline static int toINT(const bool v, I& d) {d=v; return 0;};
00108     inline static int toFL(const bool v, D& d) {d=v; return 0;};
00109     inline static int toCMPX(const bool v, Z& d) {d.re=v; d.im=0; return 0;};
00110
00111     // conversions: T=char
00112     // it may be more convenient to obtain literal array as a string
00113     // via get(string&) method.
00114     inline static int fromB01(char& v, const B d) {return ERR_CONV;};
00115     inline static int fromLIT(char& v, const C d) {v=d; return 0;};
00116     inline static int fromINT(char& v, const I d) {return ERR_CONV;};
00117     inline static int fromFL(char& v, const D d) {return ERR_CONV;};
00118     inline static int fromCMPX(char& v, const Z d) {return ERR_CONV;};
00119     inline static int toB01(const char v, B& d) {return ERR_CONV;};
00120     inline static int toLIT(const char v, C& d) {d=v; return 0;};
00121     inline static int toINT(const char v, I& d) {d=v; return 0;};
00122     inline static int toFL(const char v, D& d) {d=v; return 0;};
00123     inline static int toCMPX(const char v, Z& d) {d.re=v; d.im=0; return 0;};
00124
00125     // conversions: T=int
00126     inline static int fromB01(int& v, const B d) {v=d; return 0;};
00127     inline static int fromLIT(int& v, const C d) {v=d; return 0;};
00128     inline static int fromINT(int& v, const I d) {v=d; return 0;};
00129     inline static int fromFL(int& v, const D d) {return ERR_CONV;};
00130     inline static int fromCMPX(int& v, const Z d) {return ERR_CONV;};
00131     inline static int toB01(const int v, B& d) {return ERR_CONV;};
00132     inline static int toLIT(const int v, C& d) {return ERR_CONV;};
00133     inline static int toINT(const int v, I& d) {d=v; return 0;};
00134     inline static int toFL(const int v, D& d) {d=v; return 0;};
00135     inline static int toCMPX(const int v, Z& d) {d.re=v; d.im=0; return 0;};
00136
00137     // conversions: T=long
00138     inline static int fromB01(long& v, const B d) {v=d; return 0;};
00139     inline static int fromLIT(long& v, const C d) {v=d; return 0;};
00140     inline static int fromINT(long& v, const I d) {v=d; return 0;};
00141     inline static int fromFL(long& v, const D d) {return ERR_CONV;};
00142     inline static int fromCMPX(long& v, const Z d) {return ERR_CONV;};
00143     inline static int toB01(const long v, B& d) {return ERR_CONV;};
00144     inline static int toLIT(const long v, C& d) {return ERR_CONV;};
00145     inline static int toINT(const long v, I& d) {d=v; return 0;};
00146     inline static int toFL(const long v, D& d) {d=v; return 0;};
00147     inline static int toCMPX(const long v, Z& d) {d.re=v; d.im=0; return 0;};
00148
00149     // conversions: T=long long
00150     inline static int fromB01(long long& v, const B d) {v=d; return 0;};
00151     inline static int fromLIT(long long& v, const C d) {v=d; return 0;};
00152     inline static int fromINT(long long& v, const I d) {v=d; return 0;};
00153     inline static int fromFL(long long& v, const D d) {return ERR_CONV;};
00154     inline static int fromCMPX(long long& v, const Z d) {return ERR_CONV;};
00155     inline static int toB01(const long long v, B& d) {return ERR_CONV;};
00156     inline static int toLIT(const long long v, C& d) {return ERR_CONV;};
00157     inline static int toINT(const long long v, I& d) {d=v; return 0;};
00158     inline static int toFL(const long long v, D& d) {d=v; return 0;};
00159     inline static int toCMPX(const long long v, Z& d) {d.re=v; d.im=0; return 0;};
00160
00161     // conversions: T=double
00162     inline static int fromB01(double& v, const B d) {v=d; return 0;};
00163     inline static int fromLIT(double& v, const C d) {v=d; return 0;};
00164     inline static int fromINT(double& v, const I d) {v=d; return 0;};

```

```

00179 inline static int fromFL(double& v, const D d) {v=d; return 0;};
00180 inline static int fromCMPX(double& v, const Z d) {return ERR_CONV;};
00181 inline static int toB01(const double v, B& d) {return ERR_CONV;};
00182 inline static int toLIT(const double v, C& d) {return ERR_CONV;};
00183 inline static int toINT(const double v, I& d) {d=v; return 0;};
00184 inline static int toFL(const double v, D& d) {d=v; return 0;};
00185 inline static int toCMPX(const double v, Z& d) {d.re=v; d.im=0; return 0;};
00186
00187 // conversions: T=complex<double>
00188 inline static int fromB01(std::complex<double>& v, const B d) {
00189     return ERR_CONV;};
00190 inline static int fromLIT(std::complex<double>& v, const C d) {
00191     v=d; return 0;};
00192 inline static int fromINT(std::complex<double>& v, const I d) {
00193     v=d; return 0;};
00194 inline static int fromFL(std::complex<double>& v, const D d) {
00195     v=d; return 0;};
00196 inline static int fromCMPX(std::complex<double>& v, const Z d) {
00197     std::complex<double> c(d.re, d.im); v=c; return 0;};
00198 inline static int toB01(const std::complex<double> v, B& d) {
00199     return ERR_CONV;};
00200 inline static int toLIT(const std::complex<double> v, C& d) {
00201     return ERR_CONV;};
00202 inline static int toINT(const std::complex<double> v, I& d) {
00203     return ERR_CONV;};
00204 inline static int toFL(const std::complex<double> v, D& d) {
00205     return ERR_CONV;};
00206 inline static int toCMPX(const std::complex<double> v, Z& d) {
00207     d.re=v.real(); d.im=v.imag(); return 0;};
00211 void grab() const;
00212
00214 void release();
00215
00216 // returns pointer to the header and (if parameter is "true", which is
00217 // default) marks the array as owned by J preventing its memory from
00218 // being freed as it goes out of scope.
00219 inline I getHeader(bool give_up_ownership=true) const {
00220     if (give_up_ownership) hdr->flag|=OWNED_BY_J;
00221     return (I)hdr;
00222 };
00223
00224 private:
00225 // flag, marking the array as J owned, preventing its memory from
00226 // being freed when the array goes out of scope.
00227 const static int OWNED_BY_J=131072;
00228 // marks the array as being allocated on heap without J assistance
00229 const static int OWNED_BY_JPLUS=262144;
00230
00231 public:
00232 I getRefCount() const {return hdr->refcnt;};
00233
00242 template<class T> inline int get(T& v, const int i) const {
00243     switch(hdr->type) {
00244         case T_B01: return fromB01(v, ((B*) ((B*)hdr)+hdr->offset))[i]);
00245         case T_LIT: return fromLIT(v, ((C*) ((B*)hdr)+hdr->offset))[i]);
00246         case T_INT: return fromINT(v, ((I*) ((B*)hdr)+hdr->offset))[i]);
00247         case T_FL: return fromFL(v, ((D*) ((B*)hdr)+hdr->offset))[i]);
00248         case T_CMPX: return fromCMPX(v, ((Z*) ((B*)hdr)+hdr->offset))[i]);
00249         default: return ERR_CONV;
00250     };
00251 };
00252
00258 template<class T> inline int set(const int i, const T v) {
00259     switch(hdr->type) {
00260         case T_B01: return toB01(v, ((B*) ((B*)hdr)+hdr->offset))[i]);
00261         case T_LIT: return toLIT(v, ((C*) ((B*)hdr)+hdr->offset))[i]);
00262         case T_INT: return toINT(v, ((I*) ((B*)hdr)+hdr->offset))[i]);
00263         case T_FL: return toFL(v, ((D*) ((B*)hdr)+hdr->offset))[i]);
00264         case T_CMPX: return toCMPX(v, ((Z*) ((B*)hdr)+hdr->offset))[i]);
00265         default: return ERR_CONV;
00266     };
00267 };
00268
00273 template<class T> int get(std::vector<T>& v) const {
00274     std::vector<T> r;
00275     r.reserve(hdr->n);
00276     int e;
00277     switch(hdr->type) {
00278         case T_B01: for(int i=0; i<hdr->n; i++)
00279             if (e=fromB01(v[i], ((B*) ((B*)hdr)+hdr->offset))[i])) return e;
00280             break;
00281         case T_LIT: for(int i=0; i<hdr->n; i++)
00282             if (e=fromLIT(v[i], ((C*) ((B*)hdr)+hdr->offset))[i])) return e;
00283             break;
00284         case T_INT: for(int i=0; i<hdr->n; i++)
00285             if (e=fromINT(v[i], ((I*) ((B*)hdr)+hdr->offset))[i])) return e;
00286             break;

```

```

00287     case T_FL: for(int i=0;i<hdr->n;i++)
00288         if (e=fromFL(v[i], ((D*) ((B*)hdr)+hdr->offset))[i])) return e;
00289         break;
00290     case T_CMPX: for(int i=0;i<hdr->n;i++)
00291         if (e=fromCMPX(v[i], ((Z*) ((B*)hdr)+hdr->offset))[i])) return e;
00292         break;
00293     default: return ERR_CONV;
00294     };
00295     r.swap(v);
00296     return 0;
00297 };
00298
00304 template<class T> int get(T& v);
00305
00307 jarray();
00308
00313 jarray(jengine *je_, std::istream& in);
00314
00318 static jarray::I esize(elementType type);
00319
00321 jarray::I esize() const;
00322
00325 void addhash(SHA1 &sha);
00326
00331 bool write(std::ostream& out);
00332
00335 inline bool isValid() const {
00336     return hdr!=NULL;
00337 };
00338
00341 inline const I type() const {
00342     return hdr->type;
00343 };
00344
00347 inline const I rank() const {
00348     return hdr->rank;
00349 };
00350
00353 inline I* shape() const {
00354     return &hdr->shape[0];
00355 };
00356
00360 int extent(int dimension) const {
00361     assert((dimension>=0)&&(dimension<rank()));
00362     return shape()[dimension];
00363 };
00364
00367 inline void shape(std::vector<I> &shape) const {
00368     std::vector<I> r(&hdr->shape[0], &hdr->shape[rank()]);
00369     r.swap(shape);
00370 };
00371
00374 inline const int size() const {
00375     int n=1;
00376     for (int i=0;i<rank();i++) n*=hdr->shape[i];
00377     return n;
00378 };
00379
00382 inline I* data() const {
00383     return (I*) ((B*)hdr)+hdr->offset;
00384 };
00385
00391 jengine* getEngine() const {
00392     return je;
00393 };
00394
00398 jarray(jengine *je_, void *hdr_);
00399
00405 jarray(jengine *je_, elementType type, I rank, I* shape);
00406
00411 jarray(jengine *je_, elementType type, const std::vector<I>& shape);
00412
00416 jarray(jengine *je_, const std::string &str);
00417
00420 jarray(const jarray &other);
00421
00426 jarray& assign(const jarray& other);
00427
00431 inline jarray& operator=(const jarray& other) {
00432     return assign(other);
00433 };
00434
00440 bool operator== ( const jarray& rhs ) const;
00441
00443 ~jarray();
00444 };
00445

```



```

00450 std::ostream& operator<< (std::ostream& stream, const jarray& array);
00451
00454 template<class T> class jarray_of_type: public jarray {
00455 private:
00456     static elementType jtype(I dummy) {return T_INT;};
00457     static elementType jtype(D dummy) {return T_FL;};
00458     static elementType jtype(Z dummy) {return T_CMPX;};
00459     static elementType jtype(std::complex<D> dummy) {return T_CMPX;};
00460
00461 public:
00470     jarray_of_type(const jarray &ja):jarray() {
00471         T dummy;
00472         elementType type=jtype(dummy);
00473         if (type!=ja.type()) { // make an independent copy and convert type
00474             allocate(ja.getEngine(), type, ja.rank(), ja.shape());
00475             int n=ja.size();
00476             for (int i=0;i<n;i++) {
00477                 T v;
00478                 ja.get(v,i);
00479                 set(i,v);
00480             };
00481         } else assign(ja); // type is the same, direct access
00482     };
00483
00487     T& operator[](const I i) {
00488         // assert((i>=0)&&(i<size()));
00489         return ((T*)data())[i];
00490     };
00491
00495     const T& operator[](const I i) const {
00496         // assert((i>=0)&&(i<size()));
00497         return ((T*)data())[i];
00498     };
00499
00502     jarray_of_type(jengine *je_) {
00503         T dummy; I shape[]={}; allocate(je_, jtype(dummy), 0, shape);
00504     };
00505
00509     jarray_of_type(jengine *je_, const I l0) {
00510         T dummy; I shape[]={l0}; allocate(je_, jtype(dummy), 1, shape);
00511     };
00515     T& operator()(const I i0) {
00516         assert(rank()==1); return ((T*)data())[i0];
00517     };
00521     const T& operator()(const I i0) const {
00522         assert(rank()==1); return ((T*)data())[i0];
00523     };
00524
00529     jarray_of_type(jengine *je_, const I l0, const I l1) {
00530         T dummy; I shape[]={l0,l1}; allocate(je_, jtype(dummy), 2, shape);
00531     };
00532
00537     T& operator()(const I i0,const I i1) {
00538         assert(rank()==2); return ((T*)data())[i0*extent(1)+i1];
00539     };
00544     const T& operator()(const I i0,const I i1) const {
00545         assert(rank()==2); return ((T*)data())[i0*extent(1)+i1];
00546     };
00547
00553     jarray_of_type(jengine *je_, const I l0, const I l1, const I l2) {
00554         T dummy; I shape[]={l0,l1,l2}; allocate(je_, jtype(dummy), 3, shape);
00555     };
00556
00562     T& operator()(const I i0,const I i1,const I i2) {
00563         assert(rank()==3); return ((T*)data())[i0*extent(1)+i1*extent(2)+i2];
00564     };
00565
00571     const T& operator()(const I i0,const I i1,const I i2) const {
00572         assert(rank()==3); return ((T*)data())[i0*extent(1)+i1*extent(2)+i2];
00573     };
00574
00581     jarray_of_type(jengine *je_, const I l0, const I l1, const I l2,
00582                   const I l3) {
00583         T dummy; I shape[]={l0,l1,l2,l3}; allocate(je_, jtype(dummy), 4, shape);
00584     };
00585
00592     T& operator()(const I i0,const I i1,const I i2,const I i3) {
00593         assert(rank()==4);
00594         return ((T*)data())[((i0*extent(1)+i1)*extent(2)+i2)*extent(3)+i3];
00595     };
00596
00603     const T& operator()(const I i0,const I i1,const I i2,const I i3) const {
00604         assert(rank()==4);
00605         return ((T*)data())[((i0*extent(1)+i1)*extent(2)+i2)*extent(3)+i3];
00606     };
00607
00615     jarray_of_type(jengine *je_, const I l0, const I l1, const I l2,

```

```

00616         const I i3, const I i4) {
00617     T dummy; I shape[]={10,11,12,13,14}; allocate(je_, jtype(dummy), 5, shape);
00618 };
00619
00627 T& operator()(const I i0,const I i1,const I i2,const I i3,const I i4) {
00628     assert(rank()==4);
00629     return ((T*)data()) [ ((i0*extent(1)+i1)*extent(2)+i2)*extent(3)+i3)*extent(4)+i4];
00630 };
00631
00639 const T& operator()(const I i0,const I i1,const I i2,const I i3,const I i4) const {
00640     assert(rank()==4);
00641     return ((T*)data()) [ ((i0*extent(1)+i1)*extent(2)+i2)*extent(3)+i3)*extent(4)+i4];
00642 };
00643
00644
00649 jarray_of_type(jengine *je_, const I rank, const I *shape) {
00650     T dummy; allocate(je_, jtype(dummy), rank, shape);
00651 };
00652
00656 jarray_of_type(jengine *je_, const std::vector<I> &shape) {
00657     T dummy; allocate(je_, jtype(dummy), shape.size(), &shape[0]);
00658 };
00659
00663 T& operator()(const std::vector<I> &subscripts) {
00664     assert(rank()==subscripts.size());
00665     I pos=0;
00666     for(int i=0;i<subscripts.size();i++) pos=pos*extent(i)+subscripts[i];
00667     return ((T*)data())[pos];
00668 };
00672 const T& operator()(const std::vector<I> &subscripts) const {
00673     assert(rank()==subscripts.size());
00674     I pos=0;
00675     for(int i=0;i<subscripts.size();i++) pos=pos*extent(i)+subscripts[i];
00676     return ((T*)data())[pos];
00677 };
00682 T& operator()(const I *subscripts) {
00683     I pos=0;
00684     for(int i=0;i<rank();i++) pos=pos*extent(i)+subscripts[i];
00685     return ((T*)data())[pos];
00686 };
00691 const T& operator()(const I *subscripts) const {
00692     I pos=0;
00693     for(int i=0;i<rank();i++) pos=pos*extent(i)+subscripts[i];
00694     return ((T*)data())[pos];
00695 };
00696
00697 };
00698 #endif

```

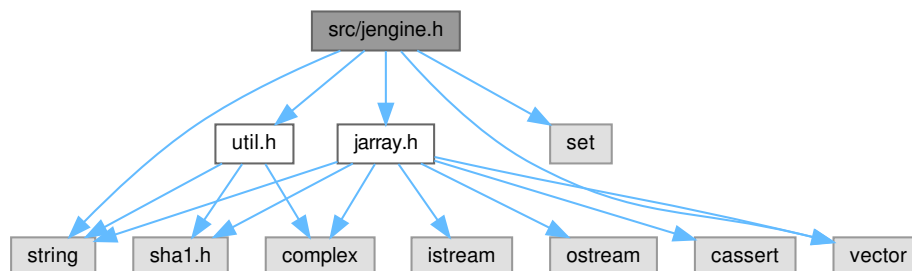
### 5.3 src/jengine.h File Reference

```

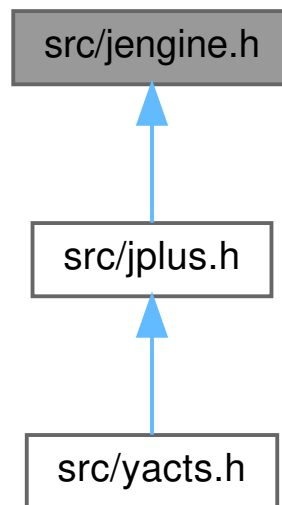
#include <string>
#include <vector>
#include <set>
#include "util.h"
#include "jarray.h"

```

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.*

## 5.4 jengine.h

**Go to the documentation of this file.**

```

00001 /* -*- mode: c++; indent-tabs-mode:nil -*- */
00002 #ifndef JENGINE_H
00003 #define JENGINE_H
00004
00005 #include <string>
00006 #include <vector>
00007 #include <set>
00008
00009 #include "util.h"
00010 #include "jarray.h"
00011
00012 class jengine {
00013 public:
00014     friend class jarray;
00015     typedef jarray (*monad) (jarray y);
00016     typedef jarray (*dyad) (jarray x, jarray y);
00017     typedef jarray (*amonad) (monad um, dyad ud, jarray y);
00018     typedef jarray (*adyad) (monad um, dyad ud, jarray x, jarray y);
00019
00020     jengine();
00021     ~jengine();
00022
00023     static void initJlibrary(std::ostream &);
00024
00025     bool doJ(const std::string s);
00026
00027     const jarray get(const std::string name);
00028
00029     bool set(const std::string name, jarray &value);
00030
00031     int getError();
00032
00033     inline bool ok() {
00034         return (getError()==0);
00035     };
00036
00037     const static int RMAX=10000;
00038
00039     bool defVerb(std::string name, monad mf, dyad df, int mr=RMAX, int lr=RMAX, int rr=RMAX);

```

```

00074
00083     bool defAdverb(std::string name, amonad mf, adyad df);
00084
00093     bool defScript(std::string name, int type, std::string code,
00094                   int mr=jengine::RMAX,
00095                   int lr=jengine::RMAX,
00096                   int rr=jengine::RMAX);
00097
00101     bool isBuiltin(std::string name) const;
00102
00105     std::set<std::string> getBuiltins() const;
00106
00109     jarray::I PROLOG();
00110
00119     jarray::I EPILOG(jarray::I oldtop);
00120
00133     void* EPILOG(jarray::I oldtop, void *hdr);
00134
00135 protected:
00142     void* GA(const jarray::I t, const jarray::I n, const jarray::I r, const jarray::I *s);
00146     void FR(void *hdr);
00147
00148 private:
00149     // copy disabled
00150     jengine(const jengine &);
00151     const jengine& operator=(const jengine&);
00152
00153     class Impl;
00154     Impl* pImpl;
00155 };
00156 #endif

```

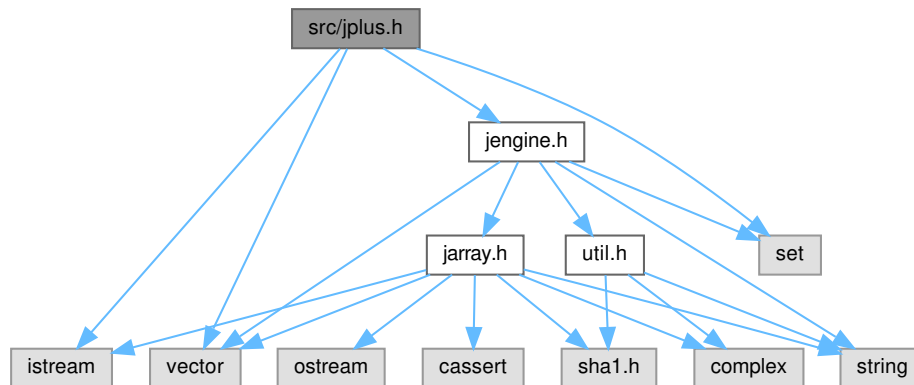
## 5.5 src/jplus.h File Reference

```

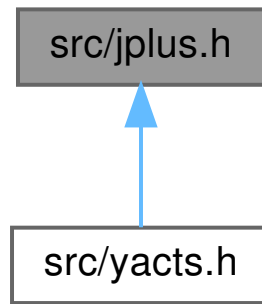
#include <istream>
#include <vector>
#include <set>
#include "jengine.h"

```

Include dependency graph for jplus.h:



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



## Data Structures

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

## 5.6 jplus.h

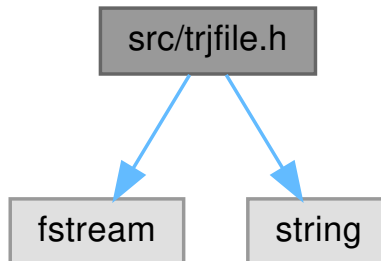
Go to the documentation of this file.

```

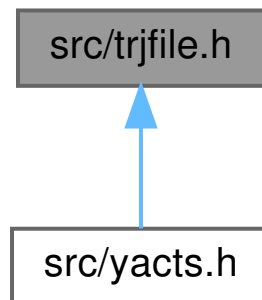
00001 /* -*- mode: c++; indent-tabs-mode:nil -*- */
00002 #ifndef JPLUS_H
00003 #define JPLUS_H
00004
00005 #include <istream>
00006 #include <vector>
00007 #include <set>
00008 #include "jengine.h"
00009
00010 class BasicBlock;
00011
00015 class jplus: public jengine {
00016 private:
00017     BasicBlock* program;
00018
00019 public:
00021     jplus();
00022
00026     bool init(std::istream &script);
00027
00029     ~jplus();
00030
00036     bool set(std::string name, jarray data);
00037
00042     jarray get(std::string name);
00043
00049     std::vector<std::string> getProgram(std::set<std::string> vars);
00050
00054     std::vector<std::string> getProgram();
00055
00056 protected:
00061     virtual void libInit();
00062
00063 private:
00064     // copy disabled
00065     jplus(const jplus &);
00066     const jplus& operator=(const jplus&);
00067 };
00068
00069 #endif
  
```

## 5.7 src/trjfile.h File Reference

```
#include <fstream>
#include <string>
Include dependency graph for trjfile.h:
```



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



### Data Structures

- class **trjfile**  
*YACTS trajectory file.*

## 5.8 trjfile.h

**Go to the documentation of this file.**

```
00001 /* -*- mode: c++; indent-tabs-mode:nil -*- */
00002 #include <fstream>
00003 #include <string>
00004
00006 class trjfile {
00007 protected:
00009     int neq;
00011     std::fstream trj;
00012 public:
00013
00015     trjfile();
00016
00023     bool open(std::string path, bool create=true);
00024
00026     bool close();
00027
00030     bool good();
00031
00040     bool saveFrame(const int neq, const double &T, const double* data);
00041
```

```

00044     bool hasNextFrame();
00045
00049     int getNEQ();
00050
00055     bool loadFrame(double &T, double* data);
00056
00059     int size();
00060
00064     bool toFrame(int frame);
00065
00066 protected:
00068     std::streampos header_size();
00069
00071     std::streampos frame_size();
00072
00076     void setHeader(int id, int neq);
00077
00078 };

```

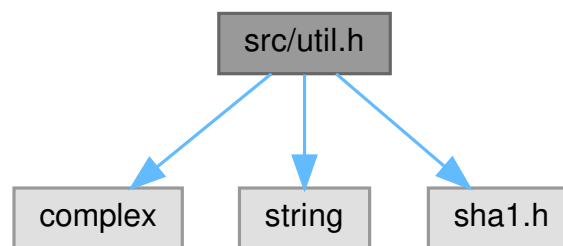
## 5.9 src/util.h File Reference

```

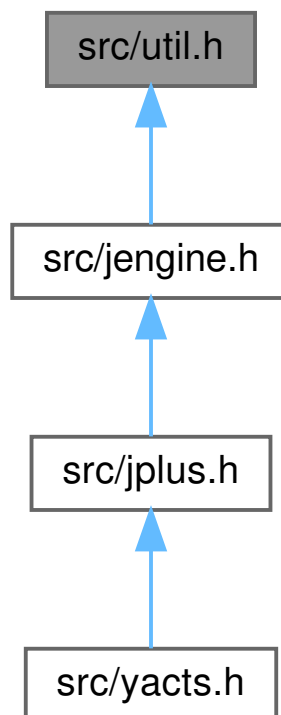
#include <complex>
#include <string>
#include "sha1.h"

```

Include dependency graph for util.h:



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



## Macros

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

## Typedefs

- `typedef double Real`
- `typedef std::complex< double > Cmplx`
- `typedef long double IReal`

## 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*

## 5.9.1 Macro Definition Documentation

### 5.9.1.1 \_stdcall

```
#define _stdcall
```

### 5.9.1.2 fftw

```
#define fftw(  
    name ) fftw_ ## name
```

### 5.9.1.3 TYPE\_STR

```
#define TYPE_STR ""
```

## 5.9.2 Typedef Documentation

### 5.9.2.1 Cmplx

```
typedef std::complex<double> Cmplx
```



### 5.9.2.2 IReal

```
typedef long double IReal
```

### 5.9.2.3 Real

```
typedef double Real
```

## 5.9.3 Function Documentation

### 5.9.3.1 parsedouble()

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

### 5.9.3.2 parseint()

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

### 5.9.3.3 parsepositivedouble()

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

### 5.9.3.4 parsepositiveint()

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

### 5.9.3.5 sha1tostring()

```
std::string shaltostring (
    SHA1 & sha1 )
```

convert sha1 to string

### 5.9.3.6 tol\_eq()

```
bool tol_eq (
    double a,
    double b )
```

tolerant comparison of two double numbers

## 5.10 util.h

Go to the documentation of this file.

```

00001 /* -*- mode: c++; indent-tabs-mode:nil -*- */
00002 #ifndef UTIL_H
00003 #define UTIL_H
00004
00005 #include <complex>
00006 #include <string>
00007 #include "sha1.h"
00008
00009 // the base type is "double" (64 bits on x86/x87)
00010 typedef double Real;
00011 typedef std::complex<double> Cmplx;
00012 typedef long double lReal;
00013 #define fftw(name) fftw_ ## name
00014 #define TYPE_STR ""
00015
00016 #ifndef _WIN32
00017 #define _stdcall
00018 #endif
00019
00020 // parses string as a positive integer, reporting errors
00021 // to stderr and returning 1 in case of failure, 0 if success.
00022 int parseint(char *str, int& res);
00023
00024 // additionally error is signalled if the number is <0
00025 int parsepositiveint(char *str, int& res);
00026
00027 // parses string as a positive double, reporting errors
00028 // to stderr and returning 1 in case of failure, 0 if success.
00029 int parsedouble(char *str, double& res);
00030
00031 // additionally error is signalled if the number is <0
00032 int parsepositivedouble(char *str, double& res);
00033
00035 std::string shaltostring(SHA1 &shal);
00036
00038 bool tol_eq(double a, double b);
00039
00040 #endif

```

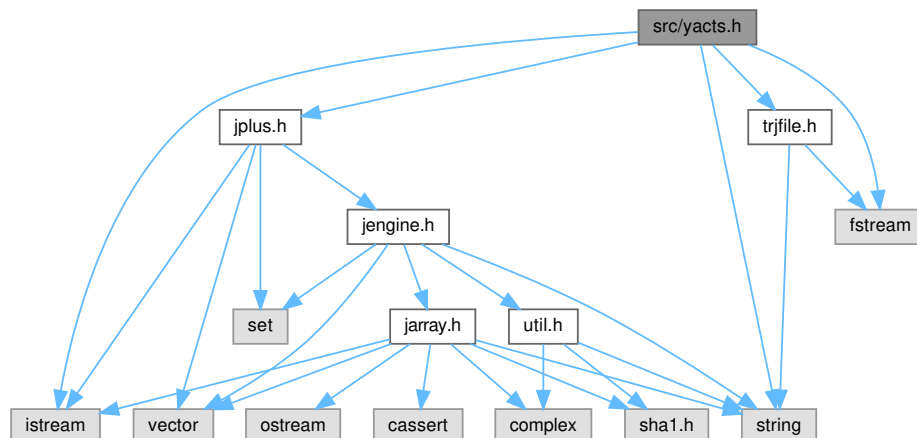
## 5.11 src/yacts.h File Reference

```

#include <istream>
#include <fstream>
#include <string>
#include "jplus.h"
#include "trjfile.h"

```

Include dependency graph for yacts.h:



## Data Structures

- class **yacts**

*YACTS – yet another continuous time simulator.*

## 5.12 yacts.h

**Go to the documentation of this file.**

```
00001 /* -*- mode: c++; indent-tabs-mode:nil -*- */
00002 #ifndef YACTS_H
00003 #define YACTS_H
00004
00005 #include <istream>
00006 #include <fstream>
00007 #include <string>
00008 #include "jplus.h"
00009 #include "trjfile.h"
00010
00015 class yacts : public jplus, protected trjfile {
00016 public:
00018     yacts();
00019
00028     bool init(std::istream &script);
00029
00031     ~yacts();
00032
00034     virtual void libInit();
00035
00039     bool setOut(std::string newOut);
00040
00043     int REPL();
00044
00050     std::string getTrajectoryFilenameBase(std::string prefix);
00051
00056     bool initTrajectory(std::string prefix);
00057
00060     int size();
00061
00065     bool setFrame(int iframe);
00066
00070     bool hasNextFrameStored();
00071
00075     bool nextFrame();
00076
00079     std::string process();
00080
00081     friend int runTests(); // testsuite is also our friend :-)
00082
00083 private:
00084     void *sunctxptr; // pointer to allocated SUNDIALS context
00085     void *sunLSptr; // pointer to allocated SUNDIALS linear solver
00086     void *cvode_mem; // memory, allocated by sundials CVODE solver
00087     void *nvs; // current state
00088     std::string outName; // out variable name
00089
00091     bool loadFrame();
00093     bool saveFrame();
00094
00095 };
00096
00097 #endif
```



# Index

- `_stdcall`
    - `util.h`, 80
  - `~jarray`
    - `jarray`, 16
  - `~jengine`
    - `jengine`, 41
  - `~jplus`
    - `jplus`, 50
  - `~yacts`
    - `yacts`, 62
- `a`
  - `jarray::MS`, 52
- `addhash`
  - `jarray`, 16
- `adyad`
  - `jengine`, 41
- `allocate`
  - `jarray`, 16
- `amonad`
  - `jengine`, 41
- `assign`
  - `jarray`, 16

**B**

- `jarray`, 12

**C**

- `jarray`, 12

- `close`
- `trjfile`, 54
- `Cmplx`
- `util.h`, 80

**D**

- `jarray`, 12

- `data`
- `jarray`, 17
- `defAdverb`
- `jengine`, 41
- `defScript`
- `jengine`, 42
- `defVerb`
- `jengine`, 42
- `doJ`
- `jengine`, 43
- `dyad`
- `jengine`, 41
- `elementType`
- `jarray`, 13

- `EPILOG`
  - `jengine`, 43
- `ERR_CONV`
  - `jarray`, 13
- `ERR_SHAPE`
  - `jarray`, 13
- `errorType`
  - `jarray`, 13
- `esize`
  - `jarray`, 17
- `extent`
  - `jarray`, 18

`fftw`

- `util.h`, 80

`flag`

- `jarray::header`, 8

`FR`

- `jengine`, 44

- `frame_size`
- `trjfile`, 54

**GA**

- `jengine`, 44

`get`

- `jarray`, 18, 19
- `jengine`, 45
- `jplus`, 50

- `getBultins`
- `jengine`, 45
- `getEngine`
- `jarray`, 19
- `getError`
- `jengine`, 45
- `getHeader`
- `jarray`, 20
- `getNEQ`
- `trjfile`, 54
- `getProgram`
- `jplus`, 50, 51
- `getRefCount`
- `jarray`, 20
- `getTrajectoryFilenameBase`
- `yacts`, 62
- `good`
- `trjfile`, 55
- `grab`
- `jarray`, 20

`hasNextFrame`

- trjfile, 55
- hasNextFrameStored
  - yacts, 62
- hdr
  - jarray, 24
- header\_size
  - trjfile, 55
- I
  - jarray, 13
- im
  - jarray::Z, 65
- init
  - jplus, 51
  - yacts, 62
- initJlibrary
  - jengine, 45
- initTrajectory
  - yacts, 63
- isBuiltin
  - jengine, 45
- isValid
  - jarray, 20
- j
  - jarray::MS, 52
- jarray, 9
  - ~jarray, 16
  - addhash, 16
  - allocate, 16
  - assign, 16
  - B, 12
  - C, 12
  - D, 12
  - data, 17
  - elementType, 13
  - ERR\_CONV, 13
  - ERR\_SHAPE, 13
  - errorType, 13
  - esize, 17
  - extent, 18
  - get, 18, 19
  - getEngine, 19
  - getHeader, 20
  - getRefCount, 20
  - grab, 20
  - hdr, 24
  - I, 13
  - isValid, 20
  - jarray, 14, 15
  - je, 24
  - jengine, 24, 47
  - operator=, 20
  - operator==, 21
  - rank, 21
  - release, 21
  - S, 13
  - set, 21
  - shape, 22
  - size, 23
  - T\_B01, 13
  - T\_CMPX, 13
  - T\_FL, 13
  - T\_INT, 13
  - T\_LIT, 13
  - type, 23
  - write, 23
- jarray.h
  - operator<<, 68
- jarray::header, 7
  - flag, 8
  - maxbytes, 8
  - n, 8
  - offset, 8
  - rank, 8
  - refcnt, 8
  - shape, 8
  - type, 9
- jarray::MS, 52
  - a, 52
  - j, 52
  - mflag, 52
  - unused, 52
- jarray::Z, 65
  - im, 65
  - re, 65
- jarray\_of\_type
  - jarray\_of\_type< T >, 28–31
- jarray\_of\_type< T >, 24
  - jarray\_of\_type, 28–31
  - operator(), 32–37
  - operator[], 38
- je
  - jarray, 24
- jengine, 39
  - ~jengine, 41
  - adyad, 41
  - amonad, 41
  - defAdverb, 41
  - defScript, 42
  - defVerb, 42
  - doJ, 43
  - dyad, 41
  - EPILOG, 43
  - FR, 44
  - GA, 44
  - get, 45
  - getBultins, 45
  - getError, 45
  - initJlibrary, 45
  - isBuiltin, 45
  - jarray, 24, 47
  - jengine, 41
  - monad, 41
  - ok, 46
  - PROLOG, 46
  - RMAX, 47

- set, 46
- jplus, 47
  - ~jplus, 50
  - get, 50
  - getProgram, 50, 51
  - init, 51
  - jplus, 50
  - libInit, 51
  - set, 51
- libInit
  - jplus, 51
  - yacts, 63
- loadFrame
  - trjfile, 55
- IRReal
  - util.h, 80
- maxbytes
  - jarray::header, 8
- mflag
  - jarray::MS, 52
- monad
  - jengine, 41
- n
  - jarray::header, 8
- neq
  - trjfile, 57
- nextFrame
  - yacts, 63
- offset
  - jarray::header, 8
- ok
  - jengine, 46
- open
  - trjfile, 56
- operator<<
  - jarray.h, 68
- operator()
  - jarray\_of\_type< T >, 32–37
- operator=
  - jarray, 20
- operator==
  - jarray, 21
- operator[]
  - jarray\_of\_type< T >, 38
- parsedouble
  - util.h, 81
- parseint
  - util.h, 81
- parsepositivedouble
  - util.h, 81
- parsepositiveint
  - util.h, 81
- process
  - yacts, 63
- PROLOG
  - jengine, 46
- rank
  - jarray, 21
  - jarray::header, 8
- re
  - jarray::Z, 65
- Real
  - util.h, 81
- refcnt
  - jarray::header, 8
- release
  - jarray, 21
- REPL
  - yacts, 63
- RMAX
  - jengine, 47
- runTests
  - yacts, 65
- S
  - jarray, 13
- saveFrame
  - trjfile, 56
- set
  - jarray, 21
  - jengine, 46
  - jplus, 51
- setFrame
  - yacts, 64
- setHeader
  - trjfile, 56
- setOut
  - yacts, 64
- sha1tostring
  - util.h, 81
- shape
  - jarray, 22
  - jarray::header, 8
- size
  - jarray, 23
  - trjfile, 57
  - yacts, 64
- src/jarray.h, 67, 69
- src/jengine.h, 74, 75
- src/jplus.h, 76, 77
- src/trjfile.h, 78
- src/util.h, 79, 82
- src/yacts.h, 82, 83
- T\_B01
  - jarray, 13
- T\_CMPX
  - jarray, 13
- T\_FL
  - jarray, 13
- T\_INT
  - jarray, 13

- T\_LIT
  - jarray, 13
- toFrame
  - trjfile, 57
- tol\_eq
  - util.h, 81
- trj
  - trjfile, 57
- trjfile, 53
  - close, 54
  - frame\_size, 54
  - getNEQ, 54
  - good, 55
  - hasNextFrame, 55
  - header\_size, 55
  - loadFrame, 55
  - neq, 57
  - open, 56
  - saveFrame, 56
  - setHeader, 56
  - size, 57
  - toFrame, 57
  - trj, 57
  - trjfile, 54
- type
  - jarray, 23
  - jarray::header, 9
- TYPE\_STR
  - util.h, 80
- unused
  - jarray::MS, 52
- util.h
  - \_stdcall, 80
  - Cmplx, 80
  - fftw, 80
  - IReal, 80
  - parsedouble, 81
  - parseint, 81
  - parsepositivedouble, 81
  - parsepositiveint, 81
  - Real, 81
  - sha1tostring, 81
  - tol\_eq, 81
  - TYPE\_STR, 80
- write
  - jarray, 23
- yacts, 58
  - ~yacts, 62
  - getTrajectoryFilenameBase, 62
  - hasNextFrameStored, 62
  - init, 62
  - initTrajectory, 63
  - libInit, 63
  - nextFrame, 63
  - process, 63
  - REPL, 63
  - runTests, 65
  - setFrame, 64
  - setOut, 64
  - size, 64
  - yacts, 62