Design Space Toolbox

2.0.1

Generated by Doxygen 1.6.3

Wed Sep 14 10:54:02 2011

Contents

Chapter 1

Todo List

File DSErrors.c Implement locks when making the error strings.

File DSIO.h Define standard input and output file formats.

Define criteria for warnings, errors and fatal errors.

File DSStd.h Add all previous functionality.

Add vertex enumeration functionality.

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_CASE_NULL** M_DS_NULL ": Case is NULL"
- #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.

- #define M_DS_VAR_NULL M_DS_NULL ": Variable Pool is NULL"

  Error message indicating a NULL variable pool.

- #define M_DS_VAR_LOCKED " DSVariablePool: Insufficient priviliges"

  Error message indicating insufficient priviliges to manipulate a variable pool.

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 DSCase related errors is M_DS_CASE_NULL.

Messages for DSMatrix related errors are M_DS_MAT_NULL, M_DS_MAT_OUTOFBOUNDS and M_DS_MAT_NOINTERNAL.

Messages for DSVariable related errors are M_DS_VAR_NULL and M_DS_VAR_LOCKED.

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.
- #define A_DS_KILLNOW A_DS_FATAL
  DEPRECATED:

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 DSGMAACCESSORS

Internal GMA Accessor macros.

Defines

- #define **DSGMAXi**(x) ((x)->Xi)
- #define **DSGMAXd**(x) ((x)->Xd)
- #define **DSGMAAlpha**(x) ((x)->alpha)
- #define **DSGMABeta**(x) ((x)->beta)
- #define **DSGMAGd**(x) ((x)->Gd)
- #define **DSGMAGi**(x) ((x)->Gi)
- #define **DSGMAHd**(x) ((x)->Hd)
- #define **DSGMAHi**(x) ((x)->Hi)
- #define **DSGMASignature**(x) ((x)->signature)

5.3.1 Detailed Description

Internal GMA Accessor macros. Used within DSGMASystem.c to access the data within a GMA data type. These macros are not to be used putside of this file, as they do not check the data dor consistency and thus would not invoke the DSError function, making it harder to trace errors.

5.4 Options for JSON conversion of DSCase object.

Defines

- #define DS_CASE_JSON_NO_SSYSTEM 1

  Flag value indicating that the S-System information should not be included in the JSON string.

- #define DS_CASE_JSON_NO_CASE_SIGNATURE 2

  Flag value indicating that the case signature should not be included in the JSON string.

- #define DS_CASE_JSON_NO_CONDITIONS 4

  Flag value indicating that the conditions for validity should not be included in the JSON string.

5.4.1 Detailed Description

Defined here are different options determining the information stored in a JSON string for a DSCase object. These options are passed to the DSIOSetCaseJSONOptions function. These options designate the value for a global flag variable

5.5 Options for JSON conversion of DSSSystem object.

Defines

- #define DS_SSYSTEM_JSON_NO_SOLUTION 1

  Flag value indicating that the S-System solution should not be included in the JSON string.

- #define DS_SSYSTEM_JSON_NO_SINGULAR 2

  Flag value indicating that the JSON string will not indicate if the S-System is singular.

5.5.1 Detailed Description

Defined here are different options determining the information stored in a JSON string for a DSSSystem object. These options are passed to the DSIOSetSSystemJSONOptions function. These options designate the value for a global flag variable.

5.6 DSSSysACCESSORS

Internal S-System Accessor macros.

Defines

- #define **DSSSysXi**(x) ((x)->Xi)
- #define **DSSSysXd**(x) ((x)->Xd)
- #define **DSSSysAlpha**(x) ((x)->alpha)
- #define **DSSSysBeta**(x) ((x)->beta)
- #define **DSSSysGd**(x) ((x)->Gd)
- #define **DSSSysGi**(x) ((x)->Gi)
- #define **DSSSysHd**(x) ((x)->Hd)
- #define **DSSSysHi**(x) ((x)->Hi)
- #define **DSSSysM**(x) ((x)->M)
- #define **DSSSysIsSingular**(x) ((x)->isSingular)
- #define **DSSSysShouldFreeXd**(x) ((x)->shouldFreeXd)
- #define **DSSSysShouldFreeXi**(x) ((x)->shouldFreeXi)

5.6.1 Detailed Description

Internal S-System Accessor macros. Used within DSSSystem.c to access the data within a S-System data type. These macros are not to be used putside of this file, as they do not check the data dor consistency and thus would not invoke the DSError function, making it harder to trace errors.

5.7 Macros to manipulate variables.

Defines

- #define DSVariableSetValue(x, y) ((x)->value = (y))

  Macro to set the value of a variable data structure.

- #define DSVariableValue(x) (((x) != NULL) ? (x)->value : NAN)

  Macro to get the value of a variable data structure.

- #define DSVariableName(x) ((x)->name)

  Macro to get the value of a variable data structure.

5.7.1 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.

5.7.2 Define Documentation

5.7.2.1 #define DSVariableName(x) ((x)->name)

Macro to get the value of a variable data structure.

This macro provides a consistent way for retrieving the value of a variable, despite the internal structure of the data type.

5.7.2.2 #define DSVariableSetValue(x, y) ((x)->value = (y))

Macro to set the value of a variable data structure.

This macro provides a consistent way for changing the value of a variable, despite the internal structure of the data type. This macro is expanded to a simple assignment.

5.7.2.3 #define DSVariableValue(x) (((x) != NULL) ? (x)->value : NAN)

Macro to get the value of a variable data structure.

This macro provides a consistent way for retrieving the value of a variable, despite the internal structure of the data type.

Chapter 6

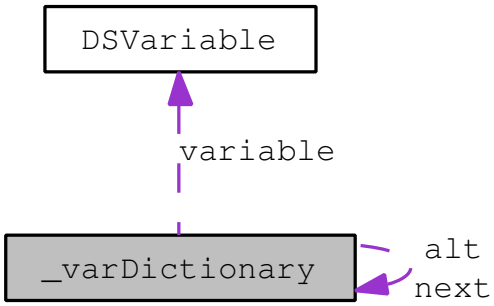
Data Structure Documentation

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.h
DSVariable.c

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

- DSTypes.h

6.2 base_info Union Reference

Data Fields

- char * name

  The string representing the name of the variable.

- double value

  The variable representing the value of a constant.

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

- DSGMASystemParsingAux.h

6.3 ds_parallelstack_t Struct Reference

Stack object used by the worker threads.

`#include <DSDesignSpaceParallel.h>`

Data Fields

- DSUInteger * base

  The pointer to the array of DSUIntegers storing the case numbers.

- DSUInteger * current

  A pointer to the top of the stack.

- DSUInteger count

  The number of elements in the stack.

- DSUInteger size

  The current size of the base array.

- pthread_mutex_t pushpop

  The mutex used when pushing and popping data from the stack.

6.3.1 Detailed Description

Stack object used by the worker threads. This structure is a stack of case numbers indicating the DSCases that need to be processed, and each pthread_t used for processing cases and determining validity (currently disabled due to the non re-entrant GLPK) must have access to a ds_parallelstack_t.

Note

One stack should be created per thread, to avoid one thread blocking another during popping and pushing operations. A single stack could be used, as the parallel stacks are thread safe, and under some conditions might be more efficient as all the threads in the thread pool will remain active until all cases have been processed. Currently, the number of cases to be processed by a thread are determined prior to launching the threads, and each thread has an equal number of cases to process. If a thread has many invalid cases, it may finish all of its cases before the other threads, and thus it is possible for the system to make less use of multiple processors. To avoid this situation, more threads than processors can be used or a single shared stack could be used.

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

- DSDesignSpaceParallel.h

6.4 DSCase Struct Reference

Data type used to represent a case.

`#include <DSTypes.h>`

Collaboration diagram for DSCase:

Data Fields

- const DSVariablePool * Xd

  A pointer to the DSVariablePool with the dependent variables.

- const DSVariablePool * Xi

  A pointer to the DSVariablePool with the independent variables.

- DSSSystem * ssys

  The DSSSystem of the case.

- DSMatrix * Cd

  The condition matrix corresponding to the dependent variables.

- DSMatrix * Ci

  The condition matrix corresponding to the independent variables.

- DSMatrix * U

  The boundary matrix corresponding to the independent variables.

- DSMatrix * delta

  The condition matrix corresponding to the constants.

- DSMatrix * zeta

  The boundary matrix corresponding to the constants.

- DSUInteger caseNumber

  The case number used to identify the case.

- DSUInteger * signature

  The case signature indicating the dominant terms used to generate the case.

6.4.1 Detailed Description

Data type used to represent a case. This data type has all the necessary information for a case in design space. It a pointer to the dependent and independent variables of the system, a pointer to the corresponding S-System, the Condition matrices and boundary matrices. It also has information about the case number and case signature.

Note

The case number is arbitrary, and can be generated by two algorithms to be either big endian or small endian. For compatibility with the current design space toolbox, big endian is the default.

The case is not responsible for freeing the Xd and Xi variables. If the case is generated from a design space, then the design space is responsible for freeing the Xi and Xd variable pools; otherwise the internal S-System is responsible for freeing this data.

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

- DSTypes.h

6.5 DSDesignSpace Struct Reference

Data type used to represent a design space/.

`#include <DSTypes.h>`

Collaboration diagram for DSDesignSpace:

Data Fields

- DSGMASystem * gma

  The gma system of the design space.

- const DSVariablePool * Xd

  A pointer to the DSVariablePool with the dependent variables.

- const DSVariablePool * Xi

  A pointer to the DSVariablePool with the dependent variables.

- DSCase ** cases

  The array of all the cases in the design space.

- bool * validCases

  The array of case validity.

- DSUInteger numberOfCases

  DSUInteger indicating the maximum number of cases in the design space.

6.5.1 Detailed Description

Data type used to represent a design space/. The design space data structure is a convenience structure that automates the construction and analysis of cases, and manages the memory associated with these cases. This behavior can be avoided by working directly with the gma system of the designspace.

See also

DSDesignSpace.h
DSDesignSpace.c

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

- DSTypes.h

6.6 dsexpression Struct Reference

Data type representing mathematical expressions.

`#include <DSTypes.h>`

Collaboration diagram for dsexpression:

Data Fields

- union {
  char **op_code**
  double **constant**
  char * variable
  A string with the name of the variable.
  } node

  Union of data types potentially contained in the node.

- int type

  Integer specifying the type of node.

- int numberOfBranches

  Number of branches of children, relevant to operators and functions.

- struct dsexpression ** branches

  Array of expression nodes with children nodes.

6.6.1 Detailed Description

Data type representing mathematical expressions. This data type is the internal representation of matematical expressions. This data type is an Abstracts Syntax Tree with only three operators: '+', '*' and '^'. All other operators ('-' and '/') are represented by a combination of the former operators. The DSExpression automatically groups constant values, and reserves the first branch of the multiplication and addition operator for constant values. These operators can have any number of branches. The '^' operator can have two, and only two, branches.

Note

Functions are handled as variables with a single argument

See also

DSExpression.h
DSExpression.c

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

- DSTypes.h

6.7 DSGMASystem Struct Reference

Data type representing a GMA-System.

`#include <DSTypes.h>`

Collaboration diagram for DSGMASystem:

Data Fields

- char ** **equations**
- DSMatrix * **alpha**
- DSMatrix * **beta**
- DSMatrixArray * **Gd**
- DSMatrixArray * **Gi**
- DSMatrixArray * **Hd**
- DSMatrixArray * **Hi**
- DSVariablePool * **Xd**
- DSVariablePool * **Xi**
- DSUInteger * **signature**

6.7.1 Detailed Description

Data type representing a GMA-System. This data structure is a standard representation of an GMA 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 GMA system uses an array of matrices to represent all the terms in all of the equations.

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

- DSTypes.h

6.8 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.

6.8.1 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.

See also

DSMatrix.h
DSMatrix.c

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

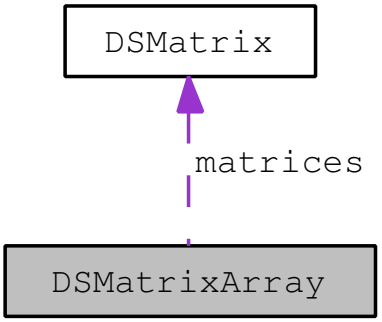
- DSTypes.h

6.9 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.

6.9.1 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 is used to represent three-dimensional matrices, as used internally by GMA's systems.

See also

DSMatrixArray.h
DSMatrixArray.c

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

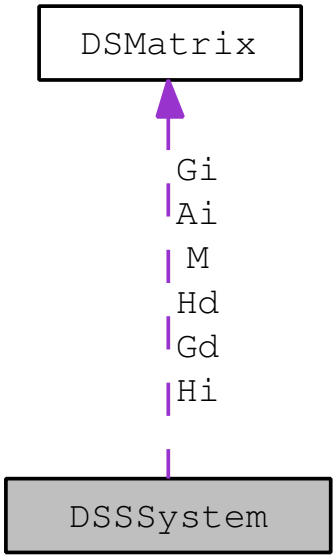
- DSTypes.h

6.10 DSSSystem Struct Reference

Data type representing an S-System.

`#include <DSTypes.h>`

Collaboration diagram for DSSSystem:



Data Fields

- DSMatrix * **alpha**
- DSMatrix * **beta**
- DSMatrix * **Gd**
- DSMatrix * **Gi**
- DSMatrix * **Hd**
- DSMatrix * **Hi**
- DSMatrix * **M**
- DSVariablePool * **Xd**
- DSVariablePool * **Xi**
- bool **isSingular**
- bool **shouldFreeXd**
- bool **shouldFreeXi**

6.10.1 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

6.11 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.

6.11.1 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.

See also

DSVariable.h
DSVariable.c

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

- DSTypes.h

6.12 DSVariablePool Struct Reference

User-level variable pool.

`#include <DSTypes.h>`

Collaboration diagram for DSVariablePool:

Data Fields

- struct _varDictionary * root

  The root of the internal dictionary.

- DSUInteger numberOfVariables

  Number of variables in the pool.

- DSVariable ** variables

  A C array with the variables stored.

- DSVariablePoolLock lock

  Indicates if the variable pool is read-only.

6.12.1 Detailed Description

User-level variable pool. This data type keeps an internal dictionary structure of type struct _varDictionary to keep track of all the variables associated with a variable pool. This data type also records the number of variables in the dictionary and the order with which they were added.

See also

struct _varDictionary
DSVariable.h
DSVariable.c

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

- DSTypes.h

6.13 DSVertices Struct Reference

Data type that contains vertices of an N-Dimensional object.

`#include <DSTypes.h>`

Data Fields

- double ** **vertices**
- DSUInteger **dimensions**
- DSUInteger **numberOfVertices**

6.13.1 Detailed Description

Data type that contains vertices of an N-Dimensional object. This data type is used ofr determining the region of validity of a case in design space. If the vertices represent a polygon, they can be orderd according to their clockwise position, starting by the right-most vertex in a XY plane.

See also

DSVertices.h
DSVertices.c

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

- DSTypes.h

6.14 expression_token Struct Reference

A data structure representing a token used when parsing strings for variable pools.

`#include <DSExpressionTokenizer.h>`

Collaboration diagram for expression_token:

Data Fields

- int type

  The current token code.

- union {
  char * name
  Used for storing the name of a variable.
  double value
  Used for storing the value of a constant.
  } data

  Union for holding either the name of a variable, or the value of a constant.

- struct expression_token * next

  A pointer to the next token in the list.

6.14.1 Detailed Description

A data structure representing a token used when parsing strings for variable pools. This structures follows the convention used with the struct variable_token and struct matrix_token, representing an ordered list of tokens, as found by the tokenizers generated by the lex program.

See also

DSExpressionTokenizer()

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

- DSExpressionTokenizer.h

6.15 matrix_token Struct Reference

Collaboration diagram for matrix_token:

Data Fields

- int **token**
- double **value**
- DSUInteger **row**
- DSUInteger **column**
- struct matrix_token * **next**

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

- DSMatrixTokenizer.h

6.16 parse_expression_s Struct Reference

Structure used when parsing a mathematical expression.

`#include <DSExpressionTokenizer.h>`

Collaboration diagram for parse_expression_s:

Data Fields

- DSExpression * root

  The pointer to the DSExpression representing the root of the syntax tree.

- bool wasSuccesful

  Indicates if the parsing was succesful.

6.16.1 Detailed Description

Structure used when parsing a mathematical expression. This structure is used to parse a mathematical expression, it holds (1) the root of the abstract syntax tree and a flag indicating if any syntax errors were found.

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

- DSExpressionTokenizer.h

6.17 parser_aux Struct Reference

Data type used to parse strings to GMA System.

`#include <DSGMASystemParsingAux.h>`

Collaboration diagram for parser_aux:

Data Structures

- union base_info

Data Fields

- char sign

  The sign of the term represented by the current node.

- union parser_aux::base_info * bases

  Dynamically allocated array of bases, can be either variables or constants.

- bool succeded

  A flag indicating if the parsing of the expression was succesful.

- double * exponents

  A dynamically allocated array of exponents, must be constants.

- DSUInteger numberOfBases

  The number of base-exponents pairs in the term.

- struct parser_aux * next

  A pointer to the next node, representing the next term in the equation.

6.17.1 Detailed Description

Data type used to parse strings to GMA System. This data structure forms an organized list of terms, each with base exponent pairs that are then used to create the system matrices. This data structure is key for the parsing of GMA systems. Each node in the gma_parseraux_t list represent a term in an expression in the order it was found, and each node points to the next term. Each expression, or equation, has it's own list of terms. If a base is a constant, then it should not have an exponent, and hence it's exponent is assigned a NAN value and this is used to indicate that the base is a constant.

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

- DSGMASystemParsingAux.h

6.18 pthread_struct Struct Reference

Data structure passed to a pthread.

`#include <DSDesignSpaceParallel.h>`

Collaboration diagram for pthread_struct:

Data Fields

- ds_parallelstack_t * **stack**
- DSDesignSpace * **ds**
- FILE * **file**

6.18.1 Detailed Description

Data structure passed to a pthread. This data structure has two fields, one is a pointer to a ds_parallelstack_t object; this stack containes a stack of case numbers to be processed in parallel. Each stack is not designed to be accessed concurrently, but should still be thread safe.

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

- DSDesignSpaceParallel.h

6.19 v_token_data Union Reference

Union containing the alternative values a struct variable_token can take.

`#include <DSVariableTokenizer.h>`

Data Fields

- char * **name**
- double **value**

6.19.1 Detailed Description

Union containing the alternative values a struct variable_token can take. The union can have either a string, used for the names of variables when an identifier is found; and a double value used when a value is found.

See also

struct variable_token

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

- DSVariableTokenizer.h

6.20 variable_token Struct Reference

A data structure representing a token used when parsing strings for variable pools.

`#include <DSVariableTokenizer.h>`

Collaboration diagram for variable_token:

Data Fields

- int **type**
- union v_token_data **data**
- struct variable_token * **next**

6.20.1 Detailed Description

A data structure representing a token used when parsing strings for variable pools.

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

- DSVariableTokenizer.h

6.21 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**

6.21.1 Field Documentation

6.21.1.1 int yy_bs_column

The column count.

6.21.1.2 int yy_bs_lineno

The line count.

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

- DSExpressionTokenizerLex.c
- DSMatrixTokenizerLex.c
- DSVariableTokenizerLex.c

6.22 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 files:

- DSExpressionTokenizerLex.c
- DSMatrixTokenizerLex.c
- DSVariableTokenizerLex.c

6.23 yyguts_t Struct Reference

Collaboration diagram for yyguts_t:

Data Fields

- YY_EXTRA_TYPE **yyextra_r**
- FILE * **yyin_r**
- FILE * **yyout_r**
- size_t yy_buffer_stack_top
- size_t yy_buffer_stack_max
- YY_BUFFER_STATE * yy_buffer_stack
- char **yy_hold_char**
- yy_size_t **yy_n_chars**
- yy_size_t **yyleng_r**
- char * **yy_c_buf_p**
- int **yy_init**
- int **yy_start**
- int **yy_did_buffer_switch_on_eof**
- int **yy_start_stack_ptr**
- int **yy_start_stack_depth**
- int * **yy_start_stack**
- yy_state_type **yy_last_accepting_state**
- char * **yy_last_accepting_cpos**
- int **yylineno_r**
- int **yy_flex_debug_r**
- char * **yytext_r**
- int **yy_more_flag**
- int **yy_more_len**

6.23.1 Field Documentation

6.23.1.1 YY_BUFFER_STATE * yy_buffer_stack

Stack as an array.

6.23.1.2 size_t yy_buffer_stack_max

capacity of stack.

6.23.1.3 size_t yy_buffer_stack_top

index of top of stack.

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

- DSExpressionTokenizerLex.c
- DSMatrixTokenizerLex.c
- DSVariableTokenizerLex.c

6.24 YYMINORTYPE Union Reference

Data Fields

- int **yyinit**
- DSExpressionParserTOKENTYPE **yy0**
- DSGMASystemParserTOKENTYPE **yy0**
- DSSSystemParserTOKENTYPE **yy0**
- DSVariablePoolParserTOKENTYPE **yy0**

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

- DSExpressionGrammar.c
- DSGMASystemGrammar.c
- DSSSystemGrammar.c
- DSVariableGrammar.c

6.25 yyParser Struct Reference

Collaboration diagram for yyParser:

Data Fields

- int **yyidx**
- int **yyerrcnt**
- DSExpressionParserARG_SDECL yyStackEntry **yystack** [YYSTACKDEPTH]
- DSGMASystemParserARG_SDECL int **yystksz**
- yyStackEntry * **yystack**
- DSSSystemParserARG_SDECL int **yystksz**
- DSVariablePoolParserARG_SDECL int **yystksz**
- ParseARG_SDECL int **yystksz**

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

- DSExpressionGrammar.c
- DSGMASystemGrammar.c
- DSSSystemGrammar.c
- DSVariableGrammar.c
- lempar.c

6.26 yyStackEntry Struct Reference

Collaboration diagram for yyStackEntry:

Data Fields

- YYACTIONTYPE **stateno**
- YYCODETYPE **major**
- YYMINORTYPE **minor**

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

- DSExpressionGrammar.c
- DSGMASystemGrammar.c
- DSSSystemGrammar.c
- DSVariableGrammar.c
- lempar.c

Chapter 7

File Documentation

7.1 DSDesignSpace.c File Reference

Implementation file with functions for dealing with Design Spaces.

```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <glpk.h>
#include "DSMemoryManager.h"
#include "DSDesignSpace.h"
#include "DSMatrix.h"
#include "DSGMASystem.h"
#include "DSSSystem.h"
#include "DSCase.h"
#include "DSDesignSpaceParallel.h"
#include "DSTypes.h"
```

Include dependency graph for DSDesignSpace.c:This graph shows which files directly or indirectly include this file:

Defines

- #define **__DS_MAC_OS_X__**
- #define **DS_PARALLEL_DEFAULT_THREADS** 3
- #define **DSDSGMA**(x) ((x)->gma)
- #define **DSDSCases**(x) ((x)->cases)
- #define **DSDSNumCases**(x) ((x)->numberOfCases)
- #define **DSDSValid**(x) ((x)->validCases)
- #define **DSDSXd**(x) ((x)->Xd)
- #define **DSDSXi**(x) ((x)->Xi)

Functions

- DSDesignSpace * **DSDesignSpaceAlloc** (void)
- void **DSDesignSpaceFree** (DSDesignSpace *ds)
- DSDesignSpace * **DSDesignSpaceByParsingStringList** (const DSVariablePool *const Xd, const char *const string,...)
- DSDesignSpace * **DSDesignSpaceByParsingStrings** (const DSVariablePool *const Xd, char *const *const strings, const DSUInteger numberOfEquations)
- void **DSDesignSpaceSetGMA** (DSDesignSpace *ds, DSGMASystem *gma)
- const DSUInteger **DSDesignSpaceNumberOfEquations** (const DSDesignSpace *ds)
- DSExpression ** **DSDesignSpaceEquations** (const DSDesignSpace *ds)
- const DSUInteger **DSDesignSpaceNumberOfCases** (const DSDesignSpace *ds)
- const DSUInteger * **DSDesignSpaceSignature** (const DSDesignSpace *ds)
- const DSCase * **DSDesignSpaceCaseWithCaseNumber** (const DSDesignSpace *ds, const DSUInteger caseNumber)
- const DSCase * **DSDesignSpaceCaseWithCaseSignature** (const DSDesignSpace *ds, const DSUInteger *signature)
- const DSCase * **DSDesignSpaceCaseWithCaseSignatureList** (const DSDesignSpace *ds, const DSUInteger firstTerm,...)
- const bool **DSDesignSpaceCaseWithCaseNumberIsValid** (const DSDesignSpace *ds, const DSUInteger caseNumber)
- const bool **DSDesignSpaceCaseWithCaseSignatureIsValid** (const DSDesignSpace *ds, const DSUInteger *signature)
- const bool **DSDesignSpaceCaseWithCaseSignatureListIsValid** (const DSDesignSpace *ds, const DSUInteger firstTerm,...)
- const DSGMASystem * **DSDesignSpaceGMASystem** (const DSDesignSpace *ds)
- void **DSDesignSpaceCalculateAllCases** (DSDesignSpace *ds)
- void **DSDesignSpaceCalculateAllCasesAndSaveToFile** (DSDesignSpace *ds, const char *path, const bool overwrite)
- void **DSDesignSpaceCalculateValidityOfCases** (DSDesignSpace *ds)
- void **DSDesignSpacePrint** (const DSDesignSpace *ds)

7.1.1 Detailed Description

Implementation file with functions for dealing with Design Spaces. 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

7.2 DSDesignSpace.h File Reference

Header file with functions for dealing with Design Spaces.

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

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

Include dependency graph for DSDesignSpace.h:This graph shows which files directly or indirectly include this file:

Defines

- #define **M_DS_DESIGN_SPACE_NULL** M_DS_NULL ": Design Space is NULL"

Functions

- DSDesignSpace * **DSDesignSpaceAlloc** (void)
- void **DSDesignSpaceFree** (DSDesignSpace *ds)
- DSDesignSpace * **DSDesignSpaceByParsingStringList** (const DSVariablePool *const Xd, const char *const string,...)
- DSDesignSpace * **DSDesignSpaceByParsingStrings** (const DSVariablePool *const Xd, char *const *const strings, const DSUInteger numberOfEquations)
- void **DSDesignSpaceSetGMA** (DSDesignSpace *ds, DSGMASystem *gma)
- const DSUInteger **DSDesignSpaceNumberOfEquations** (const DSDesignSpace *ds)
- DSExpression ** **DSDesignSpaceEquations** (const DSDesignSpace *ds)
- const DSUInteger **DSDesignSpaceNumberOfCases** (const DSDesignSpace *ds)
- const DSUInteger * **DSDesignSpaceSignature** (const DSDesignSpace *ds)
- const DSCase * **DSDesignSpaceCaseWithCaseNumber** (const DSDesignSpace *ds, const DSUInteger caseNumber)
- const DSCase * **DSDesignSpaceCaseWithCaseSignature** (const DSDesignSpace *ds, const DSUInteger *signature)
- const DSCase * **DSDesignSpaceCaseWithCaseSignatureList** (const DSDesignSpace *ds, const DSUInteger firstTerm,...)
- const bool **DSDesignSpaceCaseWithCaseNumberIsValid** (const DSDesignSpace *ds, const DSUInteger caseNumber)
- const bool **DSDesignSpaceCaseWithCaseSignatureIsValid** (const DSDesignSpace *ds, const DSUInteger *signature)
- const bool **DSDesignSpaceCaseWithCaseSignatureListIsValid** (const DSDesignSpace *ds, const DSUInteger firstTerm,...)
- const DSGMASystem * **DSDesignSpaceGMASystem** (const DSDesignSpace *ds)
- void **DSDesignSpaceCalculateAllCases** (DSDesignSpace *ds)
- void **DSDesignSpaceCalculateAllCasesAndSaveToFile** (DSDesignSpace *ds, const char *path, const bool overwrite)
- void **DSDesignSpaceCalculateValidityOfCases** (DSDesignSpace *ds)
- void **DSDesignSpacePrint** (const DSDesignSpace *ds)

7.2.1 Detailed Description

Header file with functions for dealing with Design Spaces. 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 DSDesignSpaceParallel.c File Reference

Implementation file with functions for dealing with parallel operatirons used by the design spaces.

```
#include <stdio.h>
#include <pthread.h>
#include <glpk.h>
#include "DSDesignSpaceParallel.h"
#include "DSErrors.h"
#include "DSMemoryManager.h"
#include "DSDesignSpace.h"
#include "DSGMASystem.h"
#include "DSSSystem.h"
#include "DSCase.h"
#include "DSMatrix.h"
#include <unistd.h>
```

Include dependency graph for DSDesignSpaceParallel.c:

Defines

- #define **PARALLEL_STACK_SIZE_INCREMENT** 5000

Functions

- void **DSParallelInitMutexes** (void)
- ds_parallelstack_t * **DSParallelStackAlloc** (void)
- void **DSParallelStackFree** (ds_parallelstack_t *stack)
- void **DSParallelStackPush** (ds_parallelstack_t *stack, const DSUInteger integer)
- const DSUInteger **DSParallelStackPop** (ds_parallelstack_t *stack)
- void * **DSParallelWorker** (void *pthread_struct)
- void * **DSParallelWorkerCases** (void *pthread_struct)
- void * DSParallelWorkerCasesSaveToDisk (void *pthread_struct)
- void * DSParallelWorkerValidity (void *pthread_struct)

Variables

- pthread_mutex_t **workeradd**
- pthread_mutex_t **iomutex**

7.3.1 Detailed Description

Implementation file with functions for dealing with parallel operatirons used by the design spaces. Copyright (C) 2011 Jason Lomnitz.

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

Author

Jason Lomnitz.

Date

2011

7.3.2 Function Documentation

7.3.2.1 void* DSParallelWorkerCasesSaveToDisk (void * *pthread_struct*)

Data in stack MUST be term signature if not an error will occur

7.3.2.2 void* DSParallelWorkerValidity (void * *pthread_struct*)

Data in stack MUST be a case number, if not an error will occur

7.4 DSDesignSpaceParallel.h File Reference

Header file with functions for dealing with parallel operatirons used by the design spaces.

```
#include <pthread.h>
```

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

Include dependency graph for DSDesignSpaceParallel.h:This graph shows which files directly or indirectly include this file:

Data Structures

- struct ds_parallelstack_t

  Stack object used by the worker threads.

- struct pthread_struct

  Data structure passed to a pthread.

Functions

- void **DSParallelInitMutexes** (void)
- ds_parallelstack_t * **DSParallelStackAlloc** (void)
- void **DSParallelStackFree** (ds_parallelstack_t *stack)
- void **DSParallelStackPush** (ds_parallelstack_t *stack, const DSUInteger number)
- const DSUInteger **DSParallelStackPop** (ds_parallelstack_t *stack)
- void * **DSParallelWorkerCases** (void *pthread_struct)
- void * DSParallelWorkerCasesSaveToDisk (void *pthread_struct)
- void * DSParallelWorkerValidity (void *pthread_struct)

7.4.1 Detailed Description

Header file with functions for dealing with parallel operatirons used by the design spaces. 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

7.4.2 Function Documentation

7.4.2.1 void* DSParallelWorkerCasesSaveToDisk (void * *pthread_struct*)

Data in stack MUST be term signature if not an error will occur

7.4.2.2 void* DSParallelWorkerValidity (void * *pthread_struct*)

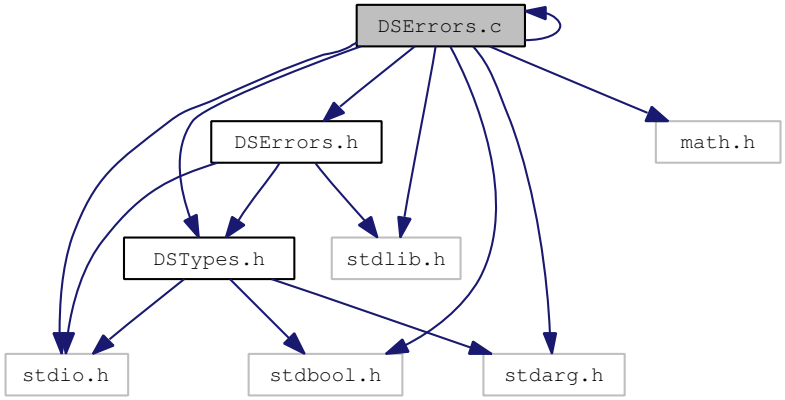
Data in stack MUST be a case number, if not an error will occur

7.5 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 "DSMemoryManager.h"
```

Include dependency graph for DSErrors.c:



Defines

- #define STACK_TRACE_NUM 10

  Maximum number of traces on the call stack.

- #define MSIZE 1500

  The maximum size of the error message string.

Functions

- 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.

7.5.1 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 report the errors

throughout the design space library. The DSErrorFunction allows different behaviors; the default behavior, errors are printed to the DSIOErrorFile, which is set to stderr by default. This behavior can be changed by setting changing DSPostWarning, DSPostError and DSPostFatalError function pointers.

See also

DSIOErrorFile
DSPostWarning
DSPostError
DSPostFatalError

Author

Jason Lomnitz.

Date

2011

Todo

Implement locks when making the error strings.

7.5.2 Define Documentation

7.5.2.1 #define MSIZE 1500

The maximum size of the error message string.

This represents the maximum number of characters that an error string can contain. The error string is a statically allocated string.

7.5.2.2 #define STACK_TRACE_NUM 10

Maximum number of traces on the call stack.

This number represents the maximum number of traces on the call stack that the DSError function adds to the error string. The trace represents all the functions called up to the error.

7.5.3 Function Documentation

7.5.3.1 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, LINE and FUNC arguments, to report the error/warning at the appropriate file, line and function.

Parameters

M_DS_Message A string containing the error message.

A_DS_ACTION A character representing an error code as described in A_DS_Actions.

FILEN A string with the name of the file where the error was reported.

LINE An integer with the line number in the file where the error was reported.

FUNC A string with the name of the function where the error was reported.

See also

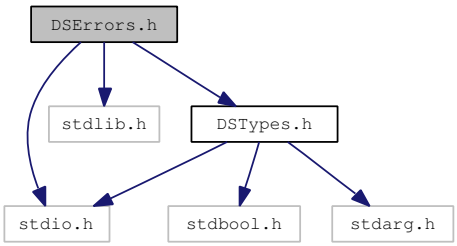
DSError
Actions for DS Errors.

7.6 DSErrors.h File Reference

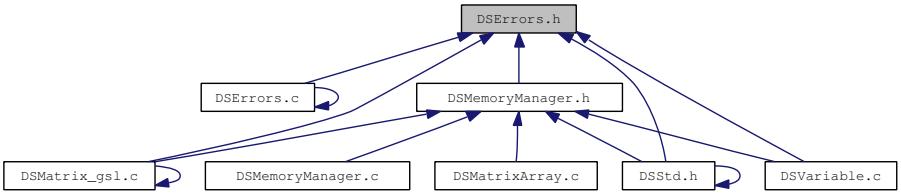
Header file with functions for error and exception handling.

```
#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 M_DS_PARSE "Could not parse data"
  Message for an error during parsing.
- #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

- 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.

7.6.1 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.

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

7.6.2 Define Documentation

7.6.2.1 #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.

7.6.3 Function Documentation

7.6.3.1 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, LINE and FUNC arguments, to report the error/warning at the appropriate file, line and function.

Parameters

M_DS_Message A string containing the error message.

A_DS_ACTION A character representing an error code as described in A_DS_Actions.

FILEN A string with the name of the file where the error was reported.

LINE An integer with the line number in the file where the error was reported.

FUNC A string with the name of the function where the error was reported.

See also

DSError
Actions for DS Errors.

7.7 DSExpression.c File Reference

Implementation file with functions for dealing with mathematical expressions.

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "DSExpression.h"
#include "DSErrors.h"
#include "DSMemoryManager.h"
#include "DSExpressionTokenizer.h"
#include "DSTypes.h"
```

Include dependency graph for DSExpression.c:This graph shows which files directly or indirectly include this file:

Defines

- #define **DS_EXPRESSION_CONSTANT_BRANCH** 0
- #define **DS_EXPRESSION_STRING_INIT_LENGTH** 1000

Functions

- DSExpression * **DSExpressionAllocWithOperator** (const char op_code)
- DSExpression * **DSExpressionAllocWithConstant** (const double value)
- DSExpression * **DSExpressionAllocWithVariableName** (const char *name)
- void **DSExpressionFree** (DSExpression *root)
- DSExpression * **DSExpressionByParsingString** (const char *string)
- void **DSExpressionAddBranch** (DSExpression *expression, DSExpression *branch)
- char * **DSExpressionAsString** (const DSExpression *expression)
- void **DSExpressionPrint** (const DSExpression *expression)

7.7.1 Detailed Description

Implementation file with functions for dealing with mathematical expressions. 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

7.8 DSExpression.h File Reference

Header file with functions for dealing with mathematical expressions.

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

Include dependency graph for DSExpression.h:This graph shows which files directly or indirectly include this file:

Defines

- #define **DS_EXPRESSION_TYPE_UNDEFINED** 0
- #define **DS_EXPRESSION_TYPE_OPERATOR** 1
- #define **DS_EXPRESSION_TYPE_CONSTANT** 2
- #define **DS_EXPRESSION_TYPE_VARIABLE** 3
- #define **DS_EXPRESSION_TYPE_FUNCTION** 4
- #define **DSExpressionSetOperator**(x, y) ((x->node.op_code) = y, (x->type = DS_EXPRESSION_TYPE_OPERATOR))
- #define **DSExpressionSetVariable**(x, y) ((x->node.variable) = y, (x->type = DS_EXPRESSION_TYPE_VARIABLE))
- #define **DSExpressionSetConstant**(x, y) ((x->node.constant) = y, (x->type = DS_EXPRESSION_TYPE_CONSTANT))
- #define **DSExpressionType**(x) (x->type)
- #define **DSExpressionNumberOfBranches**(x) (x->numberOfBranches)
- #define **DSExpressionBranchAtIndex**(x, y) ((y < DSExpressionNumberOfBranches(x)) ? x->branches[y] : NULL)
- #define **DSExpressionOperator**(x) ((x->type == DS_EXPRESSION_TYPE_OPERATOR) ? x->node.op_code : '?')
- #define **DSExpressionVariable**(x) ((x->type == DS_EXPRESSION_TYPE_VARIABLE || x->type == DS_EXPRESSION_TYPE_FUNCTION) ? x->node.variable : NULL)
- #define **DSExpressionConstant**(x) ((x->type == DS_EXPRESSION_TYPE_CONSTANT) ? x->node.constant : NAN)

Functions

- DSExpression * **DSExpressionAllocWithOperator** (const char op_code)
- DSExpression * **DSExpressionAllocWithConstant** (const double value)
- DSExpression * **DSExpressionAllocWithVariableName** (const char *name)
- void **DSExpressionFree** (DSExpression *root)
- DSExpression * **DSExpressionByParsingString** (const char *string)
- char * **DSExpressionAsString** (const DSExpression *expression)
- void **DSExpressionPrint** (const DSExpression *expression)

7.8.1 Detailed Description

Header file with functions for dealing with mathematical expressions. The mathematical expressions are converted into a form similar to the model used in MUPAD. Internally, only three operators are used: '+', '*' and '^'. The '-' operator is converted, such that $A-B$ would actually be $A+B*(-1)$ and the '/' operator is converted such that A/B would actually be $A*B^-1$. The '*' and '+' operators must have at least two branches, but may have any number of branches. The first branch for these operators is reserved for constant values, such that a+b is actually 0+a+b, and a*b is actually 1*a*b. This canonical form is used

to speed up the processing of mathematical expressions when converting them to matrices for the GMA and SSystem. The '^' must have only two branches.

Author

Jason Lomnitz.

Date

2011

7.9 DSExpressionTokenizerLex.c File Reference

Implementation file with functions for tokenizing matrices, generated by flex.

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include "DSTypes.h"
#include "DSMemoryManager.h"
#include "DSExpression.h"
#include "DSExpressionTokenizer.h"
#include <unistd.h>
```

Include dependency graph for DSExpressionTokenizerLex.c:

Data Structures

- struct yy_buffer_state
- struct yy_trans_info
- struct yyguts_t

Defines

- #define **YY_INT_ALIGNED** short int
- #define **FLEX_SCANNER**
- #define **YY_FLEX_MAJOR_VERSION** 2
- #define **YY_FLEX_MINOR_VERSION** 5
- #define **YY_FLEX_SUBMINOR_VERSION** 35
- #define **FLEX_BETA**
- #define **INT16_MIN** (-32767-1)
- #define **INT32_MIN** (-2147483647-1)
- #define **INT8_MAX** (127)
- #define **INT16_MAX** (32767)
- #define **INT32_MAX** (2147483647)
- #define **UINT8_MAX** (255U)
- #define **UINT16_MAX** (65535U)
- #define **UINT32_MAX** (4294967295U)
- #define **yyconst**
- #define **YY_NULL** 0
- #define **YY_SC_TO_UI**(c) ((unsigned int) (unsigned char) c)
- #define **YY_TYPEDEF_YY_SCANNER_T**
- #define **yyin** yyg->yyin_r
- #define **yyout** yyg->yyout_r
- #define **yyextra** yyg->yyextra_r
- #define **yyleng** yyg->yyleng_r
- #define **yytext** yyg->yytext_r

- #define **yylineno** (YY_CURRENT_BUFFER_LVALUE->yy_bs_lineno)
- #define **yycolumn** (YY_CURRENT_BUFFER_LVALUE->yy_bs_column)
- #define **yy_flex_debug** yyg->yy_flex_debug_r
- #define **BEGIN** yyg->yy_start = 1 + 2 *
- #define **YY_START** ((yyg->yy_start - 1) / 2)
- #define **YYSTATE** YY_START
- #define **YY_STATE_EOF**(state) (YY_END_OF_BUFFER + state + 1)
- #define **YY_NEW_FILE** DSExpressionFlexrestart(yyin ,yyscanner)
- #define **YY_END_OF_BUFFER_CHAR** 0
- #define **YY_BUF_SIZE** 16384
- #define **YY_STATE_BUF_SIZE** ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))
- #define **YY_TYPEDEF_YY_BUFFER_STATE**
- #define **YY_TYPEDEF_YY_SIZE_T**
- #define **EOB_ACT_CONTINUE_SCAN** 0
- #define **EOB_ACT_END_OF_FILE** 1
- #define **EOB_ACT_LAST_MATCH** 2
- #define **YY_LESS_LINENO**(n)
- #define **yyless**(n)
- #define **unput**(c) yyunput(c, yyg->yytext_ptr , yyscanner)
- #define **YY_STRUCT_YY_BUFFER_STATE**
- #define **YY_BUFFER_NEW** 0
- #define **YY_BUFFER_NORMAL** 1
- #define **YY_BUFFER_EOF_PENDING** 2
- #define **YY_CURRENT_BUFFER**
- #define **YY_CURRENT_BUFFER_LVALUE** yyg->yy_buffer_stack[yyg->yy_buffer_stack_top]

- #define **YY_FLUSH_BUFFER** DSExpressionFlex_flush_buffer(YY_CURRENT_BUFFER ,yyscanner)
- #define **yy_new_buffer** DSExpressionFlex_create_buffer
- #define **yy_set_interactive**(is_interactive)
- #define **yy_set_bol**(at_bol)
- #define **YY_AT_BOL**() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
- #define **yytext_ptr** yytext_r
- #define **YY_DO_BEFORE_ACTION**
- #define **YY_NUM_RULES** 13
- #define **YY_END_OF_BUFFER** 14
- #define **REJECT** reject_used_but_not_detected
- #define **yymore**() yymore_used_but_not_detected
- #define **YY_MORE_ADJ** 0
- #define **YY_RESTORE_YY_MORE_OFFSET**
- #define **malloc**(x) DSSecureMalloc(x)
- #define **calloc**(x, y) DSSecureCalloc(x, y)
- #define **realloc**(x, y) DSSecureRealloc(x, y)
- #define **INITIAL** 0
- #define **YY_EXTRA_TYPE** struct expression_token *
- #define **YY_READ_BUF_SIZE** 8192
- #define **ECHO** fwrite(yytext, yyleng, 1, yyout)
- #define **YY_INPUT**(buf, result, max_size)
- #define **yyterminate**() return YY_NULL
- #define **YY_START_STACK_INCR** 25

- #define **YY_FATAL_ERROR**(msg) yy_fatal_error(msg , yyscanner)
- #define **YY_DECL_IS_OURS** 1
- #define **YY_DECL** int DSExpressionFlexlex (yyscan_t yyscanner)
- #define **YY_USER_ACTION**
- #define **YY_BREAK** break;
- #define **YY_RULE_SETUP** YY_USER_ACTION
- #define **YY_EXIT_FAILURE** 2
- #define **yyless**(n)
- #define **YYTABLES_NAME** "yytables"

Typedefs

- typedef signed char **flex_int8_t**
- typedef short int **flex_int16_t**
- typedef int **flex_int32_t**
- typedef unsigned char **flex_uint8_t**
- typedef unsigned short int **flex_uint16_t**
- typedef unsigned int **flex_uint32_t**
- typedef void * **yyscan_t**
- typedef struct yy_buffer_state * **YY_BUFFER_STATE**
- typedef size_t **yy_size_t**
- typedef unsigned char **YY_CHAR**
- typedef int **yy_state_type**

Functions

- void **DSExpressionFlexrestart** (FILE *input_file, yyscan_t yyscanner)
- void **DSExpressionFlex_switch_to_buffer** (YY_BUFFER_STATE new_buffer, yyscan_t yyscanner)
- YY_BUFFER_STATE **DSExpressionFlex_create_buffer** (FILE *file, int size, yyscan_t yyscanner)

- void **DSExpressionFlex_delete_buffer** (YY_BUFFER_STATE b, yyscan_t yyscanner)
- void DSExpressionFlex_flush_buffer (YY_BUFFER_STATE b, yyscan_t yyscanner)
- void DSExpressionFlexpush_buffer_state (YY_BUFFER_STATE new_buffer, yyscan_t yyscanner)
- void DSExpressionFlexpop_buffer_state (yyscan_t yyscanner)
- YY_BUFFER_STATE DSExpressionFlex_scan_buffer (char *base, yy_size_t size, yyscan_t yyscanner)
- YY_BUFFER_STATE DSExpressionFlex_scan_string (yyconst char *yy_str, yyscan_t yyscanner)
- YY_BUFFER_STATE DSExpressionFlex_scan_bytes (yyconst char *bytes, yy_size_t len, yyscan_t yyscanner)
- void * **DSExpressionFlexalloc** (yy_size_t, yyscan_t yyscanner)
- void * **DSExpressionFlexrealloc** (void *, yy_size_t, yyscan_t yyscanner)
- void **DSExpressionFlexfree** (void *, yyscan_t yyscanner)
- int **DSExpressionFlexlex_init** (yyscan_t *scanner)
- int **DSExpressionFlexlex_init_extra** (YY_EXTRA_TYPE user_defined, yyscan_t *scanner)
- int **DSExpressionFlexlex_destroy** (yyscan_t yyscanner)
- int **DSExpressionFlexget_debug** (yyscan_t yyscanner)
- void **DSExpressionFlexset_debug** (int debug_flag, yyscan_t yyscanner)
- YY_EXTRA_TYPE DSExpressionFlexget_extra (yyscan_t yyscanner)

- void DSExpressionFlexset_extra (YY_EXTRA_TYPE user_defined, yyscan_t yyscanner)
- FILE * DSExpressionFlexget_in (yyscan_t yyscanner)
- void DSExpressionFlexset_in (FILE *in_str, yyscan_t yyscanner)
- FILE * DSExpressionFlexget_out (yyscan_t yyscanner)
- void **DSExpressionFlexset_out** (FILE *out_str, yyscan_t yyscanner)
- yy_size_t DSExpressionFlexget_leng (yyscan_t yyscanner)
- char * DSExpressionFlexget_text (yyscan_t yyscanner)
- int DSExpressionFlexget_lineno (yyscan_t yyscanner)
- void DSExpressionFlexset_lineno (int line_number, yyscan_t yyscanner)
- int **DSExpressionFlexwrap** (yyscan_t yyscanner)
- int **DSExpressionFlexlex** (yyscan_t yyscanner)
- int DSExpressionFlexget_column (yyscan_t yyscanner)
- void DSExpressionFlexset_column (int column_no, yyscan_t yyscanner)
- struct expression_token * **DSExpressionTokenizeString** (const char *string)

7.9.1 Detailed Description

Implementation file with functions for tokenizing matrices, generated by flex. This file was generated directly by the flex program, and is the source code responsible for matrix tokenization. This file was generated by flex, according to a specification written by Jason Lomnitz. To generate this file, the following command must be executed: "flex -t DSExpressionGrammar.l > DSExpressionTokenizerLex.c".

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

Author

Jason Lomnitz.

Date

2011

7.9.2 Define Documentation

7.9.2.1 #define YY_CURRENT_BUFFER

Value:

```
( yyg->yy_buffer_stack \
                          ? yyg->yy_buffer_stack[yyg->yy_buffer_stack_top] \
                          : NULL)
```

7.9.2.2 #define YY_DO_BEFORE_ACTION

Value:

```
yyg->yytext_ptr = yy_bp; \
        yyleng = (size_t) (yy_cp - yy_bp); \
        yyg->yy_hold_char = *yy_cp; \
        *yy_cp = '\0'; \
        yyg->yy_c_buf_p = yy_cp;
```

7.9.2.3 #define YY_INPUT(buf, result, max_size)

Value:

```
if ( YY_CURRENT_BUFFER_LVALUE->yy_is_interactive ) \
                { \
                int c = '*'; \
                yy_size_t n; \
                for ( n = 0; n < max_size && \
                             (c = getc( yyin )) != EOF && c != '\n'; ++n ) \
                        buf[n] = (char) c; \
                if ( c == '\n' ) \
                        buf[n++] = (char) c; \
                if ( c == EOF && ferror( yyin ) ) \
                        YY_FATAL_ERROR( "input in flex scanner failed" ); \
                result = n; \
                } \
        else \
                { \
                errno=0; \
                while ( (result = fread(buf, 1, max_size, yyin))==0 && ferror(yyi
      n)) \
                        { \
                        if( errno != EINTR) \
                                { \
                                YY_FATAL_ERROR( "input in flex scanner failed" );
       \
                                break; \
                                } \
                        errno=0; \
                        clearerr(yyin); \
                        } \
                }\
\
```

7.9.2.4 #define yy_set_bol(at_bol)

Value:

```
{ \
        if ( ! YY_CURRENT_BUFFER ){\
        DSExpressionFlexensure_buffer_stack (yyscanner); \
                YY_CURRENT_BUFFER_LVALUE =    \
            DSExpressionFlex_create_buffer(yyin,YY_BUF_SIZE ,yyscanner); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
        }
```

7.9.2.5 #define yy_set_interactive(is_interactive)

Value:

```
{ \
        if ( ! YY_CURRENT_BUFFER ){ \
        DSExpressionFlexensure_buffer_stack (yyscanner); \
                YY_CURRENT_BUFFER_LVALUE =    \
            DSExpressionFlex_create_buffer(yyin,YY_BUF_SIZE ,yyscanner); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
        }
```

7.9.2.6 #define yyless(n)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                yytext[yyleng] = yyg->yy_hold_char; \
                yyg->yy_c_buf_p = yytext + yyless_macro_arg; \
                yyg->yy_hold_char = *yyg->yy_c_buf_p; \
                *yyg->yy_c_buf_p = '\0'; \
                yyleng = yyless_macro_arg; \
                } \
        while ( 0 )
```

7.9.2.7 #define yyless(n)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                *yy_cp = yyg->yy_hold_char; \
                YY_RESTORE_YY_MORE_OFFSET \
                yyg->yy_c_buf_p = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ;
       \
                YY_DO_BEFORE_ACTION; /* set up yytext again */ \
                } \
        while ( 0 )
```

7.9.3 Function Documentation

7.9.3.1 void DSExpressionFlex_flush_buffer (YY_BUFFER_STATE *b*, yyscan_t *yyscanner*)

Discard all buffered characters. On the next scan, YY_INPUT will be called.

Parameters

b the buffer state to be flushed, usually `YY_CURRENT_BUFFER`.

yyscanner The scanner object.

7.9.3.2 YY_BUFFER_STATE DSExpressionFlex_scan_buffer (char * *base*, yy_size_t *size*, yyscan_t *yyscanner*)

Setup the input buffer state to scan directly from a user-specified character buffer.

Parameters

base the character buffer

size the size in bytes of the character buffer

yyscanner The scanner object.

Returns

the newly allocated buffer state object.

7.9.3.3 YY_BUFFER_STATE DSExpressionFlex_scan_bytes (yyconst char * *yybytes*, yy_size_t *_yybytes_len*, yyscan_t *yyscanner*)

Setup the input buffer state to scan the given bytes. The next call to DSExpressionFlexlex() will scan from a *copy* of *bytes*.

Parameters

bytes the byte buffer to scan

len the number of bytes in the buffer pointed to by *bytes*.

yyscanner The scanner object.

Returns

the newly allocated buffer state object.

7.9.3.4 YY_BUFFER_STATE DSExpressionFlex_scan_string (yyconst char * *yystr*, yyscan_t *yyscanner*)

Setup the input buffer state to scan a string. The next call to DSExpressionFlexlex() will scan from a *copy* of *str*.

Parameters

yystr a NUL-terminated string to scan

yyscanner The scanner object.

Returns

the newly allocated buffer state object.

Note

If you want to scan bytes that may contain NUL values, then use DSExpressionFlex_scan_bytes() instead.

7.9.3.5 int DSExpressionFlexget_column (yyscan_t *yyscanner*)

Get the current column number.

Parameters

yyscanner The scanner object.

7.9.3.6 YY_EXTRA_TYPE DSExpressionFlexget_extra (yyscan_t *yyscanner*)

Get the user-defined data for this scanner.

Parameters

yyscanner The scanner object.

7.9.3.7 FILE * DSExpressionFlexget_in (yyscan_t *yyscanner*)

Get the input stream.

Parameters

yyscanner The scanner object.

7.9.3.8 yy_size_t DSExpressionFlexget_leng (yyscan_t *yyscanner*)

Get the length of the current token.

Parameters

yyscanner The scanner object.

7.9.3.9 int DSExpressionFlexget_lineno (yyscan_t *yyscanner*)

Get the current line number.

Parameters

yyscanner The scanner object.

7.9.3.10 FILE * DSExpressionFlexget_out (yyscan_t *yyscanner*)

Get the output stream.

Parameters

yyscanner The scanner object.

7.9.3.11 char * DSExpressionFlexget_text (yyscan_t *yyscanner*)

Get the current token.

Parameters

yyscanner The scanner object.

7.9.3.12 void DSExpressionFlexpop_buffer_state (yyscan_t *yyscanner*)

Removes and deletes the top of the stack, if present. The next element becomes the new top.

Parameters

yyscanner The scanner object.

7.9.3.13 void DSExpressionFlexpush_buffer_state (YY_BUFFER_STATE *new_buffer*, yyscan_t *yyscanner*)

Pushes the new state onto the stack. The new state becomes the current state. This function will allocate the stack if necessary.

Parameters

new_buffer The new state.

yyscanner The scanner object.

7.9.3.14 void DSExpressionFlexset_column (int *column_no*, yyscan_t *yyscanner*)

Set the current column.

Parameters

line_number

yyscanner The scanner object.

7.9.3.15 void DSExpressionFlexset_extra (YY_EXTRA_TYPE *user_defined*, yyscan_t *yyscanner*)

Set the user-defined data. This data is never touched by the scanner.

Parameters

user_defined The data to be associated with this scanner.

yyscanner The scanner object.

7.9.3.16 void DSExpressionFlexset_in (FILE * *in_str*, yyscan_t *yyscanner*)

Set the input stream. This does not discard the current input buffer.

Parameters

in_str A readable stream.

yyscanner The scanner object.

See also

DSExpressionFlex_switch_to_buffer

7.9.3.17 void DSExpressionFlexset_lineno (int *line_number*, yyscan_t *yyscanner*)

Set the current line number.

Parameters

line_number

yyscanner The scanner object.

7.10 DSGMASystem.c File Reference

Implementation file with functions for dealing with GMA Systems.

```
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include "DSTypes.h"
#include "DSErrors.h"
#include "DSMemoryManager.h"
#include "DSGMASystem.h"
#include "DSExpression.h"
#include "DSExpressionTokenizer.h"
#include "DSGMASystemGrammar.h"
#include "DSMatrix.h"
#include "DSMatrixArray.h"
```

Include dependency graph for DSGMASystem.c:This graph shows which files directly or indirectly include this file:

Defines

- #define **DSGMAXi**(x) ((x)->Xi)
- #define **DSGMAXd**(x) ((x)->Xd)
- #define **DSGMAAlpha**(x) ((x)->alpha)
- #define **DSGMABeta**(x) ((x)->beta)
- #define **DSGMAGd**(x) ((x)->Gd)
- #define **DSGMAGi**(x) ((x)->Gi)
- #define **DSGMAHd**(x) ((x)->Hd)
- #define **DSGMAHi**(x) ((x)->Hi)
- #define **DSGMASignature**(x) ((x)->signature)

Functions

- void **DSGMASystemFree** (DSGMASystem *gma)
- DSGMASystem * **DSGMASystemByParsingStringList** (const DSVariablePool *const Xd, const char *const string,...)
- DSGMASystem * **DSGMASystemByParsingStrings** (const DSVariablePool *const Xd, char *const *const strings, const DSUInteger numberOfEquations)
- const DSUInteger **DSGMASystemNumberOfCases** (const DSGMASystem *gma)
- const DSUInteger **DSGMASystemNumberOfEquations** (const DSGMASystem *gma)
- DSExpression ** **DSGMASystemEquations** (const DSGMASystem *gma)
- const DSMatrix * **DSGMASystemAlpha** (const DSGMASystem *gma)
- const DSMatrix * **DSGMASystemBeta** (const DSGMASystem *gma)
- const DSMatrixArray * **DSGMASystemGd** (const DSGMASystem *gma)
- const DSMatrixArray * **DSGMASystemGi** (const DSGMASystem *gma)

- const DSMatrixArray * **DSGMASystemHd** (const DSGMASystem *gma)
- const DSMatrixArray * **DSGMASystemHi** (const DSGMASystem *gma)
- const DSVariablePool * **DSGMASystemXd** (const DSGMASystem *gma)
- const DSVariablePool * **DSGMASystemXi** (const DSGMASystem *gma)
- const DSUInteger * **DSGMASystemSignature** (const DSGMASystem *gma)
- void **DSGMASystemPrint** (const DSGMASystem *gma)
- void **DSGMASystemPrintEquations** (const DSGMASystem *gma)

7.10.1 Detailed Description

Implementation file with functions for dealing with GMA Systems. 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

7.11 DSGMASystem.h File Reference

Header file with functions for dealing with GMA Systems.

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

```
#include "DSVariable.h"
```

Include dependency graph for DSGMASystem.h:This graph shows which files directly or indirectly include this file:

Defines

- #define **M_DS_GMA_NULL** M_DS_NULL ": GMA System is NULL"

Functions

- void **DSGMASystemFree** (DSGMASystem *gma)
- DSGMASystem * **DSGMASystemByParsingStringList** (const DSVariablePool *const Xd, const char *const string,...)
- DSGMASystem * **DSGMASystemByParsingStrings** (const DSVariablePool *const Xd, char *const *const strings, const DSUInteger numberOfEquations)
- const DSUInteger **DSGMASystemNumberOfEquations** (const DSGMASystem *gma)
- DSExpression ** **DSGMASystemEquations** (const DSGMASystem *gma)
- const DSMatrix * **DSGMASystemAlpha** (const DSGMASystem *gma)
- const DSMatrix * **DSGMASystemBeta** (const DSGMASystem *gma)
- const DSMatrixArray * **DSGMASystemGd** (const DSGMASystem *gma)
- const DSMatrixArray * **DSGMASystemGi** (const DSGMASystem *gma)
- const DSMatrixArray * **DSGMASystemHd** (const DSGMASystem *gma)
- const DSMatrixArray * **DSGMASystemHi** (const DSGMASystem *gma)
- const DSVariablePool * **DSGMASystemXd** (const DSGMASystem *gma)
- const DSVariablePool * **DSGMASystemXi** (const DSGMASystem *gma)
- const DSUInteger **DSGMASystemNumberOfCases** (const DSGMASystem *gma)
- const DSUInteger * **DSGMASystemSignature** (const DSGMASystem *gma)
- void **DSGMASystemPrint** (const DSGMASystem *gma)
- void **DSGMASystemPrintEquations** (const DSGMASystem *gma)

7.11.1 Detailed Description

Header file with functions for dealing with GMA Systems. 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

7.12 DSGMASystemParsingAux.h File Reference

Implementation file with functions for dealing with the parsing of GMA Systems.

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

Include dependency graph for DSGMASystemParsingAux.h:

Data Structures

- struct parser_aux

  Data type used to parse strings to GMA System.

- union base_info

Defines

- #define **AUX_EXPONENT_CONSTANT_BASE** NAN
- #define **AUX_SIGN_UNDEFINED** '?'
- #define **AUX_SIGN_POSITIVE** '+'
- #define **AUX_SIGN_NEGATIVE** '-'
- #define **AUX_PARSER_FAILED** false
- #define **AUX_PARSER_SUCCESS** true
- #define **DSGMAParserAuxNumberOfBases**(x) (x->numberOfBases)
- #define **DSGMAParserAuxBaseAtIndexIsVariable**(x, y) (!isnan(DSGMAParserAuxExponentAtIndex(x, y)))
- #define **DSGMAParserAuxSetParserFailed**(x) ((x)->succeded = false)

Typedefs

- typedef struct parser_aux gma_parseraux_t

  Data type used to parse strings to GMA System.

Functions

- gma_parseraux_t * **DSGMAParserAuxAlloc** (void)
- void **DSGMAParserAuxFree** (gma_parseraux_t *root)
- void **DSGMAParserAuxNewTerm** (gma_parseraux_t *current)
- gma_parseraux_t * **DSGMAParserAuxNextNode** (const gma_parseraux_t *const aux)
- void **DSGMAParserAuxSetSign** (gma_parseraux_t *aux, const char sign)
- void **DSGMAParserAuxAddVariableExponentPair** (gma_parseraux_t *aux, const char *const name, const double exponent)
- void **DSGMAParserAuxAddConstantBase** (gma_parseraux_t *aux, const double base)
- const char **DSGMAParserAuxSign** (const gma_parseraux_t *const aux)
- const double **DSGMAParserAuxExponentAtIndex** (const gma_parseraux_t *const aux, const DSUInteger index)
- const char *const **DSGMAParserAuxVariableAtIndex** (const gma_parseraux_t *const aux, const DSUInteger index)

- const double **DSGMAParseAuxsConstantBaseAtIndex** (const gma_parseraux_t *const aux, const DSUInteger index)
- const bool **DSGMAParserAuxParsingFailed** (const gma_parseraux_t *const aux)

7.12.1 Detailed Description

Implementation file with functions for dealing with the parsing of GMA Systems. Header file with functions for dealing with the parsing of GMA Systems.

Author

Jason Lomnitz.

Date

2011

7.12.2 Typedef Documentation

7.12.2.1 typedef struct parser_aux gma_parseraux_t

Data type used to parse strings to GMA System.

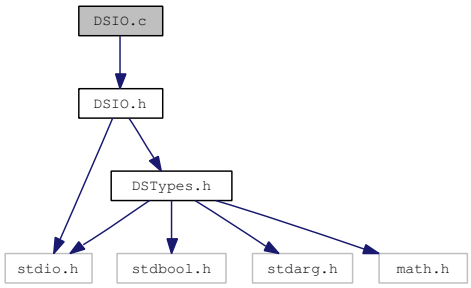
This data structure forms an organized list of terms, each with base exponent pairs that are then used to create the system matrices. This data structure is key for the parsing of GMA systems. Each node in the gma_parseraux_t list represent a term in an expression in the order it was found, and each node points to the next term. Each expression, or equation, has it's own list of terms. If a base is a constant, then it should not have an exponent, and hence it's exponent is assigned a NAN value and this is used to indicate that the base is a constant.

7.13 DSIO.c File Reference

Implementation file with standard input and output functions.

```
#include <stdio.h>
#include <string.h>
#include "DSIO.h"
#include "DSMemoryManager.h"
#include "DSVariable.h"
#include "DSMatrix.h"
#include "DSMatrixArray.h"
#include "DSGMASystem.h"
#include "DSSSystem.h"
#include "DSCase.h"
```

Include dependency graph for DSIO.c:



Functions

- void DSIOSetErrorFile (FILE *aFile)

  Function to assign default error file.

- void DSIOSetPrintFunction (int(*printFunction)(const char *,...))

  Function to assign default printf function.

- void DSIOSetPostWarningFunction (void(*warningFunction)(const char *message))

  Function to assign default warning posting function.

- void DSIOSetPostErrorFunction (void(*errorFunction)(const char *message))

  Function to assign default error posting function.

- void DSIOSetPostFatalErrorFunction (void(*fatalErrorFunction)(const char *message))

  Function to assign default fatal error posting function.

- void DSIOSetCaseJSONOptions (const DSUInteger options)

  Function that sets the conversion options for a DSCase to JSON format.

- void DSIOSetSSystemJSONOptions (const DSUInteger options)

  Function that sets the conversion options for a DSSSystem to JSON format.

- char * DSVariablePoolStringInJSONFormat (const DSVariablePool *pool)

  Function to convert a DSVariablePool into a JSON formatted string.

- char * DSMatrixStringInJSONFormat (const DSMatrix *matrix)

  Function to convert a DSMatrix into a JSON formatted string.

- char * DSMatrixArrayStringInJSONFormat (const DSMatrixArray *array)

  Function to convert a DSMatrixArray into a JSON formatted string.

- char * DSSSystemStringInJSONFormat (const DSSSystem *ssys)

  Function to convert a DSSSystem into a JSON formatted string.

- char * DSCaseStringInJSONFormat (const DSCase *aCase)

  Function to convert a DSCase into a JSON formatted string.

Variables

- DSUInteger DSSSystemPrintingOptions

  Variable with flags controlling S-System to JSON string conversion.

- DSUInteger DSCasePrintingOptions

  Variable with flags controlling the conversion of a Case to a JSON string.

7.13.1 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

7.13.2 Function Documentation

7.13.2.1 char* DSCaseStringInJSONFormat (const DSCase * *aCase*)

Function to convert a DSCase into a JSON formatted string.

This function is used to convert a DSCase into a JSON object. The DSCase is represented with a set of objects, where each object is a field of the DSCase object. The default behavior exports all of the fields, this behavior can be overwritten by changing the DSCase conversion options.

Parameters

aCase A DSCase that will be used to create the JSON object.

Returns

A C string with the JSON formatted data. If NULL, the conversion failed.

See also

DSIOSetCaseJSONOptions()

7.13.2.2 void DSIOSetCaseJSONOptions (const DSUInteger *options*)

Function that sets the conversion options for a DSCase to JSON format.

This function is used to overwrite the default export behavior of the DSCase object. The default behavior converts all of the data fields of the DSCase into a JSON format, these options can be changed so the JSON conversion only includes some fields, such as excluding the conditions, excluding the S-System, etc.

Parameters

options A DSUInteger with the option flags, as specified by the DSCase options.

See also

Options for JSON conversion of DSCase object.

7.13.2.3 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

7.13.2.4 void DSIOSetPostErrorFunction (void(∗)(const char ∗message) *errorFunction*)

Function to assign default error posting function.

This function is used to assign the function that handles the errors generated from the design space toolbox. Internally, it assigns the global variable DSPostError which points to a function.

Parameters

errorFunction A pointer to a function of the form void function(const char ∗). If NULL, default behavior is restored.

7.13.2.5 void DSIOSetPostFatalErrorFunction (void(∗)(const char ∗message) *fatalErrorFunction*)

Function to assign default fatal error posting function.

This function is used to assign the function that handles the fatal errors generated from the design space toolbox. Internally, it assigns the global variable DSPostFatalError which points to a function.

Parameters

errorFunction A pointer to a function of the form void function(const char ∗). If NULL, default behavior is restored.

7.13.2.6 void DSIOSetPostWarningFunction (void(∗)(const char ∗message) *warningFunction*)

Function to assign default warning posting function.

This function is used to assign the function that handles the warnings generated from the design space toolbox. Internally, it assigns the global variable DSPostWarning which points to a function.

Parameters

warningFunction A pointer to a function of the form void function(const char ∗). If NULL, default behavior is restored.

7.13.2.7 void DSIOSetPrintFunction (int(∗)(const char ∗,...) *printFunction*)

Function to assign default printf function.

This function is used to assign the formated print function, DSPrintf. This function assigns the DSPrintf pointer to the function that should be used to print formatted strings. This function MUST be used to avoid problems relating to dynamic linking; by using this function the global variable DSPrintf is loaded into memory prior to changing its value.

Parameters

printFunction A pointer to a function of the form int function(const char ∗, ...). If NULL, default behavior is restored.

7.13.2.8 void DSIOSetSSystemJSONOptions (const DSUInteger *options*)

Function that sets the conversion options for a DSSSystem to JSON format.

This function is used to overwrite the default export behavior of the DSSSystem object. The default behavior converts all of the data fields of the S-System into a JSON format, these options can be changed so the JSON conversion only includes some fields, such as excluding the solution.

Parameters

options A DSUInteger with the option flags, as specified by the DSSSystem options.

See also

Options for JSON conversion of DSSSystem object.

7.13.2.9 char* DSMatrixArrayStringInJSONFormat (const DSMatrixArray * *array*)

Function to convert a DSMatrixArray into a JSON formatted string.

This function is used to convert a DSMatrix into a JSON object. The matrix array is stored as an array of objects, where each object is a DSMatrix. The order of the DSMatrix object in the array represent the order of matrices in the matrix array.

Parameters

array A DSMatrixArray that will be used to create the JSON object.

Returns

A C string with the JSON formatted data. If NULL, the conversion failed.

7.13.2.10 char* DSMatrixStringInJSONFormat (const DSMatrix * *matrix*)

Function to convert a DSMatrix into a JSON formatted string.

This function is used to convert a DSMatrix into a JSON object. The matrix is stored as an array of arrays. The array of arrays represents the rows of the matrix, whereas the arrays of value are the values at the columns for a particular row.

Parameters

matrix A DSMatrix that will be used to create the JSON object.

Returns

A C string with the JSON formatted data. If NULL, the conversion failed.

7.13.2.11 char* DSSSystemStringInJSONFormat (const DSSSystem * *ssys*)

Function to convert a DSSSystem into a JSON formatted string.

This function is used to convert a DSSSystem into a JSON object. The S-System as a set of objects, where each object represents each of the fields of the DSSSystem. The default behavior exports all of the fields, this behavior can be overwritten by changing the S-System conversion options.

Parameters

ssys A DSSSystem that will be used to create the JSON object.

Returns

A C string with the JSON formatted data. If NULL, the conversion failed.

See also

DSIOSetSSystemJSONOptions()

7.13.2.12 char* DSVariablePoolStringInJSONFormat (const DSVariablePool * *pool*)

Function to convert a DSVariablePool into a JSON formatted string.

This function is used to convert a DSVariablePool into a JSON object. The variables of the variable pool are stored as pairs of a string and value.

Parameters

pool A DSVariablePool that will be used to create the JSON object.

Returns

A C string with the JSON formatted data. If NULL, the conversion failed.

7.13.3 Variable Documentation

7.13.3.1 DSUInteger DSCasePrintingOptions

Variable with flags controlling the conversion of a Case to a JSON string.

This global variable is checked when converting a Case structure to a JSON string. This variable will check several flags as specified by DS_CASE_JSON_OPTIONS. The default value of the variable indicates that all the properties will be included in the JSON string.

See also

Options for JSON conversion of DSCase object.
DSIOSetCaseJSONOptions()

7.13.3.2 DSUInteger DSSSystemPrintingOptions

Variable with flags controlling S-System to JSON string conversion.

This global variable is checked when converting a S-System structure to a JSON string. This variable will check several flags as specified by DS_SSYSTEM_JSON_OPTIONS. The default value of the variable indicates that all the properties will be included in the JSON string.

See also

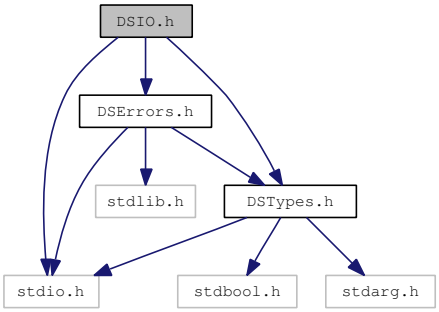
Options for JSON conversion of DSSSystem object.
DSIOSetSSystemJSONOptions()

7.14 DSIO.h File Reference

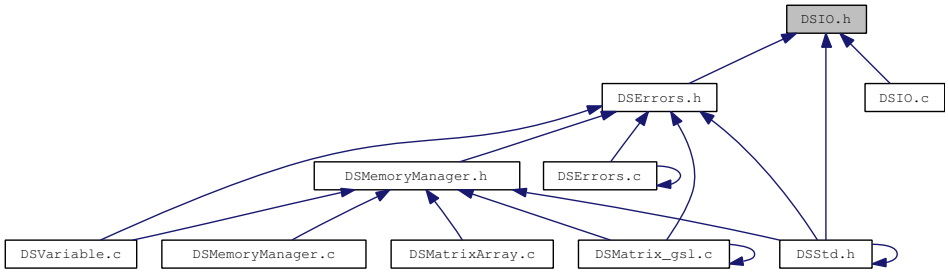
Header file with standard input and output functions.

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

Include dependency graph for DSIO.h:



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



Defines

- #define DS_CASE_JSON_NO_SSYSTEM 1

  Flag value indicating that the S-System information should not be included in the JSON string.

- #define DS_CASE_JSON_NO_CASE_SIGNATURE 2

  Flag value indicating that the case signature should not be included in the JSON string.

- #define DS_CASE_JSON_NO_CONDITIONS 4

  Flag value indicating that the conditions for validity should not be included in the JSON string.

- #define DS_SSYSTEM_JSON_NO_SOLUTION 1

  Flag value indicating that the S-System solution should not be included in the JSON string.

- #define DS_SSYSTEM_JSON_NO_SINGULAR 2

  Flag value indicating that the JSON string will not indicate if the S-System is singular.

Functions

- void DSIOSetErrorFile (FILE *aFile)
  Function to assign default error file.
- void DSIOSetPrintFunction (int(*printFunction)(const char *,...))
  Function to assign default printf function.
- void DSIOSetPostWarningFunction (void(*warningFunction)(const char *message))
  Function to assign default warning posting function.
- void DSIOSetPostErrorFunction (void(*errorFunction)(const char *message))
  Function to assign default error posting function.
- void DSIOSetPostFatalErrorFunction (void(*fatalErrorFunction)(const char *message))
  Function to assign default fatal error posting function.
- void DSIOSetCaseJSONOptions (const DSUInteger options)
  Function that sets the conversion options for a DSCase to JSON format.
- void DSIOSetSSystemJSONOptions (const DSUInteger options)
  Function that sets the conversion options for a DSSSystem to JSON format.
- char * DSVariablePoolStringInJSONFormat (const DSVariablePool *pool)
  Function to convert a DSVariablePool into a JSON formatted string.
- char * DSMatrixStringInJSONFormat (const DSMatrix *matrix)
  Function to convert a DSMatrix into a JSON formatted string.
- char * DSMatrixArrayStringInJSONFormat (const DSMatrixArray *array)
  Function to convert a DSMatrixArray into a JSON formatted string.
- char * DSSSystemStringInJSONFormat (const DSSSystem *ssys)
  Function to convert a DSSSystem into a JSON formatted string.
- char * DSCaseStringInJSONFormat (const DSCase *aCase)
  Function to convert a DSCase into a JSON formatted string.
- DSVariablePool * **DSVariablePoolByParsingStringInJSONFormat** (const char *string)
- DSMatrix * **DSMatrixByParsingStringInJSONFormat** (const char *string)
- DSMatrixArray * **DSMatrixArrayByParsingStringInJSONFormat** (const char *string)
- DSSSystem * **DSSSystemByParsingStringInJSONFormat** (const char *string)
- DSCase * **DSCaseByParsingStringInJSONFormat** (const char *string)

Variables

- int(* DSPrintf)(const char *,...)
  Pointer to a function determining how messages are printed.
- void(* DSPostWarning)(const char *message)

Pointer to a function determining how warning are handled.

- void(∗ DSPostError)(const char ∗message)

  Pointer to a function determining how errors are handled.

- void(∗ DSPostFatalError)(const char ∗message)

  Pointer to a function determining how fatal errors are handled.

- FILE ∗ DSIOErrorFile

  FILE pointer used for default error/warning printing.

7.14.1 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

Define standard input and output file formats.
Define criteria for warnings, errors and fatal errors.

7.14.2 Function Documentation

7.14.2.1 char∗ DSCaseStringInJSONFormat (const DSCase ∗ *aCase*)

Function to convert a DSCase into a JSON formatted string.

This function is used to convert a DSCase into a JSON object. The DSCase is represented with a set of objects, where each object is a field of the DSCase object. The default behavior exports all of the fields, this behavior can be overwritten by changing the DSCase conversion options.

Parameters

aCase A DSCase that will be used to create the JSON object.

Returns

A C string with the JSON formatted data. If NULL, the conversion failed.

See also

DSIOSetCaseJSONOptions()

7.14.2.2 void DSIOSetCaseJSONOptions (const DSUInteger *options*)

Function that sets the conversion options for a DSCase to JSON format.

This function is used to overwrite the default export behavior of the DSCase object. The default behavior converts all of the data fields of the DSCase into a JSON format, these options can be changed so the JSON conversion only includes some fields, such as excluding the conditions, excluding the S-System, etc.

Parameters

options A DSUInteger with the option flags, as specified by the DSCase options.

See also

Options for JSON conversion of DSCase object.

7.14.2.3 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

7.14.2.4 void DSIOSetPostErrorFunction (void(*)(const char *message) *errorFunction*)

Function to assign default error posting function.

This function is used to assign the function that handles the errors generated from the design space toolbox. Internally, it assigns the global variable DSPostError which points to a function.

Parameters

errorFunction A pointer to a function of the form void function(const char *). If NULL, default behavior is restored.

7.14.2.5 void DSIOSetPostFatalErrorFunction (void(∗)(const char ∗message) *fatalErrorFunction*)

Function to assign default fatal error posting function.

This function is used to assign the function that handles the fatal errors generated from the design space toolbox. Internally, it assigns the global variable DSPostFatalError which points to a function.

Parameters

errorFunction A pointer to a function of the form void function(const char ∗). If NULL, default behavior is restored.

7.14.2.6 void DSIOSetPostWarningFunction (void(∗)(const char ∗message) *warningFunction*)

Function to assign default warning posting function.

This function is used to assign the function that handles the warnings generated from the design space toolbox. Internally, it assigns the global variable DSPostWarning which points to a function.

Parameters

warningFunction A pointer to a function of the form void function(const char ∗). If NULL, default behavior is restored.

7.14.2.7 void DSIOSetPrintFunction (int(∗)(const char ∗,...) *printFunction*)

Function to assign default printf function.

This function is used to assign the formated print function, DSPrintf. This function assigns the DSPrintf pointer to the function that should be used to print formatted strings. This function MUST be used to avoid problems relating to dynamic linking; by using this function the global variable DSPrintf is loaded into memory prior to changing its value.

Parameters

printFunction A pointer to a function of the form int function(const char ∗, ...). If NULL, default behavior is restored.

7.14.2.8 void DSIOSetSSystemJSONOptions (const DSUInteger *options*)

Function that sets the conversion options for a DSSSystem to JSON format.

This function is used to overwrite the default export behavior of the DSSSystem object. The default behavior converts all of the data fields of the S-System into a JSON format, these options can be changed so the JSON conversion only includes some fields, such as excluding the solution.

Parameters

options A DSUInteger with the option flags, as specified by the DSSSystem options.

See also

Options for JSON conversion of DSSSystem object.

7.14.2.9 char* DSMatrixArrayStringInJSONFormat (const DSMatrixArray * *array*)

Function to convert a DSMatrixArray into a JSON formatted string.

This function is used to convert a DSMatrix into a JSON object. The matrix array is stored as an array of objects, where each object is a DSMatrix. The order of the DSMatrix object in the array represent the order of matrices in the matrix array.

Parameters

array A DSMatrixArray that will be used to create the JSON object.

Returns

A C string with the JSON formatted data. If NULL, the conversion failed.

7.14.2.10 char* DSMatrixStringInJSONFormat (const DSMatrix * *matrix*)

Function to convert a DSMatrix into a JSON formatted string.

This function is used to convert a DSMatrix into a JSON object. The matrix is stored as an array of arrays. The array of arrays represents the rows of the matrix, whereas the arrays of value are the values at the columns for a particular row.

Parameters

matrix A DSMatrix that will be used to create the JSON object.

Returns

A C string with the JSON formatted data. If NULL, the conversion failed.

7.14.2.11 char* DSSSystemStringInJSONFormat (const DSSSystem * *ssys*)

Function to convert a DSSSystem into a JSON formatted string.

This function is used to convert a DSSSystem into a JSON object. The S-System as a set of objects, where each object represents each of the fields of the DSSSystem. The default behavior exports all of the fields, this behavior can be overwritten by changing the S-System conversion options.

Parameters

ssys A DSSSystem that will be used to create the JSON object.

Returns

A C string with the JSON formatted data. If NULL, the conversion failed.

See also

DSIOSetSSystemJSONOptions()

7.14.2.12 char* DSVariablePoolStringInJSONFormat (const DSVariablePool * *pool*)

Function to convert a DSVariablePool into a JSON formatted string.

This function is used to convert a DSVariablePool into a JSON object. The variables of the variable pool are stored as pairs of a string and value.

Parameters

pool A DSVariablePool that will be used to create the JSON object.

Returns

A C string with the JSON formatted data. If NULL, the conversion failed.

7.14.3 Variable Documentation

7.14.3.1 FILE* DSIOErrorFile

FILE pointer used for default error/warning printing.

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 sets it to the stderr file. This variable is only used internally with the default behavior of DSErrorFunction. To change the error file, the function DSIOSetErrorFile should be used in order to avoid errors caused by dynamic linking. These errors involve changing the value of a global variable that has not yet been loaded by the linker.

See also

DSIOSetErrorFile
DSErrorFunction

7.14.3.2 void(* DSPostError)(const char *message)

Pointer to a function determining how errors are handled.

This pointer to a function is used by DSErrorFunction to post erros. This pointer should be used to allow better integration of errors in programs that make use of the DesignSpaceToolbox. The function takes one argument, a constant C string with the error message. To change the function used, the function DSIOSetPostErrorFunction should be used. This is to avoid errors caused by dynamic linking. These errors involve changing the value of a global variable that has not yet been loaded by the linker.

See also

DSIOSetPostErrorFunction

7.14.3.3 void(* DSPostFatalError)(const char *message)

Pointer to a function determining how fatal errors are handled.

This pointer to a function is used by DSErrorFunction to post fatal erros. This pointer should be used to allow better integration of errors in programs that make use of the DesignSpaceToolbox. The function takes one argument, a constant C string with the error message. To change the function used, the function DSIOSetPostFatalErrorFunction should be used. This is to avoid errors caused by dynamic linking. These errors involve changing the value of a global variable that has not yet been loaded by the linker.

See also

DSIOSetPostErrorFunction

7.14.3.4 void(∗ DSPostWarning)(const char ∗message)

Pointer to a function determining how warning are handled.

This pointer to a function is used by DSErrorFunction to post warnings. This pointer should be used to allow better integration of warnings in programs that make use of the DesignSpaceToolbox. The function takes one argument, a constant C string with the warning message. To change the function used, the function DSIOSetPostWarningFunction should be used. This is to avoid errors caused by dynamic linking. These errors involve changing the value of a global variable that has not yet been loaded by the linker.

See also

DSIOSetPostWarningFunction

7.14.3.5 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

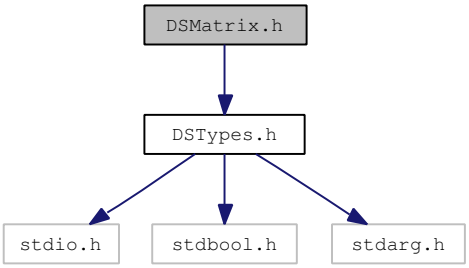
DSIOSetPrintFunction

7.15 DSMatrix.h File Reference

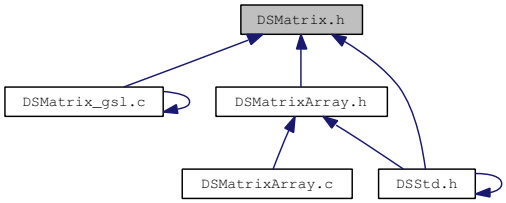
Header file with functions for dealing with matrices.

`#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 * **DSMatrixByParsingString** (const char *string)
- 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 DSMatrixDoubleValue (const DSMatrix *matrix, const DSUInteger row, const DSUInteger column)

  Returns the element of the DSMatrix specified by a row and column.

- 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)
- DSMatrix * **DSMatrixWithUniqueRows** (const DSMatrix *matrix)
- void **DSMatrixPrint** (const 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)
- void **DSMatrixSubstractByMatrix** (DSMatrix *addTo, const DSMatrix *addBy)
- void **DSMatrixAddByMatrix** (DSMatrix *addTo, const DSMatrix *addBy)
- void **DSMatrixApplyFunction** (DSMatrix *matrix, double(*function)(double))
- void **DSMatrixMultiplyByScalar** (DSMatrix *matrix, const double value)
- 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)

7.15.1 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

7.15.2 Function Documentation

7.15.2.1 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.

7.15.2.2 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.

7.15.2.3 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.

7.15.2.4 double DSMatrixDoubleValue (const DSMatrix * *matrix*, const DSUInteger *row*, const DSUInteger *column*)

Returns the element of the DSMatrix specified by a row and column.

Returns an element of the matrix, with indices i and j starting at 0.

Parameters

matrix The DSMatrix whose elements will be accessed.

row A DSUInteger specifying the row coordinate of the element to be accessed.

column A DSUInteger specifying the column coordinate of the element to be accessed.

Returns

If the value was succesfully retrieved, the double value contained at the row and column coordinate of the DSMatrix is returned. Otherwise, NaN is returned.

7.15.2.5 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.

7.15.2.6 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.

7.15.2.7 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.

7.15.2.8 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.

7.15.2.9 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.

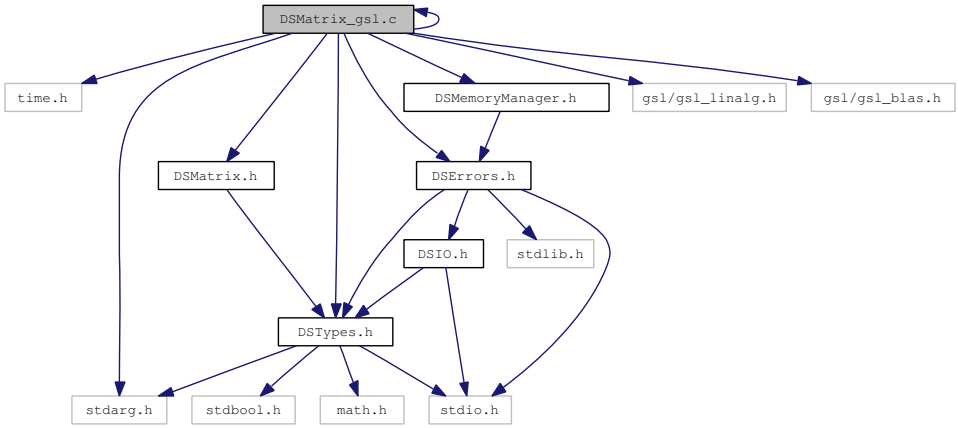
value The double variable whose value will be assigned.

7.16 DSMatrix_gsl.c File Reference

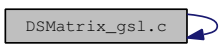
Implementation file with functions for dealing with matrices using the GNU Scientific Library (gsl).

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

Include dependency graph for DSMatrix_gsl.c:



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



Defines

- #define **DSMatrixSetRows**(x, y) ((x)->rows = (y))
- #define **DSMatrixSetColumns**(x, y) ((x)->columns = (y))

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 * **DSMatrixByParsingString** (const char *string)
- 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 DSMatrixDoubleValue (const DSMatrix *matrix, const DSUInteger row, const DSUInteger column)

  Returns the element of the DSMatrix specified by a row and column.

- 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)
- DSMatrix * **DSMatrixWithUniqueRows** (const DSMatrix *matrix)
- void **DSMatrixPrint** (const 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)
- void **DSMatrixAddByMatrix** (DSMatrix *addTo, const DSMatrix *addBy)
- void **DSMatrixSubstractByMatrix** (DSMatrix *addTo, const DSMatrix *addBy)
- void **DSMatrixApplyFunction** (DSMatrix *matrix, double(*function)(double))
- void **DSMatrixMultiplyByScalar** (DSMatrix *matrix, const double value)
- 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)

7.16.1 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

7.16.2 Function Documentation

7.16.2.1 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.

7.16.2.2 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.

7.16.2.3 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.

7.16.2.4 double DSMatrixDoubleValue (const DSMatrix * *matrix*, const DSUInteger *row*, const DSUInteger *column*)

Returns the element of the DSMatrix specified by a row and column.

Returns an element of the matrix, with indices i and j starting at 0.

Parameters

matrix The DSMatrix whose elements will be accessed.

row A DSUInteger specifying the row coordinate of the element to be accessed.

column A DSUInteger specifying the column coordinate of the element to be accessed.

Returns

If the value was succesfully retrieved, the double value contained at the row and column coordinate of the DSMatrix is returned. Otherwise, NaN is returned.

7.16.2.5 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.

7.16.2.6 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.

7.16.2.7 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.

7.16.2.8 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.

7.16.2.9 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.

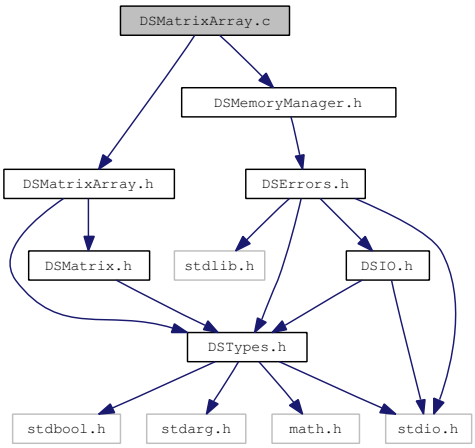
value The double variable whose value will be assigned.

7.17 DSMatrixArray.c File Reference

Implementation file with functions for dealing with matrix arrays.

```
#include <string.h>
#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.

- double **DSMatrixArrayDoubleWithIndices** (const DSMatrixArray *array, const DSUInteger i, const DSUInteger j, const DSUInteger k)
- void **DSMatrixArrayPrint** (const DSMatrixArray *array)

7.17.1 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).

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

7.17.2 Function Documentation

7.17.2.1 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.

7.17.2.2 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.

7.17.2.3 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

7.17.2.4 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.

7.17.2.5 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

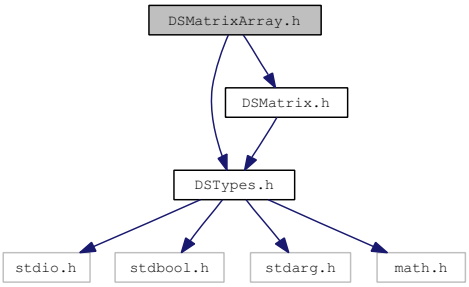
If the DSMatrix at the specified index was found, the pointer to that matrix is returned. Otherwise, NULL is returned.

7.18 DSMatrixArray.h File Reference

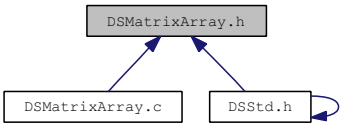
Header file with functions for dealing with matrix arrays.

```
#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.

- double **DSMatrixArrayDoubleWithIndices** (const DSMatrixArray *array, const DSUInteger i, const DSUInteger j, const DSUInteger k)
- void **DSMatrixArrayPrint** (const DSMatrixArray *array)

7.18.1 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).

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

7.18.2 Function Documentation

7.18.2.1 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.

7.18.2.2 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.

7.18.2.3 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

7.18.2.4 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.

7.18.2.5 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.

7.19 DSMatrixTokenizer.c File Reference

Implementation file with functions for tokenizing with matrices.

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

```
#include "DSMatrixTokenizer.h"
```

Include dependency graph for DSMatrixTokenizer.c:

Functions

- struct matrix_token * **DSMatrixTokenAlloc** ()
- void **DSMatrixTokenFree** (struct matrix_token *root)

7.19.1 Detailed Description

Implementation file with functions for tokenizing with matrices. 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

7.20 DSMatrixTokenizer.h File Reference

Header file with functions for tokenizing matrices.

```
#include "DSTypes.h"
#include "DSErrors.h"
#include "DSMemoryManager.h"
```

Include dependency graph for DSMatrixTokenizer.h:This graph shows which files directly or indirectly include this file:

Data Structures

- struct matrix_token

Defines

- #define DS_MATRIX_TOKEN_START 0

  Token indicating the start of a tokenization.

- #define DS_MATRIX_TOKEN_DOUBLE 1

  Token indicating a numerical value.

- #define DS_MATRIX_TOKEN_NEWLINE 2

  Token indicating a newline, indicative of a new row.

- #define DS_MATRIX_TOKEN_ERROR 3

  Token indicating an error during tokenization.

- #define **DSMatrixTokenNext**(x) ((x)->next)
- #define **DSMatrixTokenValue**(x) ((x)->value)
- #define **DSMatrixTokenType**(x) ((x)->token)
- #define **DSMatrixTokenRow**(x) ((x)->row)
- #define **DSMatrixTokenColumn**(x) ((x)->column)
- #define **DSMatrixTokenSetNext**(x, y) ((x)->next = (y))
- #define **DSMatrixTokenSetValue**(x, y) ((x)->value = (y))
- #define **DSMatrixTokenSetType**(x, y) ((x)->token = (y))
- #define **DSMatrixTokenSetRow**(x, y) ((x)->row = (y))
- #define **DSMatrixTokenSetColumn**(x, y) ((x)->column = (y))

Functions

- struct matrix_token * **DSMatrixTokenAlloc** ()
- void **DSMatrixTokenFree** (struct matrix_token *root)
- struct matrix_token * **DSMatrixTokenizeString** (const char *string)

7.20.1 Detailed Description

Header file with functions for tokenizing matrices. This header file specifies the data structure relating to the tokenization of an input string to be parsed as a matrix, as well as all the functions necessary to tokenize it. This file is a provate file, and therefore its contents will be invisible to the public API. As such, it is not necessary to place the C++ compatability decleration.

Author

Jason Lomnitz.

Date

2011

This header file specifies the data structure relating to the tokenization of an input string to be parsed as a matrix, as well as all the functions necessary to tokenize it. This file is a private file, and therefore its contents will be invisible to the public API. Therefore, it is unnecessary to place the C++ compatability declerations.

Author

Jason Lomnitz.

Date

2011

7.21 DSMatrixTokenizerLex.c File Reference

Implementation file with functions for tokenizing matrices, generated by flex.

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include "DSTypes.h"
#include "DSMemoryManager.h"
#include "DSMatrix.h"
#include "DSMatrixTokenizer.h"
#include <unistd.h>
```

Include dependency graph for DSMatrixTokenizerLex.c:

Data Structures

- struct yy_buffer_state
- struct yy_trans_info
- struct yyguts_t

Defines

- #define **YY_INT_ALIGNED** short int
- #define **FLEX_SCANNER**
- #define **YY_FLEX_MAJOR_VERSION** 2
- #define **YY_FLEX_MINOR_VERSION** 5
- #define **YY_FLEX_SUBMINOR_VERSION** 35
- #define **INT16_MIN** (-32767-1)
- #define **INT32_MIN** (-2147483647-1)
- #define **INT8_MAX** (127)
- #define **INT16_MAX** (32767)
- #define **INT32_MAX** (2147483647)
- #define **UINT8_MAX** (255U)
- #define **UINT16_MAX** (65535U)
- #define **UINT32_MAX** (4294967295U)
- #define **yyconst**
- #define **YY_NULL** 0
- #define **YY_SC_TO_UI**(c) ((unsigned int) (unsigned char) c)
- #define **YY_TYPEDEF_YY_SCANNER_T**
- #define **yyin** yyg->yyin_r
- #define **yyout** yyg->yyout_r
- #define **yyextra** yyg->yyextra_r
- #define **yyleng** yyg->yyleng_r
- #define **yytext** yyg->yytext_r
- #define **yylineno** (YY_CURRENT_BUFFER_LVALUE->yy_bs_lineno)

- #define **yycolumn** (YY_CURRENT_BUFFER_LVALUE->yy_bs_column)
- #define **yy_flex_debug** yyg->yy_flex_debug_r
- #define **BEGIN** yyg->yy_start = 1 + 2 *
- #define **YY_START** ((yyg->yy_start - 1) / 2)
- #define **YYSTATE** YY_START
- #define **YY_STATE_EOF**(state) (YY_END_OF_BUFFER + state + 1)
- #define **YY_NEW_FILE** DSMatrixFlexrestart(yyin ,yyscanner)
- #define **YY_END_OF_BUFFER_CHAR** 0
- #define **YY_BUF_SIZE** 16384
- #define **YY_STATE_BUF_SIZE** ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))
- #define **YY_TYPEDEF_YY_BUFFER_STATE**
- #define **YY_TYPEDEF_YY_SIZE_T**
- #define **EOB_ACT_CONTINUE_SCAN** 0
- #define **EOB_ACT_END_OF_FILE** 1
- #define **EOB_ACT_LAST_MATCH** 2
- #define **YY_LESS_LINENO**(n)
- #define **yyless**(n)
- #define **unput**(c) yyunput(c, yyg->yytext_ptr , yyscanner)
- #define **YY_STRUCT_YY_BUFFER_STATE**
- #define **YY_BUFFER_NEW** 0
- #define **YY_BUFFER_NORMAL** 1
- #define **YY_BUFFER_EOF_PENDING** 2
- #define **YY_CURRENT_BUFFER**
- #define **YY_CURRENT_BUFFER_LVALUE** yyg->yy_buffer_stack[yyg->yy_buffer_stack_top]
- #define **YY_FLUSH_BUFFER** DSMatrixFlex_flush_buffer(YY_CURRENT_BUFFER ,yyscanner)
- #define **yy_new_buffer** DSMatrixFlex_create_buffer
- #define **yy_set_interactive**(is_interactive)
- #define **yy_set_bol**(at_bol)
- #define **YY_AT_BOL**() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
- #define **yytext_ptr** yytext_r
- #define **YY_DO_BEFORE_ACTION**
- #define **YY_NUM_RULES** 9
- #define **YY_END_OF_BUFFER** 10
- #define **REJECT** reject_used_but_not_detected
- #define **yymore**() yymore_used_but_not_detected
- #define **YY_MORE_ADJ** 0
- #define **YY_RESTORE_YY_MORE_OFFSET**
- #define **malloc**(x) DSSecureMalloc(x)
- #define **calloc**(x, y) DSSecureCalloc(x, y)
- #define **realloc**(x, y) DSSecureRealloc(x, y)
- #define **INITIAL** 0
- #define **YY_EXTRA_TYPE** struct matrix_token *
- #define **YY_READ_BUF_SIZE** 8192
- #define **ECHO** fwrite(yytext, yyleng, 1, yyout)
- #define **YY_INPUT**(buf, result, max_size)
- #define **yyterminate**() return YY_NULL
- #define **YY_START_STACK_INCR** 25
- #define **YY_FATAL_ERROR**(msg) yy_fatal_error(msg , yyscanner)

- #define **YY_DECL_IS_OURS** 1
- #define **YY_DECL** int DSMatrixFlexlex (yyscan_t yyscanner)
- #define **YY_USER_ACTION**
- #define **YY_BREAK** break;
- #define **YY_RULE_SETUP** YY_USER_ACTION
- #define **YY_EXIT_FAILURE** 2
- #define **yyless**(n)
- #define **YYTABLES_NAME** "yytables"

Typedefs

- typedef signed char **flex_int8_t**
- typedef short int **flex_int16_t**
- typedef int **flex_int32_t**
- typedef unsigned char **flex_uint8_t**
- typedef unsigned short int **flex_uint16_t**
- typedef unsigned int **flex_uint32_t**
- typedef void * **yyscan_t**
- typedef struct yy_buffer_state * **YY_BUFFER_STATE**
- typedef size_t **yy_size_t**
- typedef unsigned char **YY_CHAR**
- typedef int **yy_state_type**

Functions

- void **DSMatrixFlexrestart** (FILE *input_file, yyscan_t yyscanner)
- void **DSMatrixFlex_switch_to_buffer** (YY_BUFFER_STATE new_buffer, yyscan_t yyscanner)
- YY_BUFFER_STATE **DSMatrixFlex_create_buffer** (FILE *file, int size, yyscan_t yyscanner)
- void **DSMatrixFlex_delete_buffer** (YY_BUFFER_STATE b, yyscan_t yyscanner)
- void DSMatrixFlex_flush_buffer (YY_BUFFER_STATE b, yyscan_t yyscanner)
- void DSMatrixFlexpush_buffer_state (YY_BUFFER_STATE new_buffer, yyscan_t yyscanner)
- void DSMatrixFlexpop_buffer_state (yyscan_t yyscanner)
- YY_BUFFER_STATE DSMatrixFlex_scan_buffer (char *base, yy_size_t size, yyscan_t yyscanner)
- YY_BUFFER_STATE DSMatrixFlex_scan_string (yyconst char *yy_str, yyscan_t yyscanner)
- YY_BUFFER_STATE DSMatrixFlex_scan_bytes (yyconst char *bytes, yy_size_t len, yyscan_t yyscanner)
- void * **DSMatrixFlexalloc** (yy_size_t, yyscan_t yyscanner)
- void * **DSMatrixFlexrealloc** (void *, yy_size_t, yyscan_t yyscanner)
- void **DSMatrixFlexfree** (void *, yyscan_t yyscanner)
- int **DSMatrixFlexlex_init** (yyscan_t *scanner)
- int **DSMatrixFlexlex_init_extra** (YY_EXTRA_TYPE user_defined, yyscan_t *scanner)
- int **DSMatrixFlexlex_destroy** (yyscan_t yyscanner)
- int **DSMatrixFlexget_debug** (yyscan_t yyscanner)
- void **DSMatrixFlexset_debug** (int debug_flag, yyscan_t yyscanner)
- YY_EXTRA_TYPE DSMatrixFlexget_extra (yyscan_t yyscanner)
- void DSMatrixFlexset_extra (YY_EXTRA_TYPE user_defined, yyscan_t yyscanner)
- FILE * DSMatrixFlexget_in (yyscan_t yyscanner)
- void DSMatrixFlexset_in (FILE *in_str, yyscan_t yyscanner)
- FILE * DSMatrixFlexget_out (yyscan_t yyscanner)

- void **DSMatrixFlexset_out** (FILE *out_str, yyscan_t yyscanner)
- yy_size_t DSMatrixFlexget_leng (yyscan_t yyscanner)
- char * DSMatrixFlexget_text (yyscan_t yyscanner)
- int DSMatrixFlexget_lineno (yyscan_t yyscanner)
- void DSMatrixFlexset_lineno (int line_number, yyscan_t yyscanner)
- int **DSMatrixFlexwrap** (yyscan_t yyscanner)
- int **DSMatrixFlexlex** (yyscan_t yyscanner)
- int DSMatrixFlexget_column (yyscan_t yyscanner)
- void DSMatrixFlexset_column (int column_no, yyscan_t yyscanner)
- struct matrix_token * **DSMatrixTokenizeString** (const char *string)

7.21.1 Detailed Description

Implementation file with functions for tokenizing matrices, generated by flex. This file was generated directly by the flex program, and is the source code responsible for matrix tokenization. This file was generated by flex, according to a specification written by Jason Lomnitz. To generate this file, the following command must be executed: "flex -t DSMatrixGrammar.l > DSMatrixTokenizerLex.c".

Author

Jason Lomnitz.

Date

2011

7.21.2 Define Documentation

7.21.2.1 #define YY_CURRENT_BUFFER

Value:

```
( yyg->yy_buffer_stack \
                          ? yyg->yy_buffer_stack[yyg->yy_buffer_stack_top] \
                          : NULL)
```

7.21.2.2 #define YY_DO_BEFORE_ACTION

Value:

```
yyg->yytext_ptr = yy_bp; \
        yyleng = (size_t) (yy_cp - yy_bp); \
        yyg->yy_hold_char = *yy_cp; \
        *yy_cp = '\0'; \
        yyg->yy_c_buf_p = yy_cp;
```

7.21.2.3 #define YY_INPUT(buf, result, max_size)

Value:

```
if ( YY_CURRENT_BUFFER_LVALUE->yy_is_interactive ) \
                { \
                int c = '*'; \
                yy_size_t n; \
                for ( n = 0; n < max_size && \
                             (c = getc( yyin )) != EOF && c != '\n'; ++n ) \
                        buf[n] = (char) c; \
                if ( c == '\n' ) \
                        buf[n++] = (char) c; \
                if ( c == EOF && ferror( yyin ) ) \
                        YY_FATAL_ERROR( "input in flex scanner failed" ); \
                result = n; \
                } \
        else \
                { \
                errno=0; \
                while ( (result = fread(buf, 1, max_size, yyin))==0 && ferror(yyi
      n)) \
                        { \
                        if( errno != EINTR) \
                                { \
                                YY_FATAL_ERROR( "input in flex scanner failed" );
       \
                                break; \
                                } \
                        errno=0; \
                        clearerr(yyin); \
                        } \
                }\
\
```

7.21.2.4 #define yy_set_bol(at_bol)

Value:

```
{ \
        if ( ! YY_CURRENT_BUFFER ){\
        DSMatrixFlexensure_buffer_stack (yyscanner); \
                YY_CURRENT_BUFFER_LVALUE =    \
            DSMatrixFlex_create_buffer(yyin,YY_BUF_SIZE ,yyscanner); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
        }
```

7.21.2.5 #define yy_set_interactive(is_interactive)

Value:

```
{ \
```

```
        if ( ! YY_CURRENT_BUFFER ){ \
        DSMatrixFlexensure_buffer_stack (yyscanner); \
                YY_CURRENT_BUFFER_LVALUE =    \
            DSMatrixFlex_create_buffer(yyin,YY_BUF_SIZE ,yyscanner); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
        }
```

7.21.2.6 #define yyless(n)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                yytext[yyleng] = yyg->yy_hold_char; \
                yyg->yy_c_buf_p = yytext + yyless_macro_arg; \
                yyg->yy_hold_char = *yyg->yy_c_buf_p; \
                *yyg->yy_c_buf_p = '\0'; \
                yyleng = yyless_macro_arg; \
                } \
        while ( 0 )
```

7.21.2.7 #define yyless(n)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                *yy_cp = yyg->yy_hold_char; \
                YY_RESTORE_YY_MORE_OFFSET \
                yyg->yy_c_buf_p = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ;
       \
                YY_DO_BEFORE_ACTION; /* set up yytext again */ \
                } \
        while ( 0 )
```

7.21.3 Function Documentation

7.21.3.1 void DSMatrixFlex_flush_buffer (YY_BUFFER_STATE *b*, yyscan_t *yyscanner*)

Discard all buffered characters. On the next scan, YY_INPUT will be called.

Parameters

b the buffer state to be flushed, usually `YY_CURRENT_BUFFER`.

yyscanner The scanner object.

7.21.3.2 YY_BUFFER_STATE DSMatrixFlex_scan_buffer (char * *base*, yy_size_t *size*, yyscan_t *yyscanner*)

Setup the input buffer state to scan directly from a user-specified character buffer.

Parameters

base the character buffer

size the size in bytes of the character buffer

yyscanner The scanner object.

Returns

the newly allocated buffer state object.

7.21.3.3 YY_BUFFER_STATE DSMatrixFlex_scan_bytes (yyconst char * *yybytes*, yy_size_t *_yybytes_len*, yyscan_t *yyscanner*)

Setup the input buffer state to scan the given bytes. The next call to DSMatrixFlexlex() will scan from a *copy* of *bytes*.

Parameters

bytes the byte buffer to scan

len the number of bytes in the buffer pointed to by *bytes*.

yyscanner The scanner object.

Returns

the newly allocated buffer state object.

7.21.3.4 YY_BUFFER_STATE DSMatrixFlex_scan_string (yyconst char * *yystr*, yyscan_t *yyscanner*)

Setup the input buffer state to scan a string. The next call to DSMatrixFlexlex() will scan from a *copy* of *str*.

Parameters

yystr a NUL-terminated string to scan

yyscanner The scanner object.

Returns

the newly allocated buffer state object.

Note

If you want to scan bytes that may contain NUL values, then use DSMatrixFlex_scan_bytes() instead.

7.21.3.5 int DSMatrixFlexget_column (yyscan_t *yyscanner*)

Get the current column number.

Parameters

yyscanner The scanner object.

7.21.3.6 YY_EXTRA_TYPE DSMatrixFlexget_extra (yyscan_t *yyscanner*)

Get the user-defined data for this scanner.

Parameters

yyscanner The scanner object.

7.21.3.7 FILE * DSMatrixFlexget_in (yyscan_t *yyscanner*)

Get the input stream.

Parameters

yyscanner The scanner object.

7.21.3.8 yy_size_t DSMatrixFlexget_leng (yyscan_t *yyscanner*)

Get the length of the current token.

Parameters

yyscanner The scanner object.

7.21.3.9 int DSMatrixFlexget_lineno (yyscan_t *yyscanner*)

Get the current line number.

Parameters

yyscanner The scanner object.

7.21.3.10 FILE * DSMatrixFlexget_out (yyscan_t *yyscanner*)

Get the output stream.

Parameters

yyscanner The scanner object.

7.21.3.11 char * DSMatrixFlexget_text (yyscan_t *yyscanner*)

Get the current token.

Parameters

yyscanner The scanner object.

7.21.3.12 void DSMatrixFlexpop_buffer_state (yyscan_t *yyscanner*)

Removes and deletes the top of the stack, if present. The next element becomes the new top.

Parameters

yyscanner The scanner object.

7.21.3.13 void DSMatrixFlexpush_buffer_state (YY_BUFFER_STATE *new_buffer*, yyscan_t *yyscanner*)

Pushes the new state onto the stack. The new state becomes the current state. This function will allocate the stack if necessary.

Parameters

new_buffer The new state.

yyscanner The scanner object.

7.21.3.14 void DSMatrixFlexset_column (int *column_no*, yyscan_t *yyscanner*)

Set the current column.

Parameters

line_number

yyscanner The scanner object.

7.21.3.15 void DSMatrixFlexset_extra (YY_EXTRA_TYPE *user_defined*, yyscan_t *yyscanner*)

Set the user-defined data. This data is never touched by the scanner.

Parameters

user_defined The data to be associated with this scanner.

yyscanner The scanner object.

7.21.3.16 void DSMatrixFlexset_in (FILE * *in_str*, yyscan_t *yyscanner*)

Set the input stream. This does not discard the current input buffer.

Parameters

in_str A readable stream.

yyscanner The scanner object.

See also

DSMatrixFlex_switch_to_buffer

7.21.3.17 void DSMatrixFlexset_lineno (int *line_number*, yyscan_t *yyscanner*)

Set the current line number.

Parameters

line_number

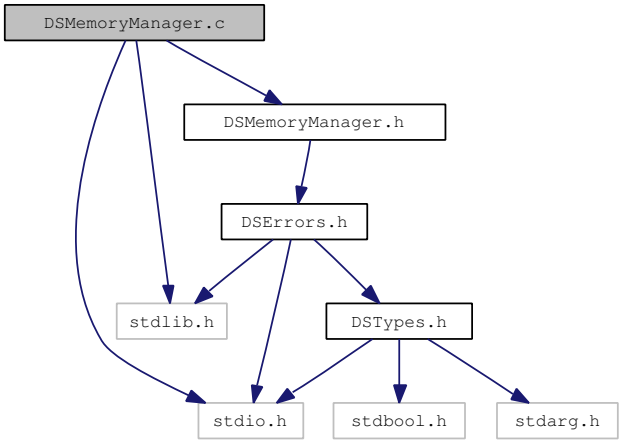
yyscanner The scanner object.

7.22 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 (size_t size)

  Function to securely allocate data using malloc.

- void * DSSecureCalloc (size_t count, size_t size)

  Function to securely allocate data using calloc.

- void * DSSecureRealloc (void *ptr, size_t size)

  Function to securely allocate data using realloc.

- void DSSecureFree (void *ptr)

  Function to securely free data.

7.22.1 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.

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

Author

Jason Lomnitz.

Date

2011

7.22.2 Function Documentation

7.22.2.1 void* DSSecureCalloc (size_t *count*, size_t *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.

7.22.2.2 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.

7.22.2.3 void* DSSecureMalloc (size_t *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.

7.22.2.4 void* DSSecureRealloc (void * *ptr*, size_t *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

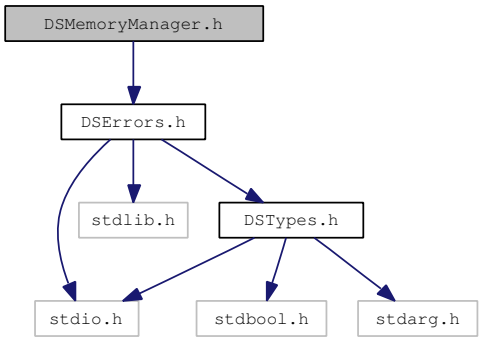
A pointer to the allocated data.

7.23 DSMemoryManager.h File Reference

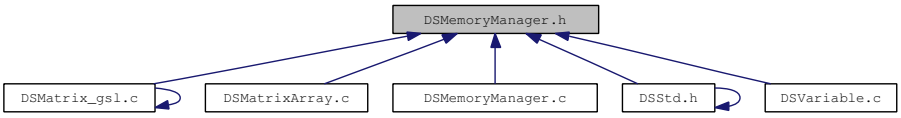
Header file with functions for secure memory allocation.

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

Include dependency graph for DSMemoryManager.h:



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



Functions

- void * DSSecureMalloc (size_t size)

  Function to securely allocate data using malloc.

- void * DSSecureCalloc (size_t count, size_t size)

  Function to securely allocate data using calloc.

- void * DSSecureRealloc (void *ptr, size_t size)

  Function to securely allocate data using realloc.

- void DSSecureFree (void *ptr)

  Function to securely free data.

7.23.1 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.

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

Author

Jason Lomnitz.

Date

2011

7.23.2 Function Documentation

7.23.2.1 void* DSSecureCalloc (size_t *count*, size_t *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.

7.23.2.2 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.

7.23.2.3 void* DSSecureMalloc (size_t *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.

7.23.2.4 void* DSSecureRealloc (void * *ptr*, size_t *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.

7.24 DSSSystem.h File Reference

Header file with functions for dealing with S-System.

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

Include dependency graph for DSSSystem.h:This graph shows which files directly or indirectly include this file:

Defines

- #define **M_DS_SSYS_NULL** M_DS_NULL ": S-System is NULL"

Functions

- void **DSSSystemFree** (DSSSystem *ssys)
- __deprecated DSSSystem * **DSSSystemFromGMAWithDominantTerms** (const DSGMASystem *gma, const DSUInteger *termList)
- DSSSystem * **DSSSystemWithTermsFromGMA** (const DSGMASystem *gma, const DSUInteger *termArray)
- DSSSystem * **DSSSystemByParsingStringList** (const DSVariablePool *const Xd, const char *const string,...)
- DSSSystem * **DSSSystemByParsingStrings** (const DSVariablePool *const Xd, char *const *const strings, const DSUInteger numberOfEquations)
- DSMatrix * **DSSSystemSteadyStateValues** (const DSSSystem *ssys)
- const DSUInteger **DSSSystemNumberOfEquations** (const DSSSystem *ssys)
- DSExpression ** **DSSSystemEquations** (const DSSSystem *ssys)
- DSExpression ** **DSSSystemSolution** (const DSSSystem *ssys)
- DSExpression ** **DSSSystemLogarithmicSolution** (const DSSSystem *ssys)
- const DSMatrix * **DSSSystemAlpha** (const DSSSystem *ssys)
- const DSMatrix * **DSSSystemBeta** (const DSSSystem *ssys)
- const DSMatrix * **DSSSystemGd** (const DSSSystem *ssys)
- const DSMatrix * **DSSSystemGi** (const DSSSystem *ssys)
- const DSMatrix * **DSSSystemHd** (const DSSSystem *ssys)
- const DSMatrix * **DSSSystemHi** (const DSSSystem *ssys)
- const DSMatrix * **DSSSystemM** (const DSSSystem *ssys)
- DSMatrix * **DSSSystemAi** (const DSSSystem *ssys)
- DSMatrix * **DSSSystemB** (const DSSSystem *ssys)
- const DSVariablePool * **DSSSystemXd** (const DSSSystem *const ssys)
- const DSVariablePool * **DSSSystemXi** (const DSSSystem *const ssys)
- const bool **DSSSystemHasSolution** (const DSSSystem *ssys)
- const bool **DSSSystemIsSingular** (const DSSSystem *ssys)
- void **DSSSystemPrint** (const DSSSystem *ssys)
- void **DSSSystemPrintEquations** (const DSSSystem *ssys)
- void **DSSSystemPrintSolution** (const DSSSystem *ssys)
- void **DSSSystemPrintLogarithmicSolution** (const DSSSystem *ssys)

7.24.1 Detailed Description

Header file with functions for dealing with S-System. 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

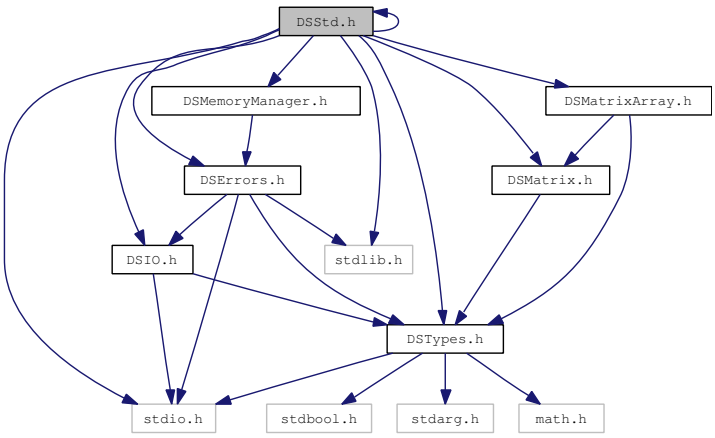
2011

7.25 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 "DSExpression.h"
#include "DSGMASystem.h"
#include "DSSSystem.h"
#include "DSCase.h"
#include "DSDesignSpace.h"
#include "DSVertices.h"
```

Include dependency graph for DSStd.h:



Defines

- #define **free**(x) DSSecureFree(x)
- #define **malloc**(x) DSSecureMalloc(x)
- #define **calloc**(x, y) DSSecureCalloc(x, y)
- #define **realloc**(x, y) DSSecureRealloc(x, y)

7.25.1 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).

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

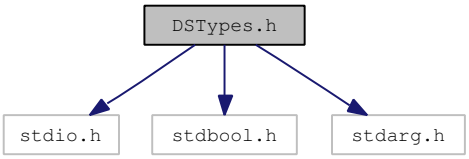
Add all previous functionality.
Add vertex enumeration functionality.

7.26 DSTypes.h File Reference

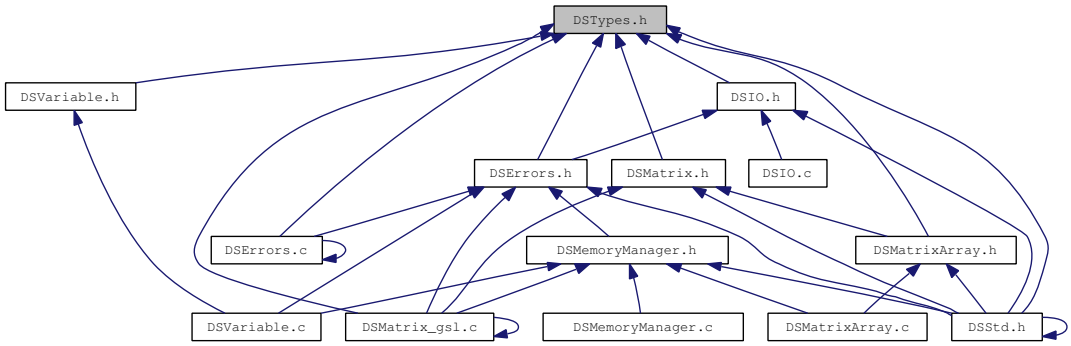
Header file with definitions for data types.

```
#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 DSVertices

  Data type that contains vertices of an N-Dimensional object.

- struct DSVariable

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

- struct _varDictionary

  Internal dictionary structure.

- struct DSVariablePool

  User-level variable pool.

- struct DSMatrix

  Data type representing a matrix.

- struct DSMatrixArray

  Data type representing an array of matrices.

- struct dsexpression

  Data type representing mathematical expressions.

- struct DSGMASystem

  Data type representing a GMA-System.

- struct DSSSystem

  Data type representing an S-System.

- struct DSCase

  Data type used to represent a case.

- struct DSDesignSpace

  Data type used to represent a design space/.

Defines

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

Typedefs

- typedef int **DSInteger**
- typedef unsigned int **DSUInteger**
- typedef struct dsexpression DSExpression

  Data type representing mathematical expressions.

Enumerations

- enum DSVariablePoolLock { DSLockReadWriteAdd, DSLockReadWrite, DSLockReadOnly, DSLockLocked }

  Data type used to lock different properties of the DSVariablePool.

7.26.1 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. Functions applying to these data types are contained elsewhere, and the individual data structures should refer to the respective files.

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

Author

Jason Lomnitz.

Date

2011

7.26.2 Typedef Documentation

7.26.2.1 typedef struct dsexpression DSExpression

Data type representing mathematical expressions.

This data type is the internal representation of matematical expressions. This data type is an Abstracts Syntax Tree with only three operators: '+', '$*$' and '^'. All other operators ('-' and '/') are represented by a combination of the former operators. The DSExpression automatically groups constant values, and reserves the first branch of the multiplication and addition operator for constant values. These operators can have any number of branches. The '^' operator can have two, and only two, branches.

Note

Functions are handled as variables with a single argument

See also

DSExpression.h
DSExpression.c

7.26.3 Enumeration Type Documentation

7.26.3.1 enum DSVariablePoolLock

Data type used to lock different properties of the DSVariablePool.

This data type enumerates the properties of the variable pool access rights. Its values indicate the different operations that can be taken with a variable pool, such as read/write/add, read/write and read.

See also

DSVariable.h
DSVariable.c

Enumerator:

DSLockReadWriteAdd The value of the Variable pool lock indicating read/write/add.

DSLockReadWrite The value of the Variable pool lock indicating read/write.

DSLockReadOnly The value of the Variable pool lock indicating read/.

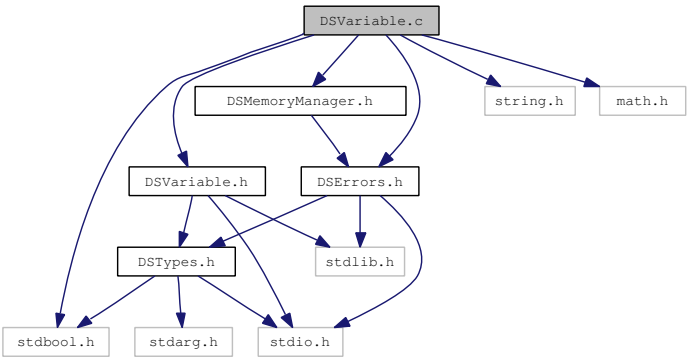
DSLockLocked The value of the Variable pool lock indicating no access.

7.27 DSVariable.c File Reference

Implementation file with functions for dealing with variables.

```
#include <stdbool.h>
#include <string.h>
#include <math.h>
#include <pthread.h>
#include "DSMemoryManager.h"
#include "DSErrors.h"
#include "DSVariable.h"
#include "DSVariableTokenizer.h"
#include "DSTypes.h"
#include "DSMatrix.h"
```

Include dependency graph for DSVariable.c:



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

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.

- DSVariablePool * DSVariablePoolAlloc (void)

  Creates a new DSVariablePool with an empty var dictionary.

- DSVariablePool * **DSVariablePoolCopy** (const DSVariablePool *const pool)
- void **DSVariablePoolFree** (DSVariablePool *pool)
- void **DSVariablePoolSetReadOnly** (DSVariablePool *pool)
- void **DSVariablePoolSetReadWrite** (DSVariablePool *pool)
- void **DSVariablePoolSetReadWriteAdd** (DSVariablePool *pool)
- bool **DSVariablePoolIsReadOnly** (const DSVariablePool *pool)
- bool **DSVariablePoolIsReadWrite** (const DSVariablePool *pool)
- bool **DSVariablePoolIsReadWriteAdd** (const DSVariablePool *pool)
- void **DSVariablePoolAddVariableWithName** (DSVariablePool *pool, const char *name)
- void **DSVariablePoolAddVariable** (DSVariablePool *pool, DSVariable *newVar)
- bool **DSVariablePoolHasVariableWithName** (const DSVariablePool *pool, const char *const name)
- DSVariable * **DSVariablePoolVariableWithName** (const DSVariablePool *pool, const char *name)
- void **DSVariablePoolSetValueForVariableWithName** (const DSVariablePool *pool, const char *name, const double value)
- const DSVariable ** **DSVariablePoolAllVariables** (const DSVariablePool *pool)
- const char ** **DSVariablePoolAllVariableNames** (const DSVariablePool *pool)
- DSUInteger **DSVariablePoolIndexOfVariable** (const DSVariablePool *pool, const DSVariable *var)
- DSUInteger **DSVariablePoolIndexOfVariableWithName** (const DSVariablePool *pool, const char *name)
- DSVariablePool * **DSVariablePoolByParsingString** (const char *string)
- void **DSVariablePoolPrint** (const DSVariablePool *const pool)
- DSMatrix * **DSVariablePoolValuesAsVector** (const DSVariablePool *pool, const bool rowVector)

Variables

- pthread_mutex_t **retaincount**

7.27.1 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).

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

7.27.2 Function Documentation

7.27.2.1 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

7.27.2.2 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.

7.27.2.3 DSVariablePool* DSVariablePoolAlloc (void)

Creates a new DSVariablePool with an empty var dictionary.

The variable pool is initialized with read/write privilages. The variable pool stores a indexed version of the variables added, as well as the order in which the variables were added. The order of the variables is kept to ensure a consistent variable index with system matrices of S-Systems and GMAs.

Returns

The pointer to the allocated DSVariablePool.

See also

DSVariablePoolFree

7.27.2.4 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

7.27.2.5 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

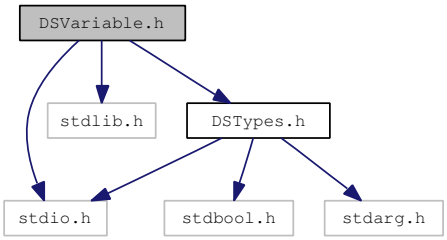
DSVariableRelease

7.28 DSVariable.h File Reference

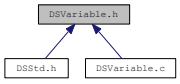
Header file with functions for dealing with variables.

```
#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)
- #define **DSVariableReturnValue**(x) DSVariableValue(x)
- #define DSVariableSetValue(x, y) ((x)->value = (y))

  Macro to set the value of a variable data structure.

- #define DSVariableValue(x) (((x) != NULL) ? (x)->value : NAN)

  Macro to get the value of a variable data structure.

- #define DSVariableName(x) ((x)->name)

  Macro to get the value of a variable data structure.

- #define M_DS_VAR_NULL M_DS_NULL ": Variable Pool is NULL"

  Error message indicating a NULL variable pool.

- #define M_DS_VAR_LOCKED " DSVariablePool: Insufficient priviliges"

  Error message indicating insufficient priviliges to manipulate a variable pool.

- #define **DSVariablePoolNumberOfVariables**(x) ((x)->numberOfVariables)
- #define **DSVariablePoolInternalDictionary**(x) ((x)->root)
- #define **DSVariablePoolVariableArray**(x) ((x)->variables)

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.

- DSVariablePool * DSVariablePoolAlloc (void)

  Creates a new DSVariablePool with an empty var dictionary.

- DSVariablePool * **DSVariablePoolCopy** (const DSVariablePool *const pool)
- void **DSVariablePoolFree** (DSVariablePool *pool)
- DSVariablePool * **DSVariablePoolByParsingString** (const char *string)
- void **DSVariablePoolSetReadOnly** (DSVariablePool *pool)
- void **DSVariablePoolSetReadWrite** (DSVariablePool *pool)
- void **DSVariablePoolSetReadWriteAdd** (DSVariablePool *pool)
- void **DSVariablePoolAddVariableWithName** (DSVariablePool *pool, const char *name)
- void **DSVariablePoolAddVariable** (DSVariablePool *pool, DSVariable *newVar)
- void **DSVariablePoolSetValueForVariableWithName** (const DSVariablePool *pool, const char *name, const double value)
- bool **DSVariablePoolIsReadOnly** (const DSVariablePool *pool)
- bool **DSVariablePoolIsReadWrite** (const DSVariablePool *pool)
- bool **DSVariablePoolIsReadWriteAdd** (const DSVariablePool *pool)
- bool **DSVariablePoolHasVariableWithName** (const DSVariablePool *pool, const char *const name)
- DSVariable * **DSVariablePoolVariableWithName** (const DSVariablePool *pool, const char *name)

- const DSVariable ** **DSVariablePoolAllVariables** (const DSVariablePool *pool)
- const char ** **DSVariablePoolAllVariableNames** (const DSVariablePool *pool)
- DSUInteger **DSVariablePoolIndexOfVariable** (const DSVariablePool *pool, const DSVariable *var)
- DSUInteger **DSVariablePoolIndexOfVariableWithName** (const DSVariablePool *pool, const char *name)
- void **DSVariablePoolPrint** (const DSVariablePool *const pool)
- DSMatrix * **DSVariablePoolValuesAsVector** (const DSVariablePool *pool, const bool rowVector)

7.28.1 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

7.28.2 Function Documentation

7.28.2.1 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

7.28.2.2 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.

7.28.2.3 DSVariablePool* DSVariablePoolAlloc (void)

Creates a new DSVariablePool with an empty var dictionary.

The variable pool is initialized with read/write privilages. The variable pool stores a indexed version of the variables added, as well as the order in which the variables were added. The order of the variables is kept to ensure a consistent variable index with system matrices of S-Systems and GMAs.

Returns

The pointer to the allocated DSVariablePool.

See also

DSVariablePoolFree

7.28.2.4 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

7.28.2.5 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

7.29 DSVariableTokenizer.c File Reference

Implementation file with functions for tokenizing with matrices.

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

```
#include "DSVariableTokenizer.h"
```

Include dependency graph for DSVariableTokenizer.c:

Functions

- struct variable_token * **DSVariableTokenAlloc** ()
- void **DSVariableTokenFree** (struct variable_token *root)
- void **DSVariableTokenSetString** (struct variable_token *root, char *string)
- void **DSVariableTokenSetDouble** (struct variable_token *root, double value)
- char * **DSVariableTokenString** (struct variable_token *root)
- double **DSVariableTokenDouble** (struct variable_token *root)

7.29.1 Detailed Description

Implementation file with functions for tokenizing with matrices. 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

7.30 DSVariableTokenizerLex.c File Reference

Implementation file with functions for tokenizing matrices, generated by flex.

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include "DSTypes.h"
#include "DSMemoryManager.h"
#include "DSVariable.h"
#include "DSVariableTokenizer.h"
#include <unistd.h>
```

Include dependency graph for DSVariableTokenizerLex.c:

Data Structures

- struct yy_buffer_state
- struct yy_trans_info
- struct yyguts_t

Defines

- #define **YY_INT_ALIGNED** short int
- #define **FLEX_SCANNER**
- #define **YY_FLEX_MAJOR_VERSION** 2
- #define **YY_FLEX_MINOR_VERSION** 5
- #define **YY_FLEX_SUBMINOR_VERSION** 35
- #define **INT16_MIN** (-32767-1)
- #define **INT32_MIN** (-2147483647-1)
- #define **INT8_MAX** (127)
- #define **INT16_MAX** (32767)
- #define **INT32_MAX** (2147483647)
- #define **UINT8_MAX** (255U)
- #define **UINT16_MAX** (65535U)
- #define **UINT32_MAX** (4294967295U)
- #define **yyconst**
- #define **YY_NULL** 0
- #define **YY_SC_TO_UI**(c) ((unsigned int) (unsigned char) c)
- #define **YY_TYPEDEF_YY_SCANNER_T**
- #define **yyin** yyg->yyin_r
- #define **yyout** yyg->yyout_r
- #define **yyextra** yyg->yyextra_r
- #define **yyleng** yyg->yyleng_r
- #define **yytext** yyg->yytext_r
- #define **yylineno** (YY_CURRENT_BUFFER_LVALUE->yy_bs_lineno)

- #define **yycolumn** (YY_CURRENT_BUFFER_LVALUE->yy_bs_column)
- #define **yy_flex_debug** yyg->yy_flex_debug_r
- #define **BEGIN** yyg->yy_start = 1 + 2 *
- #define **YY_START** ((yyg->yy_start - 1) / 2)
- #define **YYSTATE** YY_START
- #define **YY_STATE_EOF**(state) (YY_END_OF_BUFFER + state + 1)
- #define **YY_NEW_FILE** DSVariableFlexrestart(yyin ,yyscanner)
- #define **YY_END_OF_BUFFER_CHAR** 0
- #define **YY_BUF_SIZE** 16384
- #define **YY_STATE_BUF_SIZE** ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))
- #define **YY_TYPEDEF_YY_BUFFER_STATE**
- #define **YY_TYPEDEF_YY_SIZE_T**
- #define **EOB_ACT_CONTINUE_SCAN** 0
- #define **EOB_ACT_END_OF_FILE** 1
- #define **EOB_ACT_LAST_MATCH** 2
- #define **YY_LESS_LINENO**(n)
- #define **yyless**(n)
- #define **unput**(c) yyunput(c, yyg->yytext_ptr , yyscanner)
- #define **YY_STRUCT_YY_BUFFER_STATE**
- #define **YY_BUFFER_NEW** 0
- #define **YY_BUFFER_NORMAL** 1
- #define **YY_BUFFER_EOF_PENDING** 2
- #define **YY_CURRENT_BUFFER**
- #define **YY_CURRENT_BUFFER_LVALUE** yyg->yy_buffer_stack[yyg->yy_buffer_stack_top]

- #define **YY_FLUSH_BUFFER** DSVariableFlex_flush_buffer(YY_CURRENT_BUFFER ,yyscanner)
- #define **yy_new_buffer** DSVariableFlex_create_buffer
- #define **yy_set_interactive**(is_interactive)
- #define **yy_set_bol**(at_bol)
- #define **YY_AT_BOL**() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
- #define **yytext_ptr** yytext_r
- #define **YY_DO_BEFORE_ACTION**
- #define **YY_NUM_RULES** 14
- #define **YY_END_OF_BUFFER** 15
- #define **REJECT** reject_used_but_not_detected
- #define **yymore**() yymore_used_but_not_detected
- #define **YY_MORE_ADJ** 0
- #define **YY_RESTORE_YY_MORE_OFFSET**
- #define **malloc**(x) DSSecureMalloc(x)
- #define **calloc**(x, y) DSSecureCalloc(x, y)
- #define **realloc**(x, y) DSSecureRealloc(x, y)
- #define **INITIAL** 0
- #define **YY_EXTRA_TYPE** struct variable_token *
- #define **YY_READ_BUF_SIZE** 8192
- #define **ECHO** fwrite(yytext, yyleng, 1, yyout)
- #define **YY_INPUT**(buf, result, max_size)
- #define **yyterminate**() return YY_NULL
- #define **YY_START_STACK_INCR** 25
- #define **YY_FATAL_ERROR**(msg) yy_fatal_error(msg , yyscanner)

- #define **YY_DECL_IS_OURS** 1
- #define **YY_DECL** int DSVariableFlexlex (yyscan_t yyscanner)
- #define **YY_USER_ACTION**
- #define **YY_BREAK** break;
- #define **YY_RULE_SETUP** YY_USER_ACTION
- #define **YY_EXIT_FAILURE** 2
- #define **yyless**(n)
- #define **YYTABLES_NAME** "yytables"

Typedefs

- typedef signed char **flex_int8_t**
- typedef short int **flex_int16_t**
- typedef int **flex_int32_t**
- typedef unsigned char **flex_uint8_t**
- typedef unsigned short int **flex_uint16_t**
- typedef unsigned int **flex_uint32_t**
- typedef void * **yyscan_t**
- typedef struct yy_buffer_state * **YY_BUFFER_STATE**
- typedef size_t **yy_size_t**
- typedef unsigned char **YY_CHAR**
- typedef int **yy_state_type**

Functions

- void DSVariableFlexrestart (FILE *input_file, yyscan_t yyscanner)
- void DSVariableFlex_switch_to_buffer (YY_BUFFER_STATE new_buffer, yyscan_t yyscanner)
- YY_BUFFER_STATE DSVariableFlex_create_buffer (FILE *file, int size, yyscan_t yyscanner)
- void DSVariableFlex_delete_buffer (YY_BUFFER_STATE b, yyscan_t yyscanner)
- void DSVariableFlex_flush_buffer (YY_BUFFER_STATE b, yyscan_t yyscanner)
- void DSVariableFlexpush_buffer_state (YY_BUFFER_STATE new_buffer, yyscan_t yyscanner)
- void DSVariableFlexpop_buffer_state (yyscan_t yyscanner)
- YY_BUFFER_STATE DSVariableFlex_scan_buffer (char *base, yy_size_t size, yyscan_t yyscanner)
- YY_BUFFER_STATE DSVariableFlex_scan_string (yyconst char *yy_str, yyscan_t yyscanner)
- YY_BUFFER_STATE DSVariableFlex_scan_bytes (yyconst char *bytes, yy_size_t len, yyscan_t yyscanner)
- void * **DSVariableFlexalloc** (yy_size_t, yyscan_t yyscanner)
- void * **DSVariableFlexrealloc** (void *, yy_size_t, yyscan_t yyscanner)
- void **DSVariableFlexfree** (void *, yyscan_t yyscanner)
- int **DSVariableFlexlex_init** (yyscan_t *scanner)
- int **DSVariableFlexlex_init_extra** (YY_EXTRA_TYPE user_defined, yyscan_t *scanner)
- int **DSVariableFlexlex_destroy** (yyscan_t yyscanner)
- int **DSVariableFlexget_debug** (yyscan_t yyscanner)
- void **DSVariableFlexset_debug** (int debug_flag, yyscan_t yyscanner)
- YY_EXTRA_TYPE DSVariableFlexget_extra (yyscan_t yyscanner)
- void DSVariableFlexset_extra (YY_EXTRA_TYPE user_defined, yyscan_t yyscanner)
- FILE * DSVariableFlexget_in (yyscan_t yyscanner)
- void DSVariableFlexset_in (FILE *in_str, yyscan_t yyscanner)

- FILE * DSVariableFlexget_out (yyscan_t yyscanner)
- void **DSVariableFlexset_out** (FILE *out_str, yyscan_t yyscanner)
- yy_size_t DSVariableFlexget_leng (yyscan_t yyscanner)
- char * DSVariableFlexget_text (yyscan_t yyscanner)
- int DSVariableFlexget_lineno (yyscan_t yyscanner)
- void DSVariableFlexset_lineno (int line_number, yyscan_t yyscanner)
- int **DSVariableFlexwrap** (yyscan_t yyscanner)
- int **DSVariableFlexlex** (yyscan_t yyscanner)
- **if** (!yyg->yy_init)
- **while** (1)
- int **isatty** (int)
- int DSVariableFlexget_column (yyscan_t yyscanner)
- void DSVariableFlexset_column (int column_no, yyscan_t yyscanner)
- struct variable_token * **DSVariablePoolTokenizeString** (const char *string)

Variables

- YY_DECL register yy_state_type yy_current_state
- register char * **yy_cp**
- register char * **yy_bp**
- register int **yy_act**
- struct yyguts_t * **yyg** = (struct yyguts_t*)yyscanner

7.30.1 Detailed Description

Implementation file with functions for tokenizing matrices, generated by flex. This file was generated directly by the flex program, and is the source code responsible for matrix tokenization. This file was generated by flex, according to a specification written by Jason Lomnitz. To generate this file, the following command must be executed: "flex -t DSVariableGrammar.l > DSVariableTokenizerLex.c".

Author

Jason Lomnitz.

Date

2011

7.30.2 Define Documentation

7.30.2.1 #define YY_CURRENT_BUFFER

Value:

```
( yyg->yy_buffer_stack \
                          ? yyg->yy_buffer_stack[yyg->yy_buffer_stack_top] \
                          : NULL)
```

7.30.2.2 #define YY_DO_BEFORE_ACTION

Value:

```
yyg->yytext_ptr = yy_bp; \
        yyleng = (size_t) (yy_cp - yy_bp); \
        yyg->yy_hold_char = *yy_cp; \
        *yy_cp = '\0'; \
        yyg->yy_c_buf_p = yy_cp;
```

7.30.2.3 #define YY_INPUT(buf, result, max_size)

Value:

```
if ( YY_CURRENT_BUFFER_LVALUE->yy_is_interactive ) \
                { \
                int c = '*'; \
                yy_size_t n; \
                for ( n = 0; n < max_size && \
                             (c = getc( yyin )) != EOF && c != '\n'; ++n ) \
                        buf[n] = (char) c; \
                if ( c == '\n' ) \
                        buf[n++] = (char) c; \
                if ( c == EOF && ferror( yyin ) ) \
                        YY_FATAL_ERROR( "input in flex scanner failed" ); \
                result = n; \
                } \
        else \
                { \
                errno=0; \
                while ( (result = fread(buf, 1, max_size, yyin))==0 && ferror(yyi
      n)) \
                        { \
                        if( errno != EINTR) \
                                { \
                                YY_FATAL_ERROR( "input in flex scanner failed" );
       \
                                break; \
                                } \
                        errno=0; \
                        clearerr(yyin); \
                        } \
                }\
\
```

7.30.2.4 #define yy_set_bol(at_bol)

Value:

```
{ \
        if ( ! YY_CURRENT_BUFFER ){\
        DSVariableFlexensure_buffer_stack (yyscanner); \
                YY_CURRENT_BUFFER_LVALUE =    \
            DSVariableFlex_create_buffer(yyin,YY_BUF_SIZE ,yyscanner); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
        }
```

7.30.2.5 #define yy_set_interactive(is_interactive)

Value:

```
{ \
        if ( ! YY_CURRENT_BUFFER ){ \
        DSVariableFlexensure_buffer_stack (yyscanner); \
                YY_CURRENT_BUFFER_LVALUE =    \
            DSVariableFlex_create_buffer(yyin,YY_BUF_SIZE ,yyscanner); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
        }
```

7.30.2.6 #define yyless(n)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                yytext[yyleng] = yyg->yy_hold_char; \
                yyg->yy_c_buf_p = yytext + yyless_macro_arg; \
                yyg->yy_hold_char = *yyg->yy_c_buf_p; \
                *yyg->yy_c_buf_p = '\0'; \
                yyleng = yyless_macro_arg; \
                } \
        while ( 0 )
```

7.30.2.7 #define yyless(n)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                *yy_cp = yyg->yy_hold_char; \
                YY_RESTORE_YY_MORE_OFFSET \
                yyg->yy_c_buf_p = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ;
       \
                YY_DO_BEFORE_ACTION; /* set up yytext again */ \
                } \
        while ( 0 )
```

7.30.3 Function Documentation

7.30.3.1 YY_BUFFER_STATE DSVariableFlex_create_buffer (FILE * *file*, int *size*, yyscan_t *yyscanner*)

Allocate and initialize an input buffer state.

Parameters

file A readable stream.

size The character buffer size in bytes. When in doubt, use `YY_BUF_SIZE`.

yyscanner The scanner object.

Returns

the allocated buffer state.

7.30.3.2 void DSVariableFlex_delete_buffer (YY_BUFFER_STATE *b*, yyscan_t *yyscanner*)

Destroy the buffer.

Parameters

b a buffer created with DSVariableFlex_create_buffer()

yyscanner The scanner object.

7.30.3.3 void DSVariableFlex_flush_buffer (YY_BUFFER_STATE *b*, yyscan_t *yyscanner*)

Discard all buffered characters. On the next scan, YY_INPUT will be called.

Parameters

b the buffer state to be flushed, usually `YY_CURRENT_BUFFER`.

yyscanner The scanner object.

7.30.3.4 YY_BUFFER_STATE DSVariableFlex_scan_buffer (char * *base*, yy_size_t *size*, yyscan_t *yyscanner*)

Setup the input buffer state to scan directly from a user-specified character buffer.

Parameters

base the character buffer

size the size in bytes of the character buffer

yyscanner The scanner object.

Returns

the newly allocated buffer state object.

7.30.3.5 YY_BUFFER_STATE DSVariableFlex_scan_bytes (yyconst char * *yybytes*, yy_size_t *_yybytes_len*, yyscan_t *yyscanner*)

Setup the input buffer state to scan the given bytes. The next call to DSVariableFlexlex() will scan from a *copy* of *bytes*.

Parameters

bytes the byte buffer to scan

len the number of bytes in the buffer pointed to by *bytes*.

yyscanner The scanner object.

Returns

the newly allocated buffer state object.

7.30.3.6 YY_BUFFER_STATE DSVariableFlex_scan_string (yyconst char * *yystr*, yyscan_t *yyscanner*)

Setup the input buffer state to scan a string. The next call to DSVariableFlexlex() will scan from a *copy* of *str*.

Parameters

yystr a NUL-terminated string to scan

yyscanner The scanner object.

Returns

the newly allocated buffer state object.

Note

If you want to scan bytes that may contain NUL values, then use DSVariableFlex_scan_bytes() instead.

7.30.3.7 void DSVariableFlex_switch_to_buffer (YY_BUFFER_STATE *new_buffer*, yyscan_t *yyscanner*)

Switch to a different input buffer.

Parameters

new_buffer The new input buffer.

yyscanner The scanner object.

7.30.3.8 int DSVariableFlexget_column (yyscan_t *yyscanner*)

Get the current column number.

Parameters

yyscanner The scanner object.

7.30.3.9 YY_EXTRA_TYPE DSVariableFlexget_extra (yyscan_t *yyscanner*)

Get the user-defined data for this scanner.

Parameters

yyscanner The scanner object.

7.30.3.10 FILE * DSVariableFlexget_in (yyscan_t *yyscanner*)

Get the input stream.

Parameters

yyscanner The scanner object.

7.30.3.11 yy_size_t DSVariableFlexget_leng (yyscan_t *yyscanner*)

Get the length of the current token.

Parameters

yyscanner The scanner object.

7.30.3.12 int DSVariableFlexget_lineno (yyscan_t *yyscanner*)

Get the current line number.

Parameters

yyscanner The scanner object.

7.30.3.13 FILE * DSVariableFlexget_out (yyscan_t *yyscanner*)

Get the output stream.

Parameters

yyscanner The scanner object.

7.30.3.14 char * DSVariableFlexget_text (yyscan_t *yyscanner*)

Get the current token.

Parameters

yyscanner The scanner object.

7.30.3.15 void DSVariableFlexpop_buffer_state (yyscan_t *yyscanner*)

Removes and deletes the top of the stack, if present. The next element becomes the new top.

Parameters

yyscanner The scanner object.

7.30.3.16 void DSVariableFlexpush_buffer_state (YY_BUFFER_STATE *new_buffer*, yyscan_t *yyscanner*)

Pushes the new state onto the stack. The new state becomes the current state. This function will allocate the stack if necessary.

Parameters

new_buffer The new state.

yyscanner The scanner object.

7.30.3.17 void DSVariableFlexrestart (FILE * *input_file*, yyscan_t *yyscanner*)

Immediately switch to a different input stream.

Parameters

input_file A readable stream.

yyscanner The scanner object.

Note

This function does not reset the start condition to `INITIAL`.

7.30.3.18 void DSVariableFlexset_column (int *column_no*, yyscan_t *yyscanner*)

Set the current column.

Parameters

line_number

yyscanner The scanner object.

7.30.3.19 void DSVariableFlexset_extra (YY_EXTRA_TYPE *user_defined*, yyscan_t *yyscanner*)

Set the user-defined data. This data is never touched by the scanner.

Parameters

user_defined The data to be associated with this scanner.

yyscanner The scanner object.

7.30.3.20 void DSVariableFlexset_in (FILE * *in_str*, yyscan_t *yyscanner*)

Set the input stream. This does not discard the current input buffer.

Parameters

in_str A readable stream.

yyscanner The scanner object.

See also

DSVariableFlex_switch_to_buffer

7.30.3.21 void DSVariableFlexset_lineno (int *line_number*, yyscan_t *yyscanner*)

Set the current line number.

Parameters

line_number

yyscanner The scanner object.

7.30.4 Variable Documentation

7.30.4.1 YY_DECL register yy_state_type yy_current_state

The main scanner function which does all the work.

Index