# Design Space Toolbox

2.0.1

Generated by Doxygen 1.6.3

Fri Jul 22 15:46:25 2011

# Contents

# Chapter 1

# Todo List

**File DSIO.h** Add 'plug-in' for printf function.

Define standard input and output file formats.

**File DSStd.h** Add all previous functionality.

Add vertex enumeration functionality.

**Global printMembers(DSVariablePool ∗root, char ∗buffer, int position)** Create a wrapper function which creates the buffer, and initiates the position at 0.

# Chapter 2

# Module Index

## 2.1 Modules

Here is a list of all modules:

# Chapter 3

# Data Structure Index

## 3.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 5

# Module Documentation

## 5.1 Messages for DS Errors.

### Defines

- #define M_DS_NOFILE "File not found"

  *Message for no file found.*

- #define M_DS_NULL "NULL pointer"

  *Message for NULL pointer.*

- #define M_DS_NOFORMAT "Format not known"

  *Message for unknown format.*

- #define M_DS_EXISTS "Data already exists"

  *Message for data aleady existing.*

- #define M_DS_MALLOC "Memory alloc failed"

  *Message for failure to allocate data.*

- #define M_DS_NOT_IMPL "Functionality not implemented"

  *Message for a feature not yet implemented.*

- #define M_DS_MAT_NULL "Pointer to matrix is NULL"

  *Message for a NULL DSMatrix pointer.*

- #define M_DS_MAT_OUTOFBOUNDS "Row or column out of bounds"

  *Message for a row or column exceeding matrix bounds.*

- #define M_DS_MAT_NOINTERNAL "Matrix data is empty"

  *Message for a NULL internal matrix structure.*

### 5.1.1 Detailed Description

Defined here are the generic messages used to report the appropriate errors. These are used with the different actions in the macro DS_ERROR. Other messages can be reported by literally writting them in instead of these messages in the DSError macro. Also, these messages can be modified by appending a literal string in the DSError macro.

**See also**

Actions for DS Errors.
DSError

Messages for DSMatrix related errors are M_DS_MAT_NULL, M_DS_MAT_OUTOFBOUNDS and M_-DS_MAT_NOINTERNAL.

## 5.2 Actions for DS Errors.

### Defines

- #define A_DS_NOERROR 0

  *Value for no error.*

- #define A_DS_WARN -1

  *Value for a warning.*

- #define A_DS_ERROR -2

  *Value for an error.*

- #define A_DS_FATAL -3

  *Value for a fatal error, kills program.*

### 5.2.1 Detailed Description

Defined here are the appropriate reactions to a specific error, an error can have different actions depending on the sensitivity of the region involved.

**See also**

Messages for DS Errors.
DS_ERROR

## 5.3 Macros to manipulate variables.

### Defines

- #define DSVariableAssignValue(x, y) DSVariableSetValue(x, y)

  *Macro to directly change the value of a variable.*

- #define DSVariableReturnValue(x) DSVariableValue(x)

  *Macro to directly access the value of a variable.*

### 5.3.1 Detailed Description

The following macros are in place for portability and consistency. As the structure of the BSTVariable is subject to change, due to the nature of early versions of the framework, using these macros will make the dependent code less subject to errors.

### 5.3.2 Define Documentation

#### 5.3.2.1 #define DSVariableAssignValue(x, y) DSVariableSetValue(x, y)

Macro to directly change the value of a variable.

Macro for manipulating value of a DSVariable. As a measure to make the manipulation of a variable independent of the structure, this macro provides a consistent and portable way of changing a variable value. Direct access of the variable value should be avoided.

**See also**

DSVariable
DSVariableReturnValue

#### 5.3.2.2 #define DSVariableReturnValue(x) DSVariableValue(x)

Macro to directly access the value of a variable.

Macro for retrieving the value of a DSVariable. As a measure to make the manipulation of a variable independent of the structure, this macro provides a consistent and portable way of retrieving the value of a variable. Direct access of the variable value should be avoided.

**See also**

DSVariable
DSVariableAssignValue

# Chapter 6

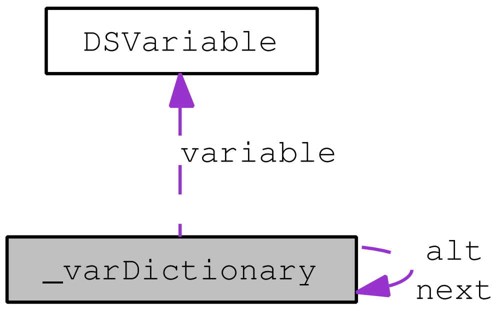
# Data Structure Documentation

## 6.1 _varDictionary Struct Reference

Internal dictionary structure.

`#include <DSTypes.h>`

Collaboration diagram for _varDictionary:



### Data Fields

- char current

  *The current character in the dictionary.*

- struct _varDictionary ∗ alt

  *The alternative character in the dictionary.*

- struct _varDictionary ∗ next

  *The next character in the dictionary.*

- DSVariable ∗ variable

  *The variable stored. Only when current is '\0'.*

### 6.1.1 Detailed Description

Internal dictionary structure. Internal dictionary for fast variable querying. The structure of the dictionary uses an alternative path, where each character is checked in order at each position, if there is a match, the next position is consequently checked. The dictionary should never be manipulated manually, adding, retrieving and removing variables should be done through the accesory functions.

**See also**

DSVariable

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

- DSTypes.h

## 6.2 DSMatrix Struct Reference

Data type representing a matrix.

```
#include <DSTypes.h>
```

### Data Fields

- void ∗ mat

  *The pointer to the internal representation of the matrix.*

- DSUInteger rows

  *A DSUInteger specifying the number of rows in the matrix.*

- DSUInteger columns

  *A DSUInteger specifying the number of columns in the matrix.*

### 6.2.1 Detailed Description

Data type representing a matrix. This data type is the front end of the matric manipulation portion of the design space toolbox. Currently, the DST library uses the gsl library; however, it is designed to be used with different back-ends. In particular, the CLAPACK package should be considered, as it will offer better performance. Thus, the matrix API should be independent of implementation, and hence a new matrix library could be used if chosen.

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

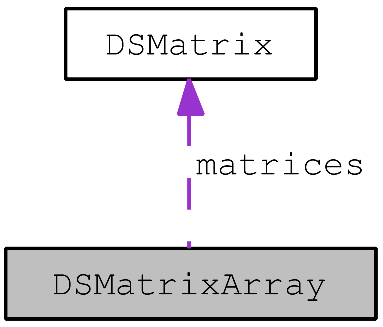
- DSTypes.h

## 6.3 DSMatrixArray Struct Reference

Data type representing an array of matrices.

```
#include <DSTypes.h>
```

Collaboration diagram for DSMatrixArray:



### Data Fields

- DSUInteger numberOfMatrices

  *A DSUInteger specifying the number of matrices in the array.*

- DSMatrix ∗∗ matrices

  *A pointer the the C-style array of matrices.*

### 6.3.1 Detailed Description

Data type representing an array of matrices. This data type is a utility data type that keeps track of arrays of matrices. This structure can also be used to represent three-dimensional matrices, as used in the internal representation of GMA's.

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

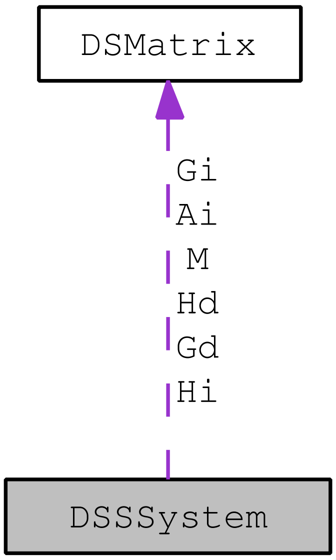
- DSTypes.h

## 6.4 DSSSystem Struct Reference

Data type representing an S-System.

```
#include <DSTypes.h>
```

Collaboration diagram for DSSSystem:



### Data Fields

- DSMatrix ∗ **Gd**
- DSMatrix ∗ **Gi**
- DSMatrix ∗ **Hd**
- DSMatrix ∗ **Hi**
- DSMatrix ∗ **M**
- DSMatrix ∗ **Ai**

### 6.4.1 Detailed Description

Data type representing an S-System. This data structure is a standard representation of an S-System using matrix notation. Here, the positive and negative terms are explicitly represented according to the Gs and Hs. Also, matrices are split up relating to either dependent and independent parameters.

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

- DSTypes.h

## 6.5 DSVariable Struct Reference

Basic variable structure containing name, value and NSString with special unicode characters for greek letters.

```
#include <DSTypes.h>
```

### Data Fields

- char ∗ name

    *Dynamically allocated name of the variable.*

- double value

    *Value of the variable.*

- DSUInteger retainCount

    *Retain counter for memory management.*

### 6.5.1 Detailed Description

Basic variable structure containing name, value and NSString with special unicode characters for greek letters. Structure that carries variable information. Internal to BSTVariables class and should not be created and/or freed manually and beyond the context of the BSTVariables class.

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

- DSTypes.h

## 6.6 yy_buffer_state Struct Reference

### Data Fields

- FILE ∗ **yy_input_file**
- char ∗ **yy_ch_buf**
- char ∗ **yy_buf_pos**
- yy_size_t **yy_buf_size**
- yy_size_t **yy_n_chars**
- int **yy_is_our_buffer**
- int **yy_is_interactive**
- int **yy_at_bol**
- int yy_bs_lineno
- int yy_bs_column
- int **yy_fill_buffer**
- int **yy_buffer_status**

### 6.6.1 Field Documentation

#### 6.6.1.1 int yy_bs_column

The column count.

#### 6.6.1.2 int yy_bs_lineno

The line count.

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

- lex.yy.c

## 6.7 yy_trans_info Struct Reference

### Data Fields

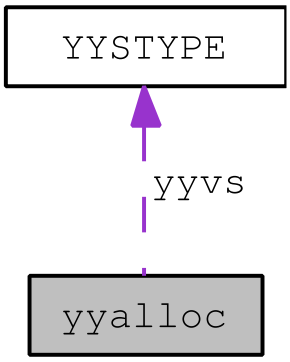
- flex_int32_t **yy_verify**
- flex_int32_t **yy_nxt**

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

- lex.yy.c

## 6.8 yyalloc Union Reference

Collaboration diagram for yyalloc:



### Data Fields

- yytype_int16 **yyss**
- YYSTYPE **yyvs**

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

- y.tab.c

## 6.9 YYSTYPE Union Reference

### Data Fields

- float **number**
- char ∗ **string**

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

- y.tab.c
- y.tab.h

# Chapter 7

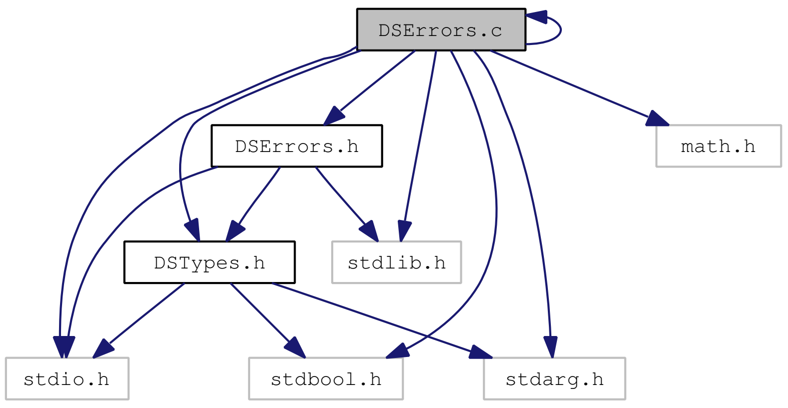
# File Documentation

## 7.1 DSErrors.c File Reference

Implementation file with functions for error and exception handling.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <execinfo.h>
#include "DSErrors.h"
#include <stdbool.h>
#include <stdarg.h>
#include "DSTypes.h"
#include "DSErrors.h"
```

Include dependency graph for DSErrors.c:



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

## Defines

- #define **STACK_TRACE_NUM** 10
- #define **MSIZE** 1200

## Functions

- void DSErrorSetPrintFunction (void(∗function)(const char ∗restrict))

  *FILE pointer used for default DSPrintFunction.*

- void **DSErrorSetErrorFile** (FILE ∗aFile)
- void DSErrorFunction (const char ∗M_DS_Message, char A_DS_ACTION, const char ∗FILEN, int LINE, const char ∗FUNC)

  *Implicit error handling function. Called by DSError which automatically adds file and line arguments.*

### 7.1.1 Detailed Description

Implementation file with functions for error and exception handling. This file specifies the design space standard for error handling. Contained here are the necessary macros and functions to succesfully report the errors throughout the design space library.

**Author**

Jason Lomnitz.

**Date**

2011

## 7.1.2 Function Documentation

### 7.1.2.1 void DSErrorFunction (const char $*$ *M_DS_Message*, char *A_DS_ACTION*, const char $*$ *FILEN*, int *LINE*, const char $*$ *FUNC*)

Implicit error handling function. Called by DSError which automatically adds file and line arguments.

This function is called implicity when using the DSError macro. The DSError adds the FILE and LINE argument, to report the error/warning at the appropriate file and line.

**See also**

DSError
Actions for DS Errors.

### 7.1.2.2 void DSErrorSetPrintFunction (void($*$)(const char $*$restrict) *function*)

FILE pointer used for default DSPrintFunction.

This pointer to a FILE tells the error handling system which FILE to print the error messages to. If this pointer is NULL, then the system uses the stderr file. This variable is only used internally with the default behavior of DSErrorFunction, however, it is intended to be used with custom functions.

**See also**

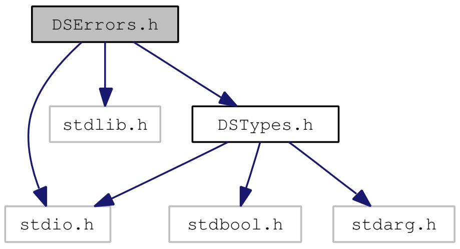
DSPrintFunction
DSErrorFunction

## 7.2 DSErrors.h File Reference

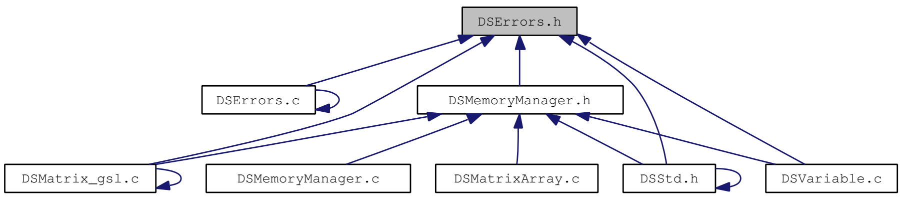
Header file with functions for error and exception handling.

```
#include <stdio.h>
#include <stdlib.h>
#include "DSTypes.h"
#include "DSIO.h"
```

Include dependency graph for DSErrors.h:



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



### Defines

- #define M_DS_NOFILE "File not found"

  *Message for no file found.*

- #define M_DS_NULL "NULL pointer"

  *Message for NULL pointer.*

- #define M_DS_NOFORMAT "Format not known"

  *Message for unknown format.*

- #define M_DS_WRONG "Inconsistent data"

  *Message for inconsistent data being used.*

- #define M_DS_EXISTS "Data already exists"

  *Message for data aleady existing.*

- #define M_DS_NOTHREAD "Thread not created"

  *Message for no thread created.*

- #define M_DS_MALLOC "Memory alloc failed"

  *Message for failure to allocate data.*

- #define M_DS_NOT_IMPL "Functionality not implemented"

  *Message for a feature not yet implemented.*

- #define A_DS_NOERROR 0

  *Value for no error.*

- #define A_DS_WARN -1

  *Value for a warning.*

- #define A_DS_ERROR -2

  *Value for an error.*

- #define A_DS_FATAL -3

  *Value for a fatal error, kills program.*

- #define A_DS_KILLNOW A_DS_FATAL

  *DEPRECATED:*

- #define DSError(M_DS_Message, A_DS_Action) DSErrorFunction(M_DS_Message, A_DS_-Action, __FILE__, __LINE__, __func__)

  *Error reporting macro.*

## Functions

- __deprecated void DSErrorSetPrintFunction (void(∗function)(const char ∗restrict))

  *FILE pointer used for default DSPrintFunction.*

- __deprecated void **DSErrorSetErrorFile** (FILE ∗aFile)
- void DSErrorFunction (const char ∗M_DS_Message, char A_DS_ACTION, const char ∗FILEN, int LINE, const char ∗FUNC)

  *Implicit error handling function. Called by DSError which automatically adds file and line arguments.*

### 7.2.1 Detailed Description

Header file with functions for error and exception handling. This file specifies the design space standard for error handling. Contained here are the necessary macros and functions to succesfully report the errors throughout the design space library.

**Author**

Jason Lomnitz.

**Date**

2011

### 7.2.2 Define Documentation

#### 7.2.2.1 #define DSError(M_DS_Message, A_DS_Action) DSErrorFunction(M_DS_Message, A_DS_Action, __FILE__, __LINE__, __func__)

Error reporting macro.

Definition of the error reporting macro used within the DesignSpace C toolbox, this is a define which takes a string, which may be a standard message, and an action and reports it via the standard warning and error posting functions in the standard IO functions. A default behavior of the DSError macro posts warning and errors to stderr, while a fatal error posts the error to stderr and aborts the program.

**See also**

DSPostWarning
DSPostError
DSPostFatalError
Messages for DS Errors.
Actions for DS Errors.

### 7.2.3 Function Documentation

#### 7.2.3.1 void DSErrorFunction (const char ∗ *M_DS_Message*, char *A_DS_ACTION*, const char ∗ *FILEN*, int *LINE*, const char ∗ *FUNC*)

Implicit error handling function. Called by DSError which automatically adds file and line arguments.

This function is called implicity when using the DSError macro. The DSError adds the FILE and LINE argument, to report the error/warning at the appropriate file and line.

**See also**

DSError
Actions for DS Errors.

#### 7.2.3.2 __deprecated void DSErrorSetPrintFunction (void(∗)(const char ∗restrict) *function*)

FILE pointer used for default DSPrintFunction.

This pointer to a FILE tells the error handling system which FILE to print the error messages to. If this pointer is NULL, then the system uses the stderr file. This variable is only used internally with the default behavior of DSErrorFunction, however, it is intended to be used with custom functions.

**See also**

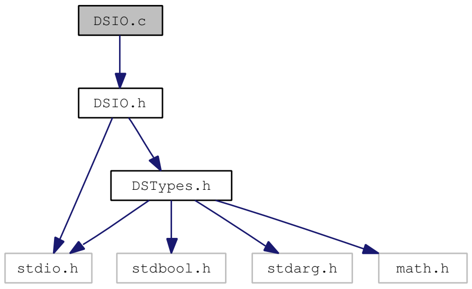
DSPrintFunction
DSErrorFunction

## 7.3 DSIO.c File Reference

Implementation file with standard input and output functions.

`#include "DSIO.h"`

Include dependency graph for DSIO.c:



### Functions

- void DSIOSetErrorFile (FILE ∗aFile)

  *Function to assign default error file.*

- void **DSIOSetPrintFunction** (int(∗printFunction)(const char ∗,...))
- void **DSIOSetPostWarningFunction** (void(∗warningFunction)(const char ∗message))
- void **DSIOSetPostErrorFunction** (void(∗errorFunction)(const char ∗message))
- void **DSIOSetPostFatalErrorFunction** (void(∗fatalErrorFunction)(const char ∗message))

### 7.3.1 Detailed Description

Implementation file with standard input and output functions. Copyright (C) 2011 Jason Lomnitz.

This file is part of the Design Space Toolbox V2 (C Library).

The Design Space Toolbox V2 is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

The Design Space Toolbox V2 is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with the Design Space Toolbox. If not, see <http://www.gnu.org/licenses/>.

**Author**

Jason Lomnitz.

**Date**

2011

## 7.3.2 Function Documentation

### 7.3.2.1 void DSIOSetErrorFile (FILE ∗ *aFile*)

Function to assign default error file.

This function is used to assign the default error file, DSIOErrorFile. Changing the error file should be done via this function, as it circumvents potential problems associated with dynamic linking.

**Parameters**

***aFile*** A FILE ∗ that will be used to write error messages when the default error posting mechanism is used.

**See also**

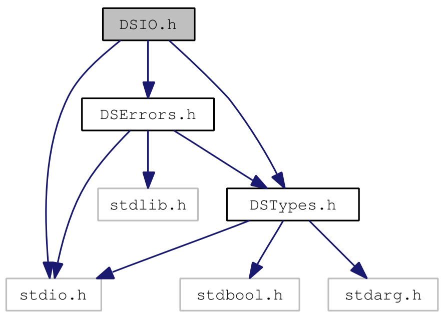
DSIOSetPostWarningFunction
DSIOSetPostErrorFunction
DSIOSetPostFatalErrorFunction
DSError

## 7.4 DSIO.h File Reference

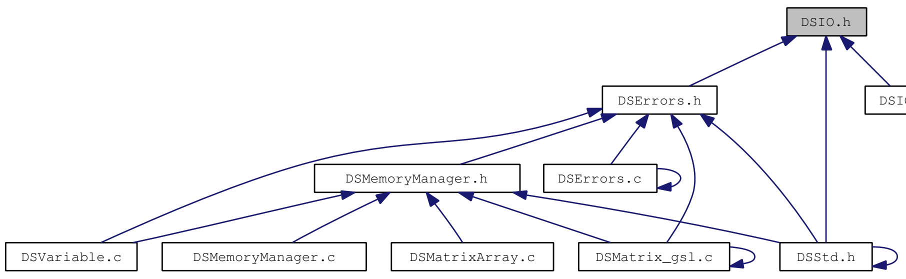
Header file with standard input and output functions.

```
#include <stdio.h>
#include "DSTypes.h"
```

Include dependency graph for DSIO.h:



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



### Functions

- void DSIOSetErrorFile (FILE ∗aFile)

    *Function to assign default error file.*

- void **DSIOSetPrintFunction** (int(∗printFunction)(const char ∗,...))
- void **DSIOSetPostWarningFunction** (void(∗warningFunction)(const char ∗message))
- void **DSIOSetPostErrorFunction** (void(∗errorFunction)(const char ∗message))
- void **DSIOSetPostFatalErrorFunction** (void(∗fatalErrorFunction)(const char ∗message))

### Variables

- int(∗ DSPrintf )(const char ∗,...)

    *Pointer to a function determining how messages are printed.*

- void(∗ **DSPostWarning** )(const char ∗message)
- void(∗ **DSPostError** )(const char ∗message)
- void(∗ **DSPostFatalError** )(const char ∗message)
- FILE ∗ **DSIOErrorFile**

### 7.4.1 Detailed Description

Header file with standard input and output functions. Copyright (C) 2011 Jason Lomnitz.

This file is part of the Design Space Toolbox V2 (C Library).

The Design Space Toolbox V2 is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

The Design Space Toolbox V2 is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with the Design Space Toolbox. If not, see <`http://www.gnu.org/licenses/`>.

**Author**

Jason Lomnitz.

**Date**

2011

**Todo**

Add 'plug-in' for printf function.
Define standard input and output file formats.

### 7.4.2 Function Documentation

#### 7.4.2.1 void DSIOSetErrorFile (FILE ∗ *aFile*)

Function to assign default error file.

This function is used to assign the default error file, DSIOErrorFile. Changing the error file should be done via this function, as it circumvents potential problems associated with dynamic linking.

**Parameters**

***aFile*** A FILE ∗ that will be used to write error messages when the default error posting mechanism is used.

**See also**

DSIOSetPostWarningFunction
DSIOSetPostErrorFunction
DSIOSetPostFatalErrorFunction
DSError

### 7.4.3 Variable Documentation

#### 7.4.3.1 int(∗ DSPrintf)(const char ∗,...)

Pointer to a function determining how messages are printed.

This pointer to a function tells the error handling system which function to call with the error messages. If this pointer is NULL, the design space toolbox should have a default printing format, using printf to stdout. This pointer is intended to be used to override default behavior to be override. An example could be by using the mexPrintf function in matlab.

**See also**

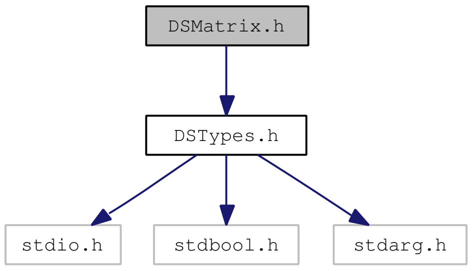
DSIOSetPrintFunction

## 7.5 DSMatrix.h File Reference

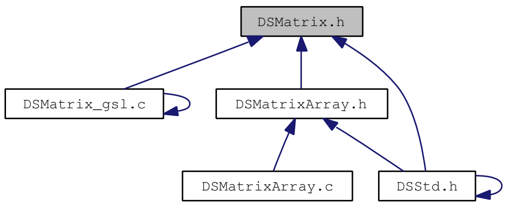
Header file with functions for dealing with matrices.

```
#include "DSTypes.h"
```

Include dependency graph for DSMatrix.h:



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



### Defines

- #define M_DS_MAT_NULL "Pointer to matrix is NULL"

  *Message for a NULL DSMatrix pointer.*

- #define M_DS_MAT_OUTOFBOUNDS "Row or column out of bounds"

  *Message for a row or column exceeding matrix bounds.*

- #define M_DS_MAT_NOINTERNAL "Matrix data is empty"

  *Message for a NULL internal matrix structure.*

- #define **DSMatrixRows**(x) ((x)->rows)
- #define **DSMatrixColumns**(x) ((x)->columns)
- #define **DSMatrixInternalPointer**(x) ((x)->mat)

### Enumerations

- enum { **__MAT_GSL__**, **__MAT_CLAPACK__** }

### Functions

- DSMatrix ∗ DSMatrixAlloc (const DSUInteger rows, const DSUInteger columns)

*Memory allocation for a DSMatrix using malloc.*

- DSMatrix ∗ DSMatrixCalloc (const DSUInteger rows, const DSUInteger columns)

  *Memory allocation for a DSMatrix using calloc.*

- DSMatrix ∗ DSMatrixCopy (const DSMatrix ∗original)

  *Copies a DSMatrix.*

- void DSMatrixFree (DSMatrix ∗matrix)

  *Freeing memory for DSMatrix.*

- DSMatrix ∗ DSMatrixIdentity (const DSUInteger size)

  *Allocates a new DSMatrix as an identity matrix.*

- DSMatrix ∗ DSMatrixRandomNumbers (const DSUInteger rows, const DSUInteger columns)

  *Allocates a new DSMatrix with random values between 0 and 1.*

- DSMatrix ∗ **DSMatrixWithVariablePoolValues** (const void ∗variablePool)
- double **DSMatrixDoubleValue** (const DSMatrix ∗matrix, const DSUInteger row, const DSUInteger column)
- void **DSMatrixSetRows** (DSMatrix ∗matrix, const DSUInteger rows)
- void **DSMatrixSetColumns** (DSMatrix ∗matrix, const DSUInteger columns)
- void **DSMatrixSetDoubleValue** (DSMatrix ∗matrix, const DSUInteger row, const DSUInteger column, const double value)
- void DSMatrixSetDoubleValueAll (DSMatrix ∗matrix, const double value)

  *Sets all the values of a matrix to a value.*

- void **DSMatrixSetDoubleValuesList** (DSMatrix ∗matrix, bool byColumns, DSUInteger numberOfValues, double firstValue,...)
- void **DSMatrixSetDoubleValues** (DSMatrix ∗matrix, bool byColumns, DSUInteger numberOfValues, double ∗values)
- void **DSMatrixRoundToSignificantFigures** (DSMatrix ∗matrix, const unsigned char figures)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingColumnList** (const DSMatrix ∗matrix, const DSUInteger numberOfColumns, const DSUInteger firstColumn,...)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingColumns** (const DSMatrix ∗matrix, const DSUInteger numberOfColumns, const DSUInteger ∗columns)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingRowList** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger firstRow,...)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingRows** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger ∗rows)
- DSMatrix ∗ **DSMatrixSubMatrixIncludingRowList** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger firstRow,...)
- DSMatrix ∗ **DSMatrixSubMatrixIncludingRows** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger ∗rows)
- DSMatrix ∗ **DSMatrixSubMatrixIncludingColumnList** (const DSMatrix ∗matrix, const DSUInteger numberOfColumns, const DSUInteger firstColumn,...)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingRowAndColumnList** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger numberOfColumns, const DSUInteger firstRow,...)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingRowsAndColumns** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger numberOfColumns, const DSUInteger ∗rows, const DSUInteger ∗columns)

- DSMatrix ∗ **DSMatrixSubMatrixIncludingColumns** (const DSMatrix ∗matrix, const DSUInteger numberOfColumns, const DSUInteger ∗columns)
- DSMatrix ∗ **DSMatrixSubMatrixIncludingRowAndColumnList** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger numberOfColumns, const DSUInteger firstRow,...)
- DSMatrix ∗ **DSMatrixAppendMatrices** (const DSMatrix ∗firstMatrix, const DSMatrix ∗secondMatrix, const bool byColumn)
- void **DSMatrixSwitchRows** (DSMatrix ∗matrix, const DSUInteger rowA, const DSUInteger rowB)

- void **DSMatrixSwitchColumns** (DSMatrix ∗matrix, const DSUInteger columnA, const DSUInteger columnB)
- void **DSMatrixPrint** (DSMatrix ∗matrix)
- bool **DSMatrixIsIdentity** (const DSMatrix ∗matrix)
- bool **DSMatrixIsSquare** (const DSMatrix ∗matrix)
- DSUInteger **DSMatrixRank** (const DSMatrix ∗matrix)
- double **minimumValue** (const DSMatrix ∗matrix, const bool shouldExcludeZero)
- double **maximumValue** (const DSMatrix ∗matrix, const bool shouldExcludeZero)
- DSMatrix ∗ **DSMatrixBySubstractingMatrix** (const DSMatrix ∗lvalue, const DSMatrix ∗rvalue)
- DSMatrix ∗ **DSMatrixByAddingMatrix** (const DSMatrix ∗lvalue, const DSMatrix ∗rvalue)
- DSMatrix ∗ **DSMatrixByDividingMatrix** (const DSMatrix ∗lvalue, const DSMatrix ∗rvalue)
- DSMatrix ∗ DSMatrixByMultiplyingMatrix (const DSMatrix ∗lvalue, const DSMatrix ∗rvalue)
- DSMatrix ∗ **DSMatrixByApplyingFunction** (const DSMatrix ∗mvalue, double(∗function)(double))

- DSMatrix ∗ **DSMatrixBySubstractingScalar** (const DSMatrix ∗lvalue, const double rvalue)
- DSMatrix ∗ **DSMatrixByAddingScalar** (const DSMatrix ∗lvalue, const double rvalue)
- DSMatrix ∗ **DSMatrixByDividingScalar** (const DSMatrix ∗lvalue, const double rvalue)
- DSMatrix ∗ **DSMatrixByMultiplyingScalar** (const DSMatrix ∗lvalue, const double rvalue)
- double **DSMatrixDeterminant** (const DSMatrix ∗matrix)
- DSMatrix ∗ **DSMatrixTranspose** (const DSMatrix ∗matrix)
- DSMatrix ∗ **DSMatrixInverse** (const DSMatrix ∗matrix)
- DSMatrixArray ∗ **DSMatrixSVD** (const DSMatrix ∗matrix)
- DSMatrix ∗ **DSMatrixRightNullspace** (const DSMatrix ∗matrix)
- DSMatrixArray ∗ DSMatrixPLUDecomposition (const DSMatrix ∗matrix)

  *Creates a LU decomposition and returns the permutation matrix.*

- double ∗ **DSMatrixDataForGLPK** (const DSMatrix ∗matrix)
- int ∗ **DSMatrixRowsForGLPK** (const DSMatrix ∗matrix)
- int ∗ **DSMatrixColumnsForGLPK** (const DSMatrix ∗matrix)

## 7.5.1 Detailed Description

Header file with functions for dealing with matrices. Copyright (C) 2011 Jason Lomnitz.

This file is part of the Design Space Toolbox V2 (C Library).

**Author**

Jason Lomnitz.

**Date**

2011

## 7.5.2 Function Documentation

#### 7.5.2.1 DSMatrix∗ DSMatrixAlloc (const DSUInteger *rows*, const DSUInteger *columns*)

Memory allocation for a DSMatrix using malloc.

Creates a new matrix of a particular size. The matrix that is allocated has all the values of the matrix defaulted to 0. The internal matrix pointer must be set to NULL; otherwise, the size of the matrix cannot be changed.

**Parameters**

***rows*** A DSUInteger with the number of rows in the new matrix.

***columns*** A DSUInteger with the number of columns in the new matrix.

**Returns**

If the matrix was created, a new pointer to a DSMatrix is returned. Otherwise, NULL is returned.

#### 7.5.2.2 DSMatrix∗ DSMatrixByMultiplyingMatrix (const DSMatrix ∗ *lvalue*, const DSMatrix ∗ *rvalue*)

Multiplication by a matrix inverse or pseudoinverse

#### 7.5.2.3 DSMatrix∗ DSMatrixCalloc (const DSUInteger *rows*, const DSUInteger *columns*)

Memory allocation for a DSMatrix using calloc.

Creates a new matrix of a particular size. The matrix that is allocated has all the values of the matrix defaulted to 0. The internal matrix pointer must be set to NULL; otherwise, the size of the matrix cannot be changed.

**Parameters**

***rows*** A DSUInteger with the number of rows in the new matrix.

***columns*** A DSUInteger with the number of columns in the new matrix.

**Returns**

If the matrix was created, a new pointer to a DSMatrix is returned. Otherwise, NULL is returned.

#### 7.5.2.4 DSMatrix∗ DSMatrixCopy (const DSMatrix ∗ *original*)

Copies a DSMatrix.

Creates a new matrix with the exact same size and contents as some other matrix. The new matrix is allocated, and thus must be freed.

**Parameters**

***original*** The DSMatrix to be copied.

**Returns**

If the copy was succesful, a pointer to a copy of the DSMatrix is returned. Otherwise, NULL is returned.

#### 7.5.2.5 void DSMatrixFree (DSMatrix ∗ *matrix*)

Freeing memory for DSMatrix.

Frees the memory associated with a DSMatrix data type. This function is a wrapper for the necessary steps needed to free the internal structure of the DSMatrix data type.

**Parameters**

***matrix*** The DSMatrix to be freed.

#### 7.5.2.6 DSMatrix∗ DSMatrixIdentity (const DSUInteger *size*)

Allocates a new DSMatrix as an identity matrix.

Allocates a square matrix of a specified size, and initializes the diagonal values to 1 and all the other values to 0, creating an identity matrix. The new matrix is therefore an identity matrix.

**Parameters**

***size*** A DSUInteger containing the number of rows and columns in the matrix.

**Returns**

If the identity matrix was succesfully created, a pointer to the DSMatrix is returned. Otherwise, NULL is returned.

#### 7.5.2.7 DSMatrixArray∗ DSMatrixPLUDecomposition (const DSMatrix ∗ *A*)

Creates a LU decomposition and returns the permutation matrix.

This function creates a LU decomposition of a DSMatrix A. This function creates an array of three matrices: a DSMatrix P, a DSMatrix L and a DSMatrix U; where $PA = LU$.

**Parameters**

***A*** A DSMatrix containing the matrix to be decomposed.

#### 7.5.2.8 DSMatrix∗ DSMatrixRandomNumbers (const DSUInteger *rows*, const DSUInteger *columns*)

Allocates a new DSMatrix with random values between 0 and 1.

Allocates a new DSMatrix with a specified size. The values of each of the entries in the matrix are randomly selected between 0 and 1.

**Parameters**

***rows*** A DSUInteger with the number of rows in the new matrix.

***columns*** A DSUInteger with the number of columns in the new matrix.

**Returns**

If the matrix was created, a new pointer to a DSMatrix is returned. Otherwise, NULL is returned.

#### 7.5.2.9 void DSMatrixSetDoubleValueAll (DSMatrix ∗ *matrix*, const double *value*)

Sets all the values of a matrix to a value.

This function does not allocate the necessary memory; instead it goes through all the rows and columns of the matrix, assigning them the specified value.

**Parameters**

***matrix*** The DSMatrix that will be assigned the value.

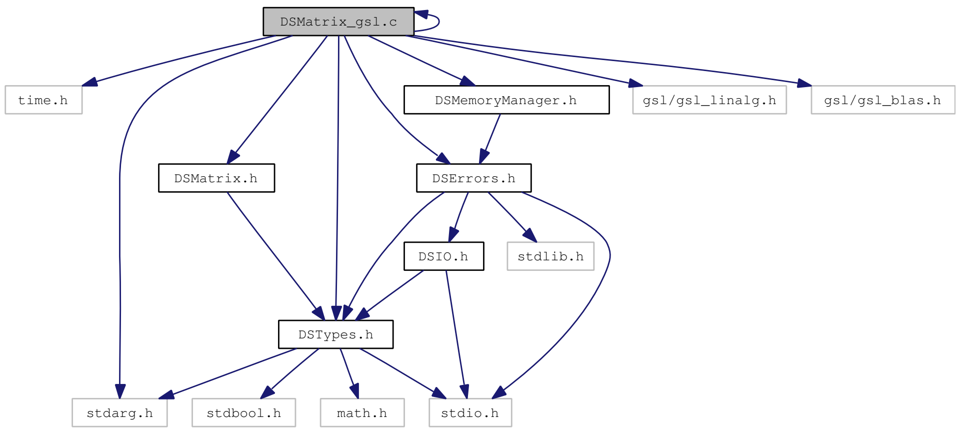
***value*** The double variable whose value will be assigned.

## 7.6 DSMatrix_gsl.c File Reference

Implementation file with functions for dealing with matrices using the GNU Scientific Library (gsl).

```
#include <time.h>
#include <stdarg.h>
#include "DSMatrix.h"
#include "DSTypes.h"
#include "DSErrors.h"
#include "DSMemoryManager.h"
#include "DSMatrix.h"
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_blas.h>
```

Include dependency graph for DSMatrix_gsl.c:



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

### Functions

- DSMatrix ∗ DSMatrixAlloc (const DSUInteger rows, const DSUInteger columns)

  *Memory allocation for a DSMatrix using malloc.*

- DSMatrix ∗ DSMatrixCalloc (const DSUInteger rows, const DSUInteger columns)

  *Memory allocation for a DSMatrix using calloc.*

- DSMatrix ∗ DSMatrixCopy (const DSMatrix ∗original)

  *Copies a DSMatrix.*

- void DSMatrixFree (DSMatrix ∗matrix)

*Freeing memory for DSMatrix.*

- DSMatrix ∗ DSMatrixIdentity (const DSUInteger size)

  *Allocates a new DSMatrix as an identity matrix.*

- DSMatrix ∗ DSMatrixRandomNumbers (const DSUInteger rows, const DSUInteger columns)

  *Allocates a new DSMatrix with random values between 0 and 1.*

- DSMatrix ∗ **DSMatrixWithVariablePoolValues** (const void ∗variablePool)
- double **DSMatrixDoubleValue** (const DSMatrix ∗matrix, const DSUInteger row, const DSUInteger column)
- void **DSMatrixSetRows** (DSMatrix ∗matrix, const DSUInteger rows)
- void **DSMatrixSetColumns** (DSMatrix ∗matrix, const DSUInteger columns)
- void **DSMatrixSetDoubleValue** (DSMatrix ∗matrix, const DSUInteger row, const DSUInteger column, const double value)
- void **DSMatrixSetDoubleValuesList** (DSMatrix ∗matrix, bool byColumns, DSUInteger numberOfValues, double firstValue,...)
- void **DSMatrixSetDoubleValues** (DSMatrix ∗matrix, bool byColumns, DSUInteger numberOfValues, double ∗values)
- void DSMatrixSetDoubleValueAll (DSMatrix ∗matrix, const double value)

  *Sets all the values of a matrix to a value.*

- void **DSMatrixRoundToSignificantFigures** (DSMatrix ∗matrix, const unsigned char figures)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingRowList** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger firstRow,...)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingRows** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger ∗rows)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingColumnList** (const DSMatrix ∗matrix, const DSUInteger numberOfColumns, const DSUInteger firstColumn,...)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingColumns** (const DSMatrix ∗matrix, const DSUInteger numberOfColumns, const DSUInteger ∗columns)
- DSMatrix ∗ **DSMatrixSubMatrixIncludingRowList** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger firstRow,...)
- DSMatrix ∗ **DSMatrixSubMatrixIncludingRows** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger ∗rows)
- DSMatrix ∗ **DSMatrixSubMatrixIncludingColumnList** (const DSMatrix ∗matrix, const DSUInteger numberOfColumns, const DSUInteger firstColumn,...)
- DSMatrix ∗ **DSMatrixSubMatrixIncludingColumns** (const DSMatrix ∗matrix, const DSUInteger numberOfColumns, const DSUInteger ∗columns)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingRowAndColumnList** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger numberOfColumns, const DSUInteger firstRow,...)
- DSMatrix ∗ **DSMatrixSubMatrixExcludingRowsAndColumns** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger numberOfColumns, const DSUInteger ∗rows, const DSUInteger ∗columns)
- DSMatrix ∗ **DSMatrixSubMatrixIncludingRowAndColumnList** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger numberOfColumns, const DSUInteger firstRow,...)
- DSMatrix ∗ **DSMatrixSubMatrixIncludingRowsAndColumns** (const DSMatrix ∗matrix, const DSUInteger numberOfRows, const DSUInteger numberOfColumns, const DSUInteger ∗rows, const DSUInteger ∗columns)
- DSMatrix ∗ **DSMatrixAppendMatrices** (const DSMatrix ∗firstMatrix, const DSMatrix ∗secondMatrix, const bool byColumn)

- void **DSMatrixSwitchRows** (DSMatrix ∗matrix, const DSUInteger rowA, const DSUInteger rowB)

- void **DSMatrixSwitchColumns** (DSMatrix ∗matrix, const DSUInteger columnA, const DSUInteger columnB)
- void **DSMatrixPrint** (DSMatrix ∗matrix)
- bool **DSMatrixIsIdentity** (const DSMatrix ∗matrix)
- bool **DSMatrixIsSquare** (const DSMatrix ∗matrix)
- DSUInteger **DSMatrixRank** (const DSMatrix ∗matrix)
- double **minimumValue** (const DSMatrix ∗matrix, const bool shouldExcludeZero)
- double **maximumValue** (const DSMatrix ∗matrix, const bool shouldExcludeZero)
- DSMatrix ∗ **DSMatrixBySubstractingMatrix** (const DSMatrix ∗lvalue, const DSMatrix ∗rvalue)
- DSMatrix ∗ **DSMatrixByAddingMatrix** (const DSMatrix ∗lvalue, const DSMatrix ∗rvalue)
- DSMatrix ∗ **DSMatrixByDividingMatrix** (const DSMatrix ∗lvalue, const DSMatrix ∗rvalue)
- DSMatrix ∗ DSMatrixByMultiplyingMatrix (const DSMatrix ∗lvalue, const DSMatrix ∗rvalue)
- DSMatrix ∗ **DSMatrixByApplyingFunction** (const DSMatrix ∗mvalue, double(∗function)(double))

- DSMatrix ∗ **DSMatrixBySubstractingScalar** (const DSMatrix ∗lvalue, const double rvalue)
- DSMatrix ∗ **DSMatrixByAddingScalar** (const DSMatrix ∗lvalue, const double rvalue)
- DSMatrix ∗ **DSMatrixByDividingScalar** (const DSMatrix ∗lvalue, const double rvalue)
- DSMatrix ∗ **DSMatrixByMultiplyingScalar** (const DSMatrix ∗lvalue, const double rvalue)
- double **DSMatrixDeterminant** (const DSMatrix ∗matrix)
- DSMatrix ∗ **DSMatrixTranspose** (const DSMatrix ∗matrix)
- DSMatrix ∗ **DSMatrixInverse** (const DSMatrix ∗matrix)
- DSMatrixArray ∗ **DSMatrixSVD** (const DSMatrix ∗matrix)
- DSMatrix ∗ **DSMatrixRightNullspace** (const DSMatrix ∗matrix)
- DSMatrixArray ∗ DSMatrixPLUDecomposition (const DSMatrix ∗A)

    *Creates a LU decomposition and returns the permutation matrix.*

- double ∗ **DSMatrixDataForGLPK** (const DSMatrix ∗matrix)
- int ∗ **DSMatrixRowsForGLPK** (const DSMatrix ∗matrix)
- int ∗ **DSMatrixColumnsForGLPK** (const DSMatrix ∗matrix)

### 7.6.1 Detailed Description

Implementation file with functions for dealing with matrices using the GNU Scientific Library (gsl). Copyright (C) 2011 Jason Lomnitz.

This file is part of the Design Space Toolbox V2 (C Library).

**Author**

Jason Lomnitz.

**Date**

2011

## 7.6.2 Function Documentation

### 7.6.2.1 DSMatrix∗ DSMatrixAlloc (const DSUInteger *rows*, const DSUInteger *columns*)

Memory allocation for a DSMatrix using malloc.

Creates a new matrix of a particular size. The matrix that is allocated has all the values of the matrix defaulted to 0. The internal matrix pointer must be set to NULL; otherwise, the size of the matrix cannot be changed.

**Parameters**

***rows*** A DSUInteger with the number of rows in the new matrix.

***columns*** A DSUInteger with the number of columns in the new matrix.

**Returns**

If the matrix was created, a new pointer to a DSMatrix is returned. Otherwise, NULL is returned.

### 7.6.2.2 DSMatrix∗ DSMatrixByMultiplyingMatrix (const DSMatrix ∗ *lvalue*, const DSMatrix ∗ *rvalue*)

Multiplication by a matrix inverse or pseudoinverse

### 7.6.2.3 DSMatrix∗ DSMatrixCalloc (const DSUInteger *rows*, const DSUInteger *columns*)

Memory allocation for a DSMatrix using calloc.

Creates a new matrix of a particular size. The matrix that is allocated has all the values of the matrix defaulted to 0. The internal matrix pointer must be set to NULL; otherwise, the size of the matrix cannot be changed.

**Parameters**

***rows*** A DSUInteger with the number of rows in the new matrix.

***columns*** A DSUInteger with the number of columns in the new matrix.

**Returns**

If the matrix was created, a new pointer to a DSMatrix is returned. Otherwise, NULL is returned.

### 7.6.2.4 DSMatrix∗ DSMatrixCopy (const DSMatrix ∗ *original*)

Copies a DSMatrix.

Creates a new matrix with the exact same size and contents as some other matrix. The new matrix is allocated, and thus must be freed.

**Parameters**

*original* The DSMatrix to be copied.

**Returns**

If the copy was succesful, a pointer to a copy of the DSMatrix is returned. Otherwise, NULL is returned.

#### 7.6.2.5 void DSMatrixFree (DSMatrix ∗ *matrix*)

Freeing memory for DSMatrix.

Frees the memory associated with a DSMatrix data type. This function is a wrapper for the necessary steps needed to free the internal structure of the DSMatrix data type.

**Parameters**

*matrix* The DSMatrix to be freed.

#### 7.6.2.6 DSMatrix∗ DSMatrixIdentity (const DSUInteger *size*)

Allocates a new DSMatrix as an identity matrix.

Allocates a square matrix of a specified size, and initializes the diagonal values to 1 and all the other values to 0, creating an identity matrix. The new matrix is therefore an identity matrix.

**Parameters**

*size* A DSUInteger containing the number of rows and columns in the matrix.

**Returns**

If the identity matrix was succesfully created, a pointer to the DSMatrix is returned. Otherwise, NULL is returned.

#### 7.6.2.7 DSMatrixArray∗ DSMatrixPLUDecomposition (const DSMatrix ∗ *A*)

Creates a LU decomposition and returns the permutation matrix.

This function creates a LU decomposition of a DSMatrix A. This function creates an array of three matrices: a DSMatrix P, a DSMatrix L and a DSMatrix U; where $PA = LU$.

**Parameters**

***A*** A DSMatrix containing the matrix to be decomposed.

#### 7.6.2.8 DSMatrix∗ DSMatrixRandomNumbers (const DSUInteger *rows*, const DSUInteger *columns*)

Allocates a new DSMatrix with random values between 0 and 1.

Allocates a new DSMatrix with a specified size. The values of each of the entries in the matrix are randomly selected between 0 and 1.

**Parameters**

***rows*** A DSUInteger with the number of rows in the new matrix.

***columns*** A DSUInteger with the number of columns in the new matrix.

**Returns**

If the matrix was created, a new pointer to a DSMatrix is returned. Otherwise, NULL is returned.

#### 7.6.2.9 void DSMatrixSetDoubleValueAll (DSMatrix ∗ *matrix*, const double *value*)

Sets all the values of a matrix to a value.

This function does not allocate the necessary memory; instead it goes through all the rows and columns of the matrix, assigning them the specified value.

**Parameters**

***matrix*** The DSMatrix that will be assigned the value.

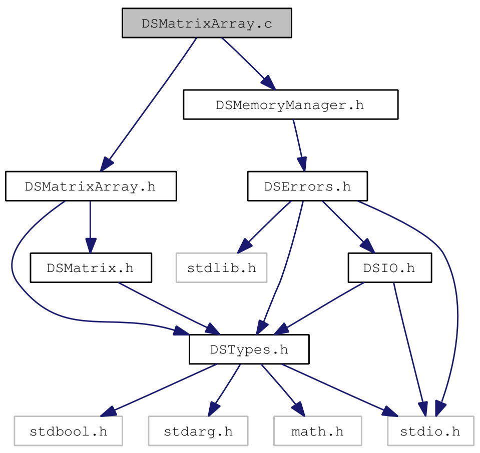
***value*** The double variable whose value will be assigned.

## 7.7 DSMatrixArray.c File Reference

Implementation file with functions for dealing with matrix arrays.

```
#include "DSMatrixArray.h"
#include "DSMemoryManager.h"
```

Include dependency graph for DSMatrixArray.c:



### Functions

- DSMatrixArray ∗ DSMatrixArrayAlloc (void)

  *Memory allocation for a DSMatrixArray.*

- DSMatrixArray ∗ DSMatrixArrayCopy (const DSMatrixArray ∗array)

  *Copies a DSMatrixArray.*

- void DSMatrixArrayFree (DSMatrixArray ∗array)

  *Freeing memory for DSMatrixArray.*

- DSMatrix ∗ DSMatrixArrayMatrix (const DSMatrixArray ∗array, const DSUInteger index)

  *Function to access a matrix in the DSMatrixArray.*

- void DSMatrixArrayAddMatrix (DSMatrixArray ∗array, const DSMatrix ∗matrixToAdd)

  *Function to add a new matrix to the DSMatrixArray.*

### 7.7.1 Detailed Description

Implementation file with functions for dealing with matrix arrays. Copyright (C) 2011 Jason Lomnitz.

This file is part of the Design Space Toolbox V2 (C Library).

**Author**

Jason Lomnitz.

**Date**

2011

## 7.7.2 Function Documentation

### 7.7.2.1 void DSMatrixArrayAddMatrix (DSMatrixArray ∗ *array*, const DSMatrix ∗ *matrixToAdd*)

Function to add a new matrix to the DSMatrixArray.

This function is the standard mechanism to add a DSMatrix to a DSMatrixArray. This function allocates the necessary space in the internal C array, and adds the DSMatrix to the end of the array. Once added to the matrix array, the memory is managed by the matrix array and is freed upon calling DSMatrixArrayFree.

**Parameters**

***array*** The DSMatrixArray that will have a new matrix added.

***matrixToAdd*** The DSMatrix to be added to the matrix array.

### 7.7.2.2 DSMatrixArray∗ DSMatrixArrayAlloc (void)

Memory allocation for a DSMatrixArray.

Creates a new DSMatrixArray with no matrices. As matrices are added, the matrix array grows, therefore the matrix array is initialized to 0, with a NULL internal pointer and number of matrices set to 0.

**Returns**

If the matrix array was created, a new pointer to a DSMatrix is returned. Otherwise, NULL is returned.

### 7.7.2.3 DSMatrixArray∗ DSMatrixArrayCopy (const DSMatrixArray ∗ *array*)

Copies a DSMatrixArray.

Creates a new DSMatrixArray with the exact same data and contents as some other matrix array. The matrices in the new DSMatrixArray are copies of the matrices in the original matrix array.

**Parameters**

***array*** The DSMatrixArray to be copied.

**Returns**

If the copy was succesful, a pointer to a copy of the DSMatrixArray is returned. Otherwise, NULL is returned.

**See also**

DSMatrixCopy

#### 7.7.2.4 void DSMatrixArrayFree (DSMatrixArray ∗ *array*)

Freeing memory for DSMatrixArray.

Frees the memory associated with a DSMatrixArray data type. This function is a wrapper for the necessary steps needed to free the internal structure of the DSMatrixArray, this includes calling DSMatrixFree for each of the contained matrices, freeing the internal pointer to the array of matrices, and the DSMatrixArray data type itself.

**Parameters**

***array*** The DSMatrixArray to be freed.

#### 7.7.2.5 DSMatrix∗ DSMatrixArrayMatrix (const DSMatrixArray ∗ *array*, const DSUInteger *index*)

Function to access a matrix in the DSMatrixArray.

This accessor function returns the DSMatrix at the specified index of the DSMatrixArray. This function is the basic accessor function, and should always be used to access a matrix in a DSMatrixArray.

**Parameters**

***array*** The DSMatrixArray containing the matrix to be accessed.

***index*** The DSUInteger specifying the index in the C array of matrices contained in the DSMatrixArray.

**Returns**

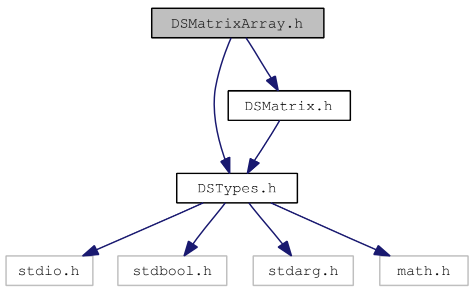
If the DSMatrix at the specified index was found, the pointer to that matrix is returned. Otherwise, NULL is returned.

## 7.8 DSMatrixArray.h File Reference

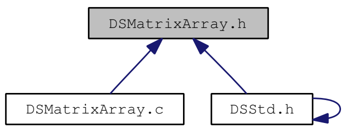
Header file with functions for dealing with matrix arrays.

```
#include "DSTypes.h"
#include "DSMatrix.h"
```

Include dependency graph for DSMatrixArray.h:



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



### Defines

- #define DSMatrixArrayNumberOfMatrices(x) ((x)->numberOfMatrices)

  *Accessor function to retrieve number of matrices in the Matrix array.*

- #define DSMatrixArrayInternalPointer(x) ((x)->matrices)

  *Accessor function to retrieve the pointer to the C matrix array.*

### Functions

- DSMatrixArray ∗ DSMatrixArrayAlloc (void)

  *Memory allocation for a DSMatrixArray.*

- DSMatrixArray ∗ DSMatrixArrayCopy (const DSMatrixArray ∗array)

  *Copies a DSMatrixArray.*

- void DSMatrixArrayFree (DSMatrixArray ∗array)

  *Freeing memory for DSMatrixArray.*

- DSMatrix ∗ DSMatrixArrayMatrix (const DSMatrixArray ∗array, const DSUInteger index)

  *Function to access a matrix in the DSMatrixArray.*

- void DSMatrixArrayAddMatrix (DSMatrixArray ∗array, const DSMatrix ∗matrixToAdd)

  *Function to add a new matrix to the DSMatrixArray.*

### 7.8.1 Detailed Description

Header file with functions for dealing with matrix arrays. Copyright (C) 2011 Jason Lomnitz.

This file is part of the Design Space Toolbox V2 (C Library).

**Author**

Jason Lomnitz.

**Date**

2011

### 7.8.2 Function Documentation

#### 7.8.2.1 void DSMatrixArrayAddMatrix (DSMatrixArray ∗ *array*, const DSMatrix ∗ *matrixToAdd*)

Function to add a new matrix to the DSMatrixArray.

This function is the standard mechanism to add a DSMatrix to a DSMatrixArray. This function allocates the necessary space in the internal C array, and adds the DSMatrix to the end of the array. Once added to the matrix array, the memory is managed by the matrix array and is freed upon calling DSMatrixArrayFree.

**Parameters**

***array*** The DSMatrixArray that will have a new matrix added.

***matrixToAdd*** The DSMatrix to be added to the matrix array.

#### 7.8.2.2 DSMatrixArray∗ DSMatrixArrayAlloc (void)

Memory allocation for a DSMatrixArray.

Creates a new DSMatrixArray with no matrices. As matrices are added, the matrix array grows, therefore the matrix array is initialized to 0, with a NULL internal pointer and number of matrices set to 0.

**Returns**

If the matrix array was created, a new pointer to a DSMatrix is returned. Otherwise, NULL is returned.

#### 7.8.2.3 DSMatrixArray∗ DSMatrixArrayCopy (const DSMatrixArray ∗ *array*)

Copies a DSMatrixArray.

Creates a new DSMatrixArray with the exact same data and contents as some other matrix array. The matrices in the new DSMatrixArray are copies of the matrices in the original matrix array.

**Parameters**

*array* The DSMatrixArray to be copied.

**Returns**

If the copy was succesful, a pointer to a copy of the DSMatrixArray is returned. Otherwise, NULL is returned.

**See also**

DSMatrixCopy

#### 7.8.2.4 void DSMatrixArrayFree (DSMatrixArray ∗ *array*)

Freeing memory for DSMatrixArray.

Frees the memory associated with a DSMatrixArray data type. This function is a wrapper for the necessary steps needed to free the internal structure of the DSMatrixArray, this includes calling DSMatrixFree for each of the contained matrices, freeing the internal pointer to the array of matrices, and the DSMatrixArray data type itself.

**Parameters**

*array* The DSMatrixArray to be freed.

#### 7.8.2.5 DSMatrix∗ DSMatrixArrayMatrix (const DSMatrixArray ∗ *array*, const DSUInteger *index*)

Function to access a matrix in the DSMatrixArray.

This accessor function returns the DSMatrix at the specified index of the DSMatrixArray. This function is the basic accessor function, and should always be used to access a matrix in a DSMatrixArray.

**Parameters**

*array* The DSMatrixArray containing the matrix to be accessed.

*index* The DSUInteger specifying the index in the C array of matrices contained in the DSMatrixArray.

**Returns**

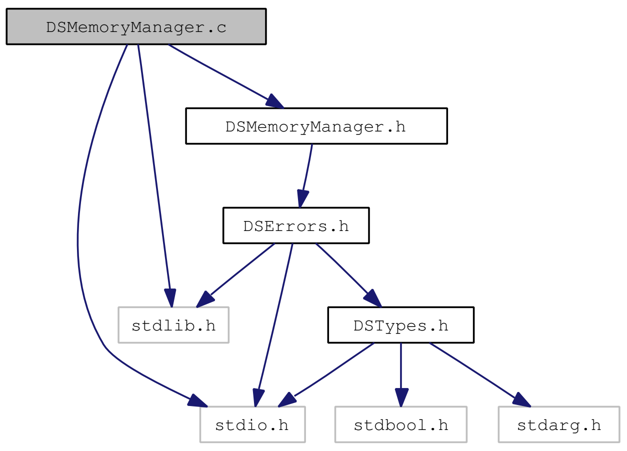
If the DSMatrix at the specified index was found, the pointer to that matrix is returned. Otherwise, NULL is returned.

## 7.9 DSMemoryManager.c File Reference

implementation file with functions for secure memory management.

```
#include <stdio.h>
#include <stdlib.h>
#include "DSMemoryManager.h"
```

Include dependency graph for DSMemoryManager.c:



### Functions

- void ∗ DSSecureMalloc (DSInteger size)

  *Function to securely allocate data using malloc.*

- void ∗ DSSecureCalloc (DSInteger count, DSInteger size)

  *Function to securely allocate data using calloc.*

- void ∗ DSSecureRealloc (void ∗ptr, DSInteger size)

  *Function to securely allocate data using realloc.*

- void DSSecureFree (void ∗ptr)

  *Function to securely free data.*

### 7.9.1 Detailed Description

implementation file with functions for secure memory management. This file specifies the design space standard for error handling. Contained here are the necessary macros and functions to succesfully report the errors throughout the design space library.

Copyright (C) 2011 Jason Lomnitz.

This file is part of the Design Space Toolbox V2 (C Library).

**Author**

Jason Lomnitz.

**Date**

2011

## 7.9.2 Function Documentation

### 7.9.2.1 void∗ DSSecureCalloc (DSInteger *count*, DSInteger *size*)

Function to securely allocate data using calloc.

This function is a secure calloc function which checks the allocated pointer. If the data pointer is null, indicative of errors allocating memory, the function issues a fatal error.

**Parameters**

***count*** A DSUInteger specifying the number of memory blocks being allocated.

***size*** The memory size of each block being allocated.

**Returns**

A pointer to the allocated data.

### 7.9.2.2 void DSSecureFree (void ∗ *ptr*)

Function to securely free data.

This function is a secure free function which checks the data pointer. If the data pointer is null, indicative of errors when freeing memory, the function issues a fatal error. This function calls malloc in case that pointer to be reallocated is NULL.

**Parameters**

***count*** A DSUInteger specifying the number of memory blocks being allocated.

***size*** The memory size of each block being allocated.

**Returns**

A pointer to the allocated data.

#### 7.9.2.3 void∗ DSSecureMalloc (DSInteger *size*)

Function to securely allocate data using malloc.

This function is a secure malloc function which checks the allocated pointer. If the data pointer is null, indicative of errors allocating memory, the function issues a fatal error.

**Parameters**

***size*** A DSUInteger specifying the size of memory being allocated.

**Returns**

A pointer to the allocated data.

#### 7.9.2.4 void∗ DSSecureRealloc (void ∗ *ptr*, DSInteger *size*)

Function to securely allocate data using realloc.

This function is a secure realloc function which checks the allocated pointer. If the data pointer is null, indicative of errors allocating memory, the function issues a fatal error. This function calls malloc in case that pointer to be reallocated is NULL.

**Parameters**

***count*** A DSUInteger specifying the number of memory blocks being allocated.

***size*** The memory size of each block being allocated.

**Returns**

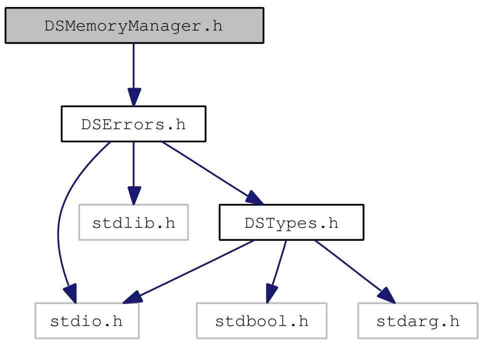
A pointer to the allocated data.

## 7.10 DSMemoryManager.h File Reference

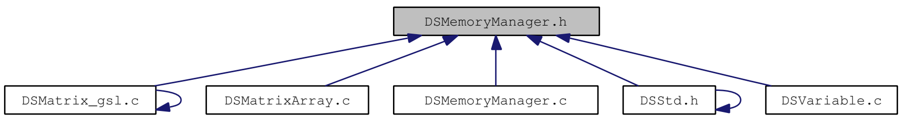
Header file with functions for secure memory allocation.

```
#include "DSErrors.h"
```

Include dependency graph for DSMemoryManager.h:



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



### Functions

- void ∗ DSSecureMalloc (DSInteger size)

  *Function to securely allocate data using malloc.*

- void ∗ DSSecureCalloc (DSInteger count, DSInteger size)

  *Function to securely allocate data using calloc.*

- void ∗ DSSecureRealloc (void ∗ptr, DSInteger size)

  *Function to securely allocate data using realloc.*

- void DSSecureFree (void ∗ptr)

  *Function to securely free data.*

### 7.10.1 Detailed Description

Header file with functions for secure memory allocation. This file specifies the design space standard for error handling. Contained here are the necessary macros and functions to succesfully report the errors throughout the design space library.

Copyright (C) 2011 Jason Lomnitz.

This file is part of the Design Space Toolbox V2 (C Library).

**Author**

Jason Lomnitz.

**Date**

2011

## 7.10.2 Function Documentation

### 7.10.2.1 void∗ DSSecureCalloc (DSInteger *count*, DSInteger *size*)

Function to securely allocate data using calloc.

This function is a secure calloc function which checks the allocated pointer. If the data pointer is null, indicative of errors allocating memory, the function issues a fatal error.

**Parameters**

***count*** A DSUInteger specifying the number of memory blocks being allocated.

***size*** The memory size of each block being allocated.

**Returns**

A pointer to the allocated data.

### 7.10.2.2 void DSSecureFree (void ∗ *ptr*)

Function to securely free data.

This function is a secure free function which checks the data pointer. If the data pointer is null, indicative of errors when freeing memory, the function issues a fatal error. This function calls malloc in case that pointer to be reallocated is NULL.

**Parameters**

***count*** A DSUInteger specifying the number of memory blocks being allocated.

***size*** The memory size of each block being allocated.

**Returns**

A pointer to the allocated data.

#### 7.10.2.3 void∗ DSSecureMalloc (DSInteger *size*)

Function to securely allocate data using malloc.

This function is a secure malloc function which checks the allocated pointer. If the data pointer is null, indicative of errors allocating memory, the function issues a fatal error.

**Parameters**

*size* A DSUInteger specifying the size of memory being allocated.

**Returns**

A pointer to the allocated data.

#### 7.10.2.4 void∗ DSSecureRealloc (void ∗ *ptr*, DSInteger *size*)

Function to securely allocate data using realloc.

This function is a secure realloc function which checks the allocated pointer. If the data pointer is null, indicative of errors allocating memory, the function issues a fatal error. This function calls malloc in case that pointer to be reallocated is NULL.

**Parameters**

*count* A DSUInteger specifying the number of memory blocks being allocated.

*size* The memory size of each block being allocated.

**Returns**

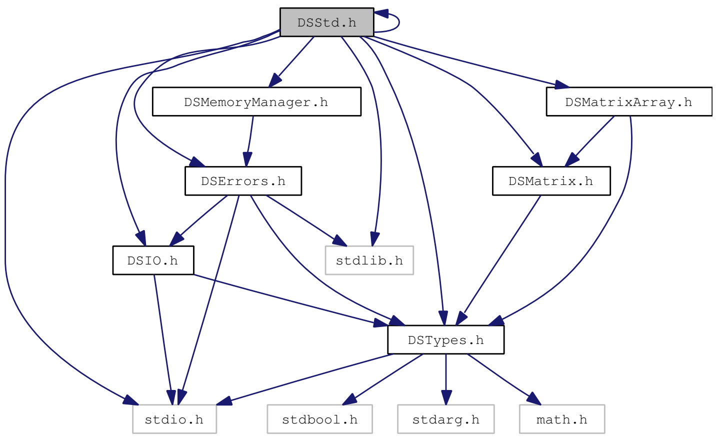
A pointer to the allocated data.

## 7.11 DSStd.h File Reference

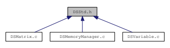
Header file for the design space toolbox.

```
#include <stdio.h>
#include <stdlib.h>
#include "DSTypes.h"
#include "DSIO.h"
#include "DSErrors.h"
#include "DSMemoryManager.h"
#include "DSVariable.h"
#include "DSMatrix.h"
#include "DSMatrixArray.h"
```

Include dependency graph for DSStd.h:



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



### 7.11.1 Detailed Description

Header file for the design space toolbox. Copyright (C) 2011 Jason Lomnitz.

This file is part of the Design Space Toolbox V2 (C Library).

**Author**

Jason Lomnitz.

**Date**

2011

**Todo**

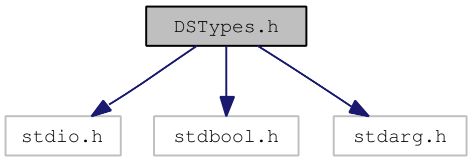
Add all previous functionality.
Add vertex enumeration functionality.

## 7.12 DSTypes.h File Reference

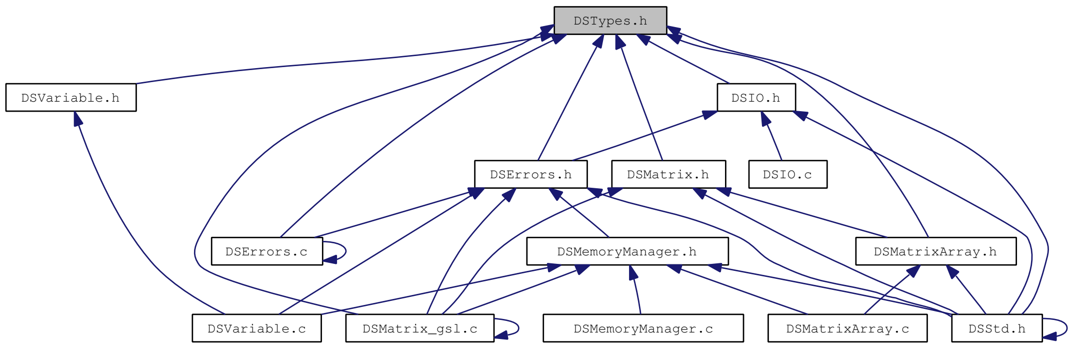
Header file with definitions for data types.

```
#include <stdio.h>
#include <stdbool.h>
#include <stdarg.h>
#include <math.h>
```

Include dependency graph for DSTypes.h:



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



### Data Structures

- struct DSVariable

  *Basic variable structure containing name, value and NSString with special unicode characters for greek letters.*

- struct _varDictionary

  *Internal dictionary structure.*

- struct DSMatrix

  *Data type representing a matrix.*

- struct DSMatrixArray

  *Data type representing an array of matrices.*

- struct DSSSystem

  *Data type representing an S-System.*

## Defines

- #define **endif**
- #define **__deprecated**
- #define **INFINITY** HUGE_VAL

## Typedefs

- typedef int **DSInteger**
- typedef unsigned int **DSUInteger**
- typedef struct _varDictionary DSVariablePool

  *Internal dictionary structure.*

## Enumerations

- enum DSException { DS_NOERROR = 0, DS_WARN, DS_ERROR }

  *Data type that is used to store errors.*

### 7.12.1 Detailed Description

Header file with definitions for data types. This file specifies the design space standard data types. Contained here are strictly the data type definitions, and specific macros to access data in data structures. Functions applying to these data types are contained elsewhere, and the individual data structures should refer to these files.

**Author**

Jason Lomnitz.

**Date**

2011

### 7.12.2 Typedef Documentation

#### 7.12.2.1 typedef struct _varDictionary DSVariablePool

Internal dictionary structure.

Internal dictionary for fast variable querying. The structure of the dictionary uses an alternative path, where each character is checked in order at each position, if there is a match, the next position is consequently checked. The dictionary should never be manipulated manually, adding, retrieving and removing variables should be done through the accesory functions.

**See also**

DSVariable

### 7.12.3 Enumeration Type Documentation

#### 7.12.3.1 enum DSException

Data type that is used to store errors.

Type definition for the internal error handling, independent from the error reporting mentioned in M_DS_-Messages and A_DS_Actions. The possible values this type definition has only symbolizes the severity of the error.

**See also**

DSErrors.h
Messages for DS Errors.
Actions for DS Errors.

**Enumerator:**

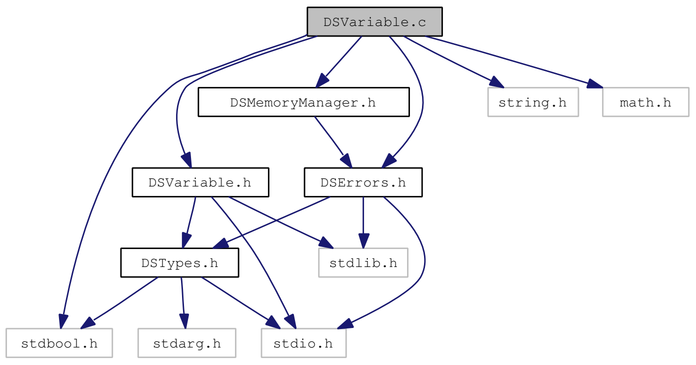
| | |
|---|---|
| ***DS_NOERROR*** | The value for no errors found. |
| ***DS_WARN*** | The value for a warning found. |
| ***DS_ERROR*** | The value for a error being found. |

## 7.13 DSVariable.c File Reference

Implementation file with functions for dealing with variables.

```
#include <stdbool.h>
#include <string.h>
#include <math.h>
#include "DSMemoryManager.h"
#include "DSErrors.h"
#include "DSVariable.h"
```

Include dependency graph for DSVariable.c:



### Functions

- DSVariable ∗ DSVariableAlloc (const char ∗name)

  *Creates a new DSVariable with INFINITY as a default value.*

- DSVariable ∗ **DSNewVariable** (const char ∗name)
- void DSVariableFree (DSVariable ∗var)

  *Function frees allocated memory of a DSVariable.*

- DSVariable ∗ DSVariableRetain (DSVariable ∗aVariable)

  *Function to increase variable retain count by one.*

- void DSVariableRelease (DSVariable ∗aVariable)

  *Function to decrease variable retain count by one.*

- DSVariablePool ∗ DSVariablePoolAddVariableWithName (const char ∗name, DSVariablePool ∗root)

  *Adds a DSVariable to the dictionary.*

- DSVariablePool ∗ DSVariablePoolAddVariable (DSVariable ∗newVar, DSVariablePool ∗root)

  *Adds a DSVariable to the dictionary.*

- DSVariable ∗ DSVariablePoolVariableWithName (const char ∗name, DSVariablePool ∗root)

  *Retrieves the variable in the dictionary with a matching name.*

- DSVariable ∗ **DSVariableWithName** (const char ∗name, DSVariablePool ∗root)
- void DSVariablePoolFree (DSVariablePool ∗root)

  *Function to remove all nodes in the dictionary.*

- void printVarDictionary (DSVariablePool ∗root, int indent)

  *Prints the VarDictionary.*

- void printMembers (DSVariablePool ∗root, char ∗buffer, int position)

  *Function that prints the members in the dictionary.*

### 7.13.1 Detailed Description

Implementation file with functions for dealing with variables. Copyright (C) 2011 Jason Lomnitz.

This file is part of the Design Space Toolbox V2 (C Library).

**Author**

Jason Lomnitz.

**Date**

2011

### 7.13.2 Function Documentation

#### 7.13.2.1 DSVariable∗ DSVariableAlloc (const char ∗ *name*)

Creates a new DSVariable with INFINITY as a default value.

This function may be used throughout, in order to create new variables consistently and portably. As variables are allocated individually, it is important to not that they should be released with the accesory method.

**Parameters**

***name*** A string with which to identify the DSVariable.

**Returns**

The pointer to the newly allocated DSVariable.

**See also**

DSVariable
DSVariableFree

#### 7.13.2.2 void DSVariableFree (DSVariable ∗ *var*)

Function frees allocated memory of a DSVariable.

This function should be used for each newDSVariable that is called. The internal structure is subject to changes in consequent versions and therefore freeing memory of DSVariables should be strictly through this function.

**Parameters**

***var*** The pointer to the variable to free.

#### 7.13.2.3 DSVariablePool∗ DSVariablePoolAddVariable (DSVariable ∗ *newVar*, DSVariablePool ∗ *root*)

Adds a DSVariable to the dictionary.

This is the function to add DSVariables to the dictionary. Since the dictionary works alphabetically, the root of the dictionary changes and is defined by this function, and the root, be it the new one or the old one, is returned. This function is used in creating a new dictionary.

**Parameters**

***newVar*** The pointer to the variable which is to be added to the dictionary.

***root*** The root of the dictionary. THE ROOT MAY BE REASSIGNED. If null, creates a new dictionary.

**Returns**

The root of the dictionary, which is likely to change.

**See also**

_varDictionary
DSVariableWithName
_addNewBranch

#### 7.13.2.4 DSVariablePool∗ DSVariablePoolAddVariableWithName (const char ∗ *name*, DSVariablePool ∗ *root*)

Adds a DSVariable to the dictionary.

This is the function to add DSVariables to the dictionary. Since the dictionary works alphabetically, the root of the dictionary changes and is defined by this function, and the root, be it the new one or the old one, is returned. This function is used in creating a new dictionary.

**Parameters**

***newVar*** The pointer to the variable which is to be added to the dictionary.

***root*** The root of the dictionary. THE ROOT MAY BE REASSIGNED. If null, creates a new dictionary.

**Returns**

The root of the dictionary, which is likely to change.

**See also**

_varDictionary
DSVariableWithName
_addNewBranch

#### 7.13.2.5 void DSVariablePoolFree (DSVariablePool ∗ *root*)

Function to remove all nodes in the dictionary.

The function is RECURSIVE and frees all the nodes past the root which is passed. In theory, it could be used to clear a portion of the dictionary, yet doing so would require direct manipulation of the dictionary which is strongly discouraged. The variables themselves are NOT freed, since the dictionary does not allocate them.

**Parameters**

***root*** The root of the dictionary which is to be cleared.

**See also**

DSVariablePool
DSVariableWithName
DSVariablePoolAddVariable

#### 7.13.2.6 DSVariable∗ DSVariablePoolVariableWithName (const char ∗ *name*, DSVariablePool ∗ *root*)

Retrieves the variable in the dictionary with a matching name.

This is the function for retriving a DSVariable from a dictionary.

**Parameters**

***name*** A string containing a NULL terminated string containing the name of the desired DSVariable.

***root*** The root of the dictionary, determined during the adding of variables.

**Returns**

A pointer to the variable. If there is no matching variable, NULL is returned.

**See also**

_checkNameAtPos
DSVariablePool
DSVariablePoolAddVariable

#### 7.13.2.7 void DSVariableRelease (DSVariable ∗ *aVariable*)

Function to decrease variable retain count by one.

Fast processing tree is made to decrease its retain count by one, when the retain count hits zero, the function DSVariableFree() is invoked, freeing the memory space. Fast processing tree does not have an equivalent to autorelease, forcing the developer to use greater care when directly managing memory.

**Parameters**

***aVariable*** The variable which will have its retain count reduced.

**See also**

DSVariableRetain
DSVariableFree

#### 7.13.2.8 DSVariable∗ DSVariableRetain (DSVariable ∗ *aVariable*)

Function to increase variable retain count by one.

Variables utilize a similar memory management system used in Objective-C NSObject subclasses, where objects a retained/released.

**Parameters**

***aVariable*** The variable which will have its retain count increased.

**Returns**

The same variable which received the retain count increase is returned, for convinience.

**See also**

DSVariableRelease

#### 7.13.2.9 void printMembers (DSVariablePool ∗ *root*, char ∗ *buffer*, int *position*)

Function that prints the members in the dictionary.

This function recursively checks the dictionary for all names, and prints them out. It requires of a buffer string which is used to store the current characters, since it is a recursive function.

**Parameters**

***root*** The root of the dictionary, and the current node for consequent calls.
***buffer*** The string in which to store the characters, function does not check size of buffer.
***position*** Should be 0 when called, increases as it searches the dictionary.

**See also**

_varDictionary
DSVariablePoolAddVariable

**Todo**

Create a wrapper function which creates the buffer, and initiates the position at 0.

#### 7.13.2.10 void printVarDictionary (DSVariablePool ∗ *root*, int *indent*)

Prints the VarDictionary.

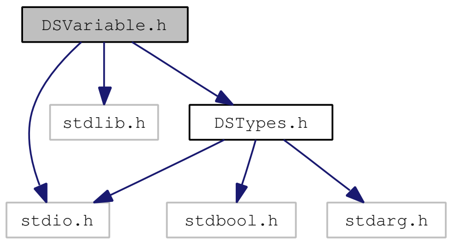
A debugging function which prints the dictionary structure to the stderr file

## 7.14 DSVariable.h File Reference

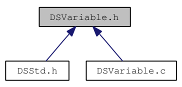
Header file with functions for dealing with variables.

```
#include <stdio.h>
#include <stdlib.h>
#include "DSTypes.h"
```

Include dependency graph for DSVariable.h:



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



### Defines

- #define DSVariableAssignValue(x, y) DSVariableSetValue(x, y)

  *Macro to directly change the value of a variable.*

- #define DSVariableReturnValue(x) DSVariableValue(x)

  *Macro to directly access the value of a variable.*

- #define **DSVariableSetValue**(x, y) (x)->value = (y)
- #define **DSVariableValue**(x) (x)->value
- #define **DSVariableName**(x) (x)->name

### Functions

- DSVariable ∗ DSVariableAlloc (const char ∗name)

  *Creates a new DSVariable with INFINITY as a default value.*

- void DSVariableFree (DSVariable ∗var)

  *Function frees allocated memory of a DSVariable.*

- DSVariable ∗ DSVariableRetain (DSVariable ∗aVariable)

  *Function to increase variable retain count by one.*

- void DSVariableRelease (DSVariable ∗aVariable)

*Function to decrease variable retain count by one.*

- __deprecated DSVariable ∗ **DSNewVariable** (const char ∗name)
- DSVariablePool ∗ DSVariablePoolAddVariableWithName (const char ∗name, DSVariablePool ∗root)

  *Adds a DSVariable to the dictionary.*

- DSVariablePool ∗ DSVariablePoolAddVariable (DSVariable ∗newVar, DSVariablePool ∗root)

  *Adds a DSVariable to the dictionary.*

- DSVariable ∗ DSVariablePoolVariableWithName (const char ∗name, DSVariablePool ∗root)

  *Retrieves the variable in the dictionary with a matching name.*

- void DSVariablePoolFree (DSVariablePool ∗root)

  *Function to remove all nodes in the dictionary.*

- __deprecated DSVariable ∗ **DSVariableWithName** (const char ∗name, DSVariablePool ∗root)

## 7.14.1 Detailed Description

Header file with functions for dealing with variables. Copyright (C) 2011 Jason Lomnitz.

This file is part of the Design Space Toolbox V2 (C Library).

The Design Space Toolbox V2 is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

The Design Space Toolbox V2 is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with the Design Space Toolbox. If not, see <http://www.gnu.org/licenses/>.

**Author**

Jason Lomnitz.

**Date**

2011

## 7.14.2 Function Documentation

### 7.14.2.1 DSVariable∗ DSVariableAlloc (const char ∗ *name*)

Creates a new DSVariable with INFINITY as a default value.

This function may be used throughout, in order to create new variables consistently and portably. As variables are allocated individually, it is important to not that they should be released with the accesory method.

**Parameters**

***name*** A string with which to identify the DSVariable.

**Returns**

The pointer to the newly allocated DSVariable.

**See also**

DSVariable
DSVariableFree

#### 7.14.2.2 void DSVariableFree (DSVariable ∗ *var*)

Function frees allocated memory of a DSVariable.

This function should be used for each newDSVariable that is called. The internal structure is subject to changes in consequent versions and therefore freeing memory of DSVariables should be strictly through this function.

**Parameters**

***var*** The pointer to the variable to free.

#### 7.14.2.3 DSVariablePool∗ DSVariablePoolAddVariable (DSVariable ∗ *newVar*, DSVariablePool ∗ *root*)

Adds a DSVariable to the dictionary.

This is the function to add DSVariables to the dictionary. Since the dictionary works alphabetically, the root of the dictionary changes and is defined by this function, and the root, be it the new one or the old one, is returned. This function is used in creating a new dictionary.

**Parameters**

***newVar*** The pointer to the variable which is to be added to the dictionary.

***root*** The root of the dictionary. THE ROOT MAY BE REASSIGNED. If null, creates a new dictionary.

**Returns**

The root of the dictionary, which is likely to change.

**See also**

_varDictionary
DSVariableWithName
_addNewBranch

#### 7.14.2.4 DSVariablePool∗ DSVariablePoolAddVariableWithName (const char ∗ *name*, DSVariablePool ∗ *root*)

Adds a DSVariable to the dictionary.

This is the function to add DSVariables to the dictionary. Since the dictionary works alphabetically, the root of the dictionary changes and is defined by this function, and the root, be it the new one or the old one, is returned. This function is used in creating a new dictionary.

**Parameters**

***newVar*** The pointer to the variable which is to be added to the dictionary.

***root*** The root of the dictionary. THE ROOT MAY BE REASSIGNED. If null, creates a new dictionary.

**Returns**

The root of the dictionary, which is likely to change.

**See also**

_varDictionary
DSVariableWithName
_addNewBranch

#### 7.14.2.5 void DSVariablePoolFree (DSVariablePool ∗ *root*)

Function to remove all nodes in the dictionary.

The function is RECURSIVE and frees all the nodes past the root which is passed. In theory, it could be used to clear a portion of the dictionary, yet doing so would require direct manipulation of the dictionary which is strongly discouraged. The variables themselves are NOT freed, since the dictionary does not allocate them.

**Parameters**

***root*** The root of the dictionary which is to be cleared.

**See also**

DSVariablePool
DSVariableWithName
DSVariablePoolAddVariable

#### 7.14.2.6 DSVariable∗ DSVariablePoolVariableWithName (const char ∗ *name*, DSVariablePool ∗ *root*)

Retrieves the variable in the dictionary with a matching name.

This is the function for retriving a DSVariable from a dictionary.

**Parameters**

***name*** A string containing a NULL terminated string containing the name of the desired DSVariable.

***root*** The root of the dictionary, determined during the adding of variables.

**Returns**

A pointer to the variable. If there is no matching variable, NULL is returned.

**See also**

_checkNameAtPos
DSVariablePool
DSVariablePoolAddVariable

#### 7.14.2.7 void DSVariableRelease (DSVariable ∗ *aVariable*)

Function to decrease variable retain count by one.

Fast processing tree is made to decrease its retain count by one, when the retain count hits zero, the function DSVariableFree() is invoked, freeing the memory space. Fast processing tree does not have an equivalent to autorelease, forcing the developer to use greater care when directly managing memory.

**Parameters**

***aVariable*** The variable which will have its retain count reduced.

**See also**

DSVariableRetain
DSVariableFree

#### 7.14.2.8 DSVariable∗ DSVariableRetain (DSVariable ∗ *aVariable*)

Function to increase variable retain count by one.

Variables utilize a similar memory management system used in Objective-C NSObject subclasses, where objects a retained/released.

**Parameters**

***aVariable*** The variable which will have its retain count increased.

**Returns**

The same variable which received the retain count increase is returned, for convinience.

**See also**

DSVariableRelease

# Index