

PPODESUITE Parser

0.1beta

Generated by Doxygen 1.8.4

Mon Mar 24 2014 09:39:39

Contents

1	Todo List	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Data Structure Documentation	7
4.1	Function Struct Reference	7
4.1.1	Detailed Description	7
4.1.2	Field Documentation	7
4.1.2.1	dx	7
4.1.2.2	j	8
4.1.2.3	neq	8
4.1.2.4	np	8
4.1.2.5	p	8
4.1.2.6	t	8
4.1.2.7	x	8
4.2	Node Struct Reference	8
4.2.1	Detailed Description	9
4.2.2	Field Documentation	9
4.2.2.1	children	9
4.2.2.2	ignore	9
4.2.2.3	iname	9
4.2.2.4	ival	10
4.2.2.5	next	10
4.2.2.6	parent	10
4.2.2.7	previous	10
4.2.2.8	tag	10
4.3	Variable Struct Reference	10
4.3.1	Detailed Description	11

4.3.2	Field Documentation	11
4.3.2.1	iname	11
4.3.2.2	next	11
4.3.2.3	previous	11
4.3.2.4	rel	11
4.3.2.5	type	11
4.3.2.6	zero	11
5	File Documentation	13
5.1	fortran.c File Reference	13
5.1.1	Macro Definition Documentation	14
5.1.1.1	_GNU_SOURCE	14
5.1.2	Function Documentation	14
5.1.2.1	F_and	14
5.1.2.2	F_arrayindex	14
5.1.2.3	F_assign	14
5.1.2.4	F_combine	14
5.1.2.5	F_div	14
5.1.2.6	F_elseif	14
5.1.2.7	F_eq_op	14
5.1.2.8	F_for	14
5.1.2.9	F_ge_op	14
5.1.2.10	F_gt_op	14
5.1.2.11	F_if	14
5.1.2.12	F_ifelse	14
5.1.2.13	F_ifelseif	14
5.1.2.14	F_ifelseifelse	14
5.1.2.15	F_indexrange	14
5.1.2.16	F_le_op	14
5.1.2.17	F_lt_op	14
5.1.2.18	F_minus	14
5.1.2.19	F_mul	14
5.1.2.20	F_ne_op	15
5.1.2.21	F_negative	15
5.1.2.22	F_not	15
5.1.2.23	F_or	15
5.1.2.24	F_plus	15
5.1.2.25	F_pow	15
5.1.2.26	F_range	15
5.1.2.27	F_while	15

5.1.2.28	F_zeros	15
5.1.2.29	functionToFortran	15
5.1.2.30	line	15
5.1.2.31	printFortranFunction	15
5.1.2.32	toFortran	15
5.2	fortran.h File Reference	15
5.2.1	Function Documentation	16
5.2.1.1	F_and	16
5.2.1.2	F_arrayindex	16
5.2.1.3	F_assign	16
5.2.1.4	F_combine	16
5.2.1.5	F_div	16
5.2.1.6	F_elseif	16
5.2.1.7	F_eq_op	16
5.2.1.8	F_for	16
5.2.1.9	F_ge_op	16
5.2.1.10	F_gt_op	16
5.2.1.11	F_if	16
5.2.1.12	F_ifelse	16
5.2.1.13	F_ifelseif	16
5.2.1.14	F_ifelseifelse	16
5.2.1.15	F_indexrange	16
5.2.1.16	F_le_op	16
5.2.1.17	F_lt_op	16
5.2.1.18	F_minus	16
5.2.1.19	F_mul	16
5.2.1.20	F_ne_op	16
5.2.1.21	F_negative	16
5.2.1.22	F_not	16
5.2.1.23	F_or	16
5.2.1.24	F_plus	17
5.2.1.25	F_pow	17
5.2.1.26	F_range	17
5.2.1.27	F_while	17
5.2.1.28	F_zeros	17
5.2.1.29	functionToFortran	17
5.2.1.30	line	17
5.2.1.31	printFortranFunction	17
5.2.1.32	toFortran	17
5.3	jacobian.c File Reference	17

5.3.1	Macro Definition Documentation	18
5.3.1.1	<code>_GNU_SOURCE</code>	18
5.3.2	Function Documentation	18
5.3.2.1	<code>D</code>	18
5.3.2.2	<code>D_arrayindex</code>	18
5.3.2.3	<code>D_assign</code>	18
5.3.2.4	<code>D_div</code>	18
5.3.2.5	<code>D_for</code>	18
5.3.2.6	<code>D_if</code>	18
5.3.2.7	<code>D_ifelse</code>	18
5.3.2.8	<code>D_minus</code>	18
5.3.2.9	<code>D_mul</code>	18
5.3.2.10	<code>D_negative</code>	18
5.3.2.11	<code>D_plus</code>	18
5.3.2.12	<code>D_pow</code>	18
5.3.2.13	<code>D_var</code>	18
5.3.2.14	<code>D_zeros</code>	18
5.3.2.15	<code>derivative</code>	18
5.3.2.16	<code>functionToJacobian</code>	18
5.4	<code>jacobian.h</code> File Reference	18
5.4.1	Function Documentation	19
5.4.1.1	<code>D</code>	19
5.4.1.2	<code>D_arrayindex</code>	19
5.4.1.3	<code>D_assign</code>	19
5.4.1.4	<code>D_div</code>	19
5.4.1.5	<code>D_for</code>	19
5.4.1.6	<code>D_if</code>	19
5.4.1.7	<code>D_ifelse</code>	19
5.4.1.8	<code>D_minus</code>	19
5.4.1.9	<code>D_mul</code>	19
5.4.1.10	<code>D_negative</code>	19
5.4.1.11	<code>D_plus</code>	19
5.4.1.12	<code>D_pow</code>	19
5.4.1.13	<code>D_var</code>	19
5.4.1.14	<code>D_zeros</code>	19
5.4.1.15	<code>derivative</code>	19
5.4.1.16	<code>functionToJacobian</code>	19
5.4.1.17	<code>main</code>	19
5.5	<code>node.c</code> File Reference	19
5.5.1	Function Documentation	20

5.5.1.1	appendChild	20
5.5.1.2	appendStatement	20
5.5.1.3	compareNodes	20
5.5.1.4	copyNode	21
5.5.1.5	createConstant	21
5.5.1.6	createOperation	21
5.5.1.7	createVariable	21
5.5.1.8	findVariable	22
5.5.1.9	last	22
5.5.1.10	removeNode	22
5.5.1.11	setIdentifier	22
5.6	node.h File Reference	22
5.6.1	Enumeration Type Documentation	23
5.6.1.1	NodeTag	23
5.6.2	Function Documentation	24
5.6.2.1	appendChild	24
5.6.2.2	appendStatement	24
5.6.2.3	compareNodes	25
5.6.2.4	copyNode	25
5.6.2.5	createConstant	25
5.6.2.6	createOperation	25
5.6.2.7	createVariable	26
5.6.2.8	findVariable	26
5.6.2.9	last	26
5.6.2.10	removeNode	26
5.6.2.11	setIdentifier	26
5.7	simplify.c File Reference	27
5.7.1	Macro Definition Documentation	27
5.7.1.1	_GNU_SOURCE	27
5.7.2	Function Documentation	27
5.7.2.1	getVariableZero	27
5.7.2.2	registerZeroVar	27
5.7.2.3	S_removePlusZero	27
5.7.2.4	S_replaceZeroAssignments	27
5.7.2.5	S_zeroAssignments	27
5.7.2.6	simplifyStructure	27
5.8	simplify.h File Reference	27
5.8.1	Enumeration Type Documentation	28
5.8.1.1	SimplifyState	28
5.8.2	Function Documentation	28

5.8.2.1	getVariableZero	28
5.8.2.2	registerZeroVar	28
5.8.2.3	S_removePlusZero	28
5.8.2.4	S_replaceZeroAssignments	28
5.8.2.5	S_zeroAssignments	28
5.8.2.6	simplifyStructure	28
5.8.3	Variable Documentation	28
5.8.3.1	simplifyStateSize	28
5.9	tree.c File Reference	28
5.9.1	Macro Definition Documentation	29
5.9.1.1	_GNU_SOURCE	29
5.9.2	Function Documentation	29
5.9.2.1	depth	29
5.9.2.2	emalloc	29
5.9.2.3	fatalError	29
5.9.2.4	getRelativeToY	29
5.9.2.5	print_tree	29
5.9.2.6	processDependentVectorIdentifier	30
5.9.2.7	processFunctionHeader	30
5.9.2.8	processIdentifier	30
5.9.2.9	registerVariable	30
5.10	tree.h File Reference	30
5.10.1	Macro Definition Documentation	31
5.10.1.1	MIN	31
5.10.2	Enumeration Type Documentation	31
5.10.2.1	VariableType	31
5.10.3	Function Documentation	31
5.10.3.1	emalloc	31
5.10.3.2	fatalError	31
5.10.3.3	getRelativeToY	31
5.10.3.4	print_tree	31
5.10.3.5	processDependentVectorIdentifier	31
5.10.3.6	processFunctionHeader	31
5.10.3.7	processIdentifier	31
5.10.3.8	registerVariable	31
5.10.4	Variable Documentation	31
5.10.4.1	func	32
5.10.4.2	labelcount	32
5.10.4.3	out	32
5.10.4.4	vars	32

5.10.4.5 warn	32
-------------------------	----

Index	33
--------------	-----------

Chapter 1

Todo List

Global `findVariable` (struct `Node` *n)

Describe this function

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Function	Datatype of a function definition	7
Node	A two dimensional node/tree data	8
Variable	Datatype of a variable	10

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

fortran.c	13
fortran.h	15
jacobian.c	17
jacobian.h	18
node.c	19
node.h	22
simplify.c	27
simplify.h	27
tree.c	28
tree.h	30

Chapter 4

Data Structure Documentation

4.1 Function Struct Reference

Datatype of a function definition.

```
#include <tree.h>
```

Data Fields

- char * **t**
Identifier of the independent variable.
- char * **x**
Identifier of the dependent variables.
- char * **p**
Identifier of the parameters.
- char * **dx**
Identifier of the differential ($dx = f(x(t))$).
- char * **neq**
Identifier of the Number of EQations.
- char * **np**
Identifier of the Number of Parameters.
- char * **j**
Identifier of the current Jacobian index.

4.1.1 Detailed Description

Datatype of a function definition.

Holds the names of the various arguments of the ODE function. In Matlab the ODE function should be defined as follows:

```
dx = function(t, x, p, neq, np)
```

4.1.2 Field Documentation

4.1.2.1 char* dx

Identifier of the differential ($dx = f(x(t))$).

4.1.2.2 char* j

Identifier of the current Jacobian index.

Note

Does not occur in the function definition.

4.1.2.3 char* neq

Identifier of the Number of EQations.

4.1.2.4 char* np

Identifier of the Number of Parameters.

4.1.2.5 char* p

Identifier of the parameters.

4.1.2.6 char* t

Identifier of the independent variable.

4.1.2.7 char* x

Identifier of the dependent variables.

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

- [tree.h](#)

4.2 Node Struct Reference

A two dimensional node/tree data.

```
#include <node.h>
```

Data Fields

- enum [NodeTag](#) tag
The tag of the [Node](#), specifying the type of the node.
- struct [Node](#) * next
Pointer to the next [Node](#).
- struct [Node](#) * previous
Pointer to the previous [Node](#).
- struct [Node](#) * children
Pointer to the first child [Node](#).
- struct [Node](#) * parent
Pointer to the previous [Node](#).
- double ival

Value of the number represented by this [Node](#).

- char * [iname](#)

The name of the identifier represented by this [Node](#).

- int [ignore](#)

Property that will make parsers ignore the current assignment if set to 1.

4.2.1 Detailed Description

A two dimensional node/tree data.

This data structure consists of nodes that specify their neighbours. Each node has a left neighbour, right neighbour, parent and children, resulting in a structure like this:

```
node-1
  |-> node-1.1
  |-> node-1.2
```

Where node-1 has no neighbours, but two [Node::children](#): node-1.1 and node-1.2. node-1.1 is positioned [Node::previous](#) of node-1.2, whereas node-1.2 is the [Node::next](#) node relative to node-1.1. Both node-1.1 and node-1.2 have node-1 as [Node::parent](#).

Each node also has a [Node::tag](#), defining the type of the node. Examples of tags are [TPLUS](#) and [TMUL](#), specifying a plus (+) or multiplication (*) operation respectively.

If the node specifies a number (i.e. the tag is [TNUM](#)), [Node::ival](#) will hold the value of this number. Every number is interpreted as a double precision number, since Matlab does not really distinguishes between integers and floating point numbers either.

The [Node::iname](#) property can be used in combination with the [TVAR](#), [TARRAYINDEX](#) or [TFOR](#) tags to specify the name of the identifier used.

The [Node::ignore](#) property can be used to ignore Matlab specific assignments that should not be translated to Fortran code.

See Also

[copyNode](#), [removeNode](#), [createOperation](#), [createConstant](#), [createVariable](#), [appendChild](#), [print_tree](#)

4.2.2 Field Documentation

4.2.2.1 struct [Node](#)* children

Pointer to the first child [Node](#).

All other children can be accessed using the [Node::next](#) properties of the children. Null when the [Node](#) does not have children.

4.2.2.2 int ignore

Property that will make parsers ignore the current assignment if set to 1.

4.2.2.3 char* iname

The name of the identifier represented by this [Node](#).

This property is only set when the tag of the [Node](#) is either [TVAR](#), [TARRAYINDEX](#) or [TFOR](#). In the case of [TVAR](#) and [TARRAYINDEX](#), this property specifies the name of the variable. When the tag is [TFOR](#), this property specifies the name of the variable that is used by the for statement.

4.2.2.4 double ival

Value of the number represented by this [Node](#).

This property is only set when the tag of the [Node](#) is [TNUM](#).

4.2.2.5 struct [Node](#)* next

Pointer to the next [Node](#).

NULL when the [Node](#) does not have a next neighbour.

4.2.2.6 struct [Node](#)* parent

Pointer to the previous [Node](#).

NULL when the [Node](#) is the topmost [Node](#).

4.2.2.7 struct [Node](#)* previous

Pointer to the previous [Node](#).

NULL when the [Node](#) does not have a previous neighbour.

4.2.2.8 enum [NodeTag](#) tag

The tag of the [Node](#), specifying the type of the node.

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

- [node.h](#)

4.3 Variable Struct Reference

Datatype of a variable.

```
#include <tree.h>
```

Data Fields

- enum [VariableType](#) type
Type of the variable.
- struct [Variable](#) * next
Pointer to the next variable in the linked list.
- struct [Variable](#) * previous
Pointer to the previous variable in the linked list.
- char * [iname](#)
Name/identifier of the variable.
- struct [Node](#) * rel
The relation of the variable to the independent variable of the ODE system.
- int [zero](#)
1 when this variable is always zero, 0 otherwise.

4.3.1 Detailed Description

Datatype of a variable.

This datatype can hold information about the type and name of the variable. It is also a linked list.

4.3.2 Field Documentation

4.3.2.1 `char* iname`

Name/identifier of the variable.

4.3.2.2 `struct Variable* next`

Pointer to the next variable in the linked list.

4.3.2.3 `struct Variable* previous`

Pointer to the previous variable in the linked list.

4.3.2.4 `struct Node* rel`

The relation of the variable to the independent variable of the ODE system.

4.3.2.5 `enum VariableType type`

Type of the variable.

4.3.2.6 `int zero`

1 when this variable is always zero, 0 otherwise.

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

- [tree.h](#)

Chapter 5

File Documentation

5.1 fortran.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "fortran.h"
#include "jacobian.h"
#include "tree.h"
#include "node.h"
```

Macros

- `#define _GNU_SOURCE`

Functions

- void [functionToFortran](#) (struct [Node](#) *t)
- void [printFortranFunction](#) (struct [Node](#) *t, int jac)
- char * [toFortran](#) (struct [Node](#) *t)
- char * [F_plus](#) (char *s1, char *s2)
- char * [F_minus](#) (char *s1, char *s2)
- char * [F_negative](#) (char *s1)
- char * [F_mul](#) (char *s1, char *s2)
- char * [F_div](#) (char *s1, char *s2)
- char * [F_pow](#) (char *s1, char *s2)
- char * [F_not](#) (char *s1)
- char * [F_or](#) (char *s1, char *s2)
- char * [F_and](#) (char *s1, char *s2)
- char * [F_eq_op](#) (char *s1, char *s2)
- char * [F_ne_op](#) (char *s1, char *s2)
- char * [F_gt_op](#) (char *s1, char *s2)
- char * [F_ge_op](#) (char *s1, char *s2)
- char * [F_lt_op](#) (char *s1, char *s2)
- char * [F_le_op](#) (char *s1, char *s2)
- char * [F_assign](#) (char *s1, char *s2)
- char * [F_for](#) (char *s1, char *s2, char *s3)
- char * [F_range](#) (char *s1, char *s2, char *s3)

- char * [F_while](#) (char *s1, char *s2)
- char * [F_if](#) (char *s1, char *s2)
- char * [F_ifelse](#) (char *s1, char *s2, char *s3)
- char * [F_ifelseif](#) (char *s1, char *s2, char *s3)
- char * [F_elseif](#) (char *s1, char *s2)
- char * [F_ifelseifelse](#) (char *s1, char *s2, char *s3, char *s4)
- char * [F_arrayindex](#) (char *s1, char *s2)
- char * [F_indexrange](#) (char *s1, char *s2)
- char * [F_zeros](#) (char *s1, char *s2, int allocate)
- char * [F_combine](#) (char *s1, char *s2)
- char * [line](#) (int label, char *content)

5.1.1 Macro Definition Documentation

5.1.1.1 #define _GNU_SOURCE

5.1.2 Function Documentation

5.1.2.1 char* F_and (char * s1, char * s2)

5.1.2.2 char* F_arrayindex (char * s1, char * s2)

5.1.2.3 char* F_assign (char * s1, char * s2)

5.1.2.4 char* F_combine (char * s1, char * s2)

5.1.2.5 char* F_div (char * s1, char * s2)

5.1.2.6 char* F_elseif (char * s1, char * s2)

5.1.2.7 char* F_eq_op (char * s1, char * s2)

5.1.2.8 char* F_for (char * s1, char * s2, char * s3)

5.1.2.9 char* F_ge_op (char * s1, char * s2)

5.1.2.10 char* F_gt_op (char * s1, char * s2)

5.1.2.11 char* F_if (char * s1, char * s2)

5.1.2.12 char* F_ifelse (char * s1, char * s2, char * s3)

5.1.2.13 char* F_ifelseif (char * s1, char * s2, char * s3)

5.1.2.14 char* F_ifelseifelse (char * s1, char * s2, char * s3, char * s4)

5.1.2.15 char* F_indexrange (char * s1, char * s2)

5.1.2.16 char* F_le_op (char * s1, char * s2)

5.1.2.17 char* F_lt_op (char * s1, char * s2)

5.1.2.18 char* F_minus (char * s1, char * s2)

5.1.2.19 char* F_mul (char * s1, char * s2)

- 5.1.2.20 char* F_ne_op (char * s1, char * s2)
- 5.1.2.21 char* F_negative (char * s1)
- 5.1.2.22 char* F_not (char * s1)
- 5.1.2.23 char* F_or (char * s1, char * s2)
- 5.1.2.24 char* F_plus (char * s1, char * s2)
- 5.1.2.25 char* F_pow (char * s1, char * s2)
- 5.1.2.26 char* F_range (char * s1, char * s2, char * s3)
- 5.1.2.27 char* F_while (char * s1, char * s2)
- 5.1.2.28 char* F_zeros (char * s1, char * s2, int allocate)
- 5.1.2.29 void functionToFortran (struct Node * t)
- 5.1.2.30 char* line (int label, char * content)
- 5.1.2.31 void printFortranFunction (struct Node * t, int jac)
- 5.1.2.32 char* toFortran (struct Node * t)

5.2 fortran.h File Reference

Functions

- char * toFortran (struct Node *t)
- void functionToFortran (struct Node *t)
- void printFortranFunction (struct Node *t, int jac)
- char * line (int label, char *content)
- char * F_plus (char *s1, char *s2)
- char * F_minus (char *s1, char *s2)
- char * F_negative (char *s1)
- char * F_mul (char *s1, char *s2)
- char * F_div (char *s1, char *s2)
- char * F_pow (char *s1, char *s2)
- char * F_not (char *s1)
- char * F_eq_op (char *s1, char *s2)
- char * F_ne_op (char *s1, char *s2)
- char * F_gt_op (char *s1, char *s2)
- char * F_ge_op (char *s1, char *s2)
- char * F_lt_op (char *s1, char *s2)
- char * F_le_op (char *s1, char *s2)
- char * F_assign (char *s1, char *s2)
- char * F_for (char *s1, char *s2, char *s3)
- char * F_range (char *s1, char *s2, char *s3)
- char * F_while (char *s1, char *s2)
- char * F_and (char *s1, char *s2)
- char * F_or (char *s1, char *s2)
- char * F_if (char *s1, char *s2)
- char * F_ifelse (char *s1, char *s2, char *s3)

- char * [F_ifelseif](#) (char *s1, char *s2, char *s3)
- char * [F_elseif](#) (char *s1, char *s2)
- char * [F_ifelseifelse](#) (char *s1, char *s2, char *s3, char *s4)
- char * [F_arrayindex](#) (char *s1, char *s2)
- char * [F_indexrange](#) (char *s1, char *s2)
- char * [F_zeros](#) (char *s1, char *s2, int allocate)
- char * [F_combine](#) (char *s1, char *s2)

5.2.1 Function Documentation

5.2.1.1 char* [F_and](#) (char * s1, char * s2)

5.2.1.2 char* [F_arrayindex](#) (char * s1, char * s2)

5.2.1.3 char* [F_assign](#) (char * s1, char * s2)

5.2.1.4 char* [F_combine](#) (char * s1, char * s2)

5.2.1.5 char* [F_div](#) (char * s1, char * s2)

5.2.1.6 char* [F_elseif](#) (char * s1, char * s2)

5.2.1.7 char* [F_eq_op](#) (char * s1, char * s2)

5.2.1.8 char* [F_for](#) (char * s1, char * s2, char * s3)

5.2.1.9 char* [F_ge_op](#) (char * s1, char * s2)

5.2.1.10 char* [F_gt_op](#) (char * s1, char * s2)

5.2.1.11 char* [F_if](#) (char * s1, char * s2)

5.2.1.12 char* [F_ifelse](#) (char * s1, char * s2, char * s3)

5.2.1.13 char* [F_ifelseif](#) (char * s1, char * s2, char * s3)

5.2.1.14 char* [F_ifelseifelse](#) (char * s1, char * s2, char * s3, char * s4)

5.2.1.15 char* [F_indexrange](#) (char * s1, char * s2)

5.2.1.16 char* [F_le_op](#) (char * s1, char * s2)

5.2.1.17 char* [F_lt_op](#) (char * s1, char * s2)

5.2.1.18 char* [F_minus](#) (char * s1, char * s2)

5.2.1.19 char* [F_mul](#) (char * s1, char * s2)

5.2.1.20 char* [F_ne_op](#) (char * s1, char * s2)

5.2.1.21 char* [F_negative](#) (char * s1)

5.2.1.22 char* [F_not](#) (char * s1)

5.2.1.23 char* [F_or](#) (char * s1, char * s2)

5.2.1.24 `char* F_plus (char * s1, char * s2)`

5.2.1.25 `char* F_pow (char * s1, char * s2)`

5.2.1.26 `char* F_range (char * s1, char * s2, char * s3)`

5.2.1.27 `char* F_while (char * s1, char * s2)`

5.2.1.28 `char* F_zeros (char * s1, char * s2, int allocate)`

5.2.1.29 `void functionToFortran (struct Node * t)`

5.2.1.30 `char* line (int label, char * content)`

5.2.1.31 `void printFortranFunction (struct Node * t, int jac)`

5.2.1.32 `char* toFortran (struct Node * t)`

5.3 jacobian.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "jacobian.h"
#include "simplify.h"
#include "fortran.h"
#include "tree.h"
#include "node.h"
```

Macros

- `#define _GNU_SOURCE`

Functions

- `char * D (char *name)`
- `void functionToJacobian (struct Node *t)`
- `struct Node * derivative (struct Node *n, struct Node *j)`
- `struct Node * D_zeros (char *iden, struct Node *n)`
- `struct Node * D_plus (struct Node *n1, struct Node *n2, struct Node *j)`
- `struct Node * D_minus (struct Node *n1, struct Node *n2, struct Node *j)`
- `struct Node * D_negative (struct Node *n1, struct Node *j)`
- `struct Node * D_mul (struct Node *n1, struct Node *n2, struct Node *j)`
- `struct Node * D_div (struct Node *n1, struct Node *n2, struct Node *j)`
- `struct Node * D_pow (struct Node *n1, struct Node *n2, struct Node *j)`
- `struct Node * D_var (struct Node *n)`
- `struct Node * D_assign (struct Node *n1, struct Node *n2)`
- `struct Node * D_if (struct Node *n1, struct Node *n2, struct Node *j)`
- `struct Node * D_ifelse (struct Node *n1, struct Node *n2, struct Node *n3, struct Node *j)`
- `struct Node * D_arrayindex (char *iden, struct Node *n)`
- `struct Node * D_for (char *iden, struct Node *n1, struct Node *n2)`

5.3.1 Macro Definition Documentation

5.3.1.1 `#define _GNU_SOURCE`

5.3.2 Function Documentation

5.3.2.1 `char* D (char * name)`

5.3.2.2 `struct Node* D_arrayindex (char * iden, struct Node * n)`

5.3.2.3 `struct Node* D_assign (struct Node * n1, struct Node * n2)`

5.3.2.4 `struct Node* D_div (struct Node * n1, struct Node * n2, struct Node * j)`

5.3.2.5 `struct Node* D_for (char * iden, struct Node * n1, struct Node * n2)`

5.3.2.6 `struct Node* D_if (struct Node * n1, struct Node * n2, struct Node * j)`

5.3.2.7 `struct Node* D_ifelse (struct Node * n1, struct Node * n2, struct Node * n3, struct Node * j)`

5.3.2.8 `struct Node* D_minus (struct Node * n1, struct Node * n2, struct Node * j)`

5.3.2.9 `struct Node* D_mul (struct Node * n1, struct Node * n2, struct Node * j)`

5.3.2.10 `struct Node* D_negative (struct Node * n1, struct Node * j)`

5.3.2.11 `struct Node* D_plus (struct Node * n1, struct Node * n2, struct Node * j)`

5.3.2.12 `struct Node* D_pow (struct Node * n1, struct Node * n2, struct Node * j)`

5.3.2.13 `struct Node* D_var (struct Node * n)`

5.3.2.14 `struct Node* D_zeros (char * iden, struct Node * n)`

5.3.2.15 `struct Node* derivative (struct Node * n, struct Node * j)`

5.3.2.16 `void functionToJacobian (struct Node * t)`

5.4 jacobian.h File Reference

Functions

- void [functionToJacobian](#) (struct [Node](#) *t)
- struct [Node](#) * [derivative](#) (struct [Node](#) *n, struct [Node](#) *j)
- char * [D](#) (char *name)
- struct [Node](#) * [D_plus](#) (struct [Node](#) *n1, struct [Node](#) *n2, struct [Node](#) *j)
- struct [Node](#) * [D_minus](#) (struct [Node](#) *n1, struct [Node](#) *n2, struct [Node](#) *j)
- struct [Node](#) * [D_negative](#) (struct [Node](#) *n1, struct [Node](#) *j)
- struct [Node](#) * [D_mul](#) (struct [Node](#) *n1, struct [Node](#) *n2, struct [Node](#) *j)
- struct [Node](#) * [D_div](#) (struct [Node](#) *n1, struct [Node](#) *n2, struct [Node](#) *j)
- struct [Node](#) * [D_pow](#) (struct [Node](#) *n1, struct [Node](#) *n2, struct [Node](#) *j)
- struct [Node](#) * [D_var](#) (struct [Node](#) *n)
- struct [Node](#) * [D_assign](#) (struct [Node](#) *n1, struct [Node](#) *n2)
- struct [Node](#) * [D_if](#) (struct [Node](#) *n1, struct [Node](#) *n2, struct [Node](#) *j)
- struct [Node](#) * [D_ifelse](#) (struct [Node](#) *n1, struct [Node](#) *n2, struct [Node](#) *n3, struct [Node](#) *j)

- struct [Node](#) * [D_for](#) (char *iden, struct [Node](#) *n1, struct [Node](#) *n2)
- struct [Node](#) * [D_zeros](#) (char *iden, struct [Node](#) *n)
- struct [Node](#) * [D_arrayindex](#) (char *iden, struct [Node](#) *n)
- int [main](#) (void)

5.4.1 Function Documentation

5.4.1.1 char* [D](#) (char * *name*)

5.4.1.2 struct [Node](#)* [D_arrayindex](#) (char * *iden*, struct [Node](#) * *n*)

5.4.1.3 struct [Node](#)* [D_assign](#) (struct [Node](#) * *n1*, struct [Node](#) * *n2*)

5.4.1.4 struct [Node](#)* [D_div](#) (struct [Node](#) * *n1*, struct [Node](#) * *n2*, struct [Node](#) * *j*)

5.4.1.5 struct [Node](#)* [D_for](#) (char * *iden*, struct [Node](#) * *n1*, struct [Node](#) * *n2*)

5.4.1.6 struct [Node](#)* [D_if](#) (struct [Node](#) * *n1*, struct [Node](#) * *n2*, struct [Node](#) * *j*)

5.4.1.7 struct [Node](#)* [D_ifelse](#) (struct [Node](#) * *n1*, struct [Node](#) * *n2*, struct [Node](#) * *n3*, struct [Node](#) * *j*)

5.4.1.8 struct [Node](#)* [D_minus](#) (struct [Node](#) * *n1*, struct [Node](#) * *n2*, struct [Node](#) * *j*)

5.4.1.9 struct [Node](#)* [D_mul](#) (struct [Node](#) * *n1*, struct [Node](#) * *n2*, struct [Node](#) * *j*)

5.4.1.10 struct [Node](#)* [D_negative](#) (struct [Node](#) * *n1*, struct [Node](#) * *j*)

5.4.1.11 struct [Node](#)* [D_plus](#) (struct [Node](#) * *n1*, struct [Node](#) * *n2*, struct [Node](#) * *j*)

5.4.1.12 struct [Node](#)* [D_pow](#) (struct [Node](#) * *n1*, struct [Node](#) * *n2*, struct [Node](#) * *j*)

5.4.1.13 struct [Node](#)* [D_var](#) (struct [Node](#) * *n*)

5.4.1.14 struct [Node](#)* [D_zeros](#) (char * *iden*, struct [Node](#) * *n*)

5.4.1.15 struct [Node](#)* [derivative](#) (struct [Node](#) * *n*, struct [Node](#) * *j*)

5.4.1.16 void [functionToJacobian](#) (struct [Node](#) * *t*)

5.4.1.17 int [main](#) (void)

5.5 node.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "node.h"
#include "tree.h"
```

Functions

- struct [Node](#) * [last](#) (struct [Node](#) *t)

Find the last [Node](#) on the same level as the specified [Node](#).

- struct [Node](#) * [createOperation](#) (enum [NodeTag](#) op)
Create a [Node](#) of the specified operation type.
- void [appendChild](#) (struct [Node](#) *parent, struct [Node](#) *child)
Append a child [Node](#) to a parent [Node](#).
- void [setIdentifier](#) (struct [Node](#) *node, char *identifier)
Set the identifier of a [Node](#).
- struct [Node](#) * [createConstant](#) (double num)
Create a [Node](#) representing a constant number.
- struct [Node](#) * [createVariable](#) (char *varname)
Create a [Node](#) representing a variable.
- struct [Node](#) * [appendStatement](#) (struct [Node](#) *prevnodes, struct [Node](#) *newnode)
Append a statement to a list of statements.
- struct [Node](#) * [copyNode](#) (struct [Node](#) *n)
Copies the data of a [Node](#) to a new [Node](#) instance.
- struct [Node](#) * [findVariable](#) (struct [Node](#) *n)
- int [compareNodes](#) (struct [Node](#) *n1, struct [Node](#) *n2)
Compare two Nodes and their children to check whether they are equivalent.
- void [removeNode](#) (struct [Node](#) *n)
Remove an [Node](#) instance and free the memory it used.

5.5.1 Function Documentation

5.5.1.1 void [appendChild](#) (struct [Node](#) * *parent*, struct [Node](#) * *child*)

Append a child [Node](#) to a parent [Node](#).

Append the [Node](#) *child* to the list of children of [Node](#) *parent* and sets *parent* as the parent [Node](#) of *child*.

Parameters

<i>parent</i>	Node where the <i>child</i> Node is to be appended to.
<i>child</i>	Node to be appended to the children of <i>parent</i> .

5.5.1.2 struct [Node](#)* [appendStatement](#) (struct [Node](#) * *prevnodes*, struct [Node](#) * *newnode*)

Append a statement to a list of statements.

Appends the statement *newnode* to the list of statements *prevnodes*. *prevnodes* does not have to point at the last [Node](#) of that level. This function will find the last [Node](#) of the same level and append the *newnode* there.

Parameters

<i>prevnodes</i>	Pointer to a Node on which level the Node <i>newnode</i> should be added.
<i>newnode</i>	Node to add to the linked list.

Returns

Pointer to a [Node](#) of the level where *newnode* was added.

5.5.1.3 int [compareNodes](#) (struct [Node](#) * *n1*, struct [Node](#) * *n2*)

Compare two Nodes and their children to check whether they are equivalent.

Compare the Nodes *n1* and *n2* and their children to check whether they are equivalent. All properties of the [Node](#) struct are compared, just like the complete structure the Nodes represent.

Parameters

<i>n1</i>	First Node .
<i>n2</i>	Second Node .

Returns

1 when the first and second [Node](#) are equivalent, 0 otherwise.

5.5.1.4 struct Node* copyNode (struct Node * *n*)

Copies the data of a [Node](#) to a new [Node](#) instance.

Copy the contents of [Node](#) *n* to a new [Node](#) and recursively does the same to all children. The neighbours of [Node](#) *n* will not be copied.

Parameters

<i>n</i>	Node to copy.
----------	-------------------------------

Returns

Pointer to the new [Node](#) with the same data as *n*.

5.5.1.5 struct Node* createConstant (double *num*)

Create a [Node](#) representing a constant number.

Create a [Node](#) representing the number *num*.

Parameters

<i>num</i>	The value of the constant.
------------	----------------------------

Returns

Pointer to the created [Node](#) instance.

5.5.1.6 struct Node* createOperation (enum NodeTag *op*)

Create a [Node](#) of the specified operation type.

Create a [Node](#) of the specified operation, whilst setting all other properties to zero or NULL.

Parameters

<i>op</i>	Operation type of the Node to be created.
-----------	---

Returns

The created [Node](#) instance.

5.5.1.7 struct Node* createVariable (char * *varname*)

Create a [Node](#) representing a variable.

Create a [Node](#) representing a variable with the name *varname*.

Parameters

<i>varname</i>	The name of the variable.
----------------	---------------------------

Returns

Pointer to the created [Node](#) instance.

5.5.1.8 `struct Node* findVariable (struct Node * n)`

Todo Describe this function

5.5.1.9 `struct Node* last (struct Node * t)`

Find the last [Node](#) on the same level as the specified [Node](#).

Search the last [Node](#) on the same level as [Node](#) *t*.

Parameters

<i>t</i>	Node of which to find the last neighbour/peer.
----------	--

Returns

The last [Node](#) on the same level as [Node](#) *t*.

5.5.1.10 `void removeNode (struct Node * n)`

Remove an [Node](#) instance and free the memory it used.

Removes [Node](#) *n* and all its children.

Parameters

<i>n</i>	Node to remove.
----------	---------------------------------

5.5.1.11 `void setIdentifier (struct Node * node, char * identifier)`

Set the identifier of a [Node](#).

Set the identifier ([Node::iname](#)) of [Node](#) *node* to *identifier*.

Parameters

<i>node</i>	Node to set the identifier of.
<i>identifier</i>	Name of the identifier.

5.6 node.h File Reference

Data Structures

- [struct Node](#)

A two dimensional node/tree data.

Enumerations

- enum `NodeTag` {
`TPLUS`, `TMINUS`, `TMUL`, `TDIV`,
`TPOW`, `TNUM`, `TVAR`, `TFOR`,
`TWHILE`, `TASSIGN`, `TRANGE`, `TOR`,
`TAND`, `TEQ_OP`, `TNE_OP`, `TGT_OP`,
`TLT_OP`, `TGE_OP`, `TLE_OP`, `TIF`,
`TIFELSE`, `TIFELSEIF`, `TELSEIF`, `TIFELSEIFELSE`,
`TARRAYINDEX`, `TLIST`, `TFUNCDEC`, `TMISC`,
`TCOMBINE`, `TNOT`, `TNEGATIVE` }

Tags the `Node` struct uses to specify its type or purpose.

Functions

- struct `Node` * `createOperation` (enum `NodeTag` op)
Create a `Node` of the specified operation type.
- void `appendChild` (struct `Node` *parent, struct `Node` *child)
Append a child `Node` to a parent `Node`.
- void `setIdentifier` (struct `Node` *node, char *identifier)
Set the identifier of a `Node`.
- struct `Node` * `last` (struct `Node` *t)
Find the last `Node` on the same level as the specified `Node`.
- struct `Node` * `createConstant` (double num)
Create a `Node` representing a constant number.
- struct `Node` * `createVariable` (char *varname)
Create a `Node` representing a variable.
- struct `Node` * `appendStatement` (struct `Node` *prevnodes, struct `Node` *newnode)
Append a statement to a list of statements.
- struct `Node` * `copyNode` (struct `Node` *n)
Copies the data of a `Node` to a new `Node` instance.
- void `removeNode` (struct `Node` *n)
Remove an `Node` instance and free the memory it used.
- struct `Node` * `findVariable` (struct `Node` *n)
- int `compareNodes` (struct `Node` *n1, struct `Node` *n2)
Compare two `Nodes` and their children to check whether they are equivalent.

5.6.1 Enumeration Type Documentation

5.6.1.1 enum `NodeTag`

Tags the `Node` struct uses to specify its type or purpose.

Enumerator

- `TPLUS`** Plus operation (a+b), two children.
- `TMINUS`** Minus operation (a-b), two children.
- `TMUL`** Multiplication operation (a*b), two children.
- `TDIV`** Division operation (a/b), two children.
- `TPOW`** Power operation (a^b; a**b), two children.
- `TNUM`** Number, no children. `Node::ival` should be specified.
- `TVAR`** `Variable`, no children. `Node::iname` should be specified.

TFOR For statement, at least 2 children. [Node::iname](#) should be specified. The first child specifies a range, the subsequent statements are sub-statements of the for loop.

TWHILE While statement, at least 2 children. The first child has to specify the condition of the loop. The subsequent statements are sub-statements of the while loop.

TASSIGN Assign statement ($a = b$), 2 children.

TRANGE Range operation ($a:b$), 2 children.

TOR OR operation ($a || b$; a OR b), 2 children.

TAND AND operation ($a \&\& b$; a AND b), 2 children.

TEQ_OP EQUAL operation ($a == b$), 2 children.

TNE_OP NOT EQUAL operation ($a \sim= b$; $a != b$), 2 children.

TGT_OP GREATER THAN operation ($a > b$), 2 children.

TLT_OP LESSER THAN operation ($a < b$), 2 children.

TGE_OP GREATER OR EQUAL operation ($a >= b$), 2 children.

TLE_OP LESSER OR EQUAL operation ($a <= b$), 2 children.

TIF If statement.

TIFELSE If-else statement.

TIFELSEIF If-elseif statement.

TELSEIF Elseif statement.

TIFELSEIFELSE If-elseif-else statement.

TARRAYINDEX Array index operation, ($k[a]$), one child. [Node::iname](#) should be specified.

TLIST List/tuple operation (a,b).

TFUNCDEC [Function](#) declaration.

TMISC Miscellaneous, internal usage.

TCOMBINE Groups various statements. That is, a child of TCOMBINE actually belongs one level higher in the tree.

TNOT NOT operation ($!a$, $\sim a$, NOT a), one child.

TNEGATIVE Negative sign operation ($-a$), one child.

5.6.2 Function Documentation

5.6.2.1 void appendChild (struct Node * parent, struct Node * child)

Append a child [Node](#) to a parent [Node](#).

Append the [Node](#) *child* to the list of children of [Node](#) *parent* and sets *parent* as the parent [Node](#) of *child*.

Parameters

<i>parent</i>	Node where the <i>child</i> Node is to be appended to.
<i>child</i>	Node to be appended to the children of <i>parent</i> .

5.6.2.2 struct Node* appendStatement (struct Node * prevnodes, struct Node * newnode)

Append a statement to a list of statements.

Appends the statement *newnode* to the list of statements *prevnodes*. *prevnodes* does not have to point at the last [Node](#) of that level. This function will find the last [Node](#) of the same level and append the *newnode* there.

Parameters

<i>prevnodes</i>	Pointer to a Node on which level the Node <i>newnode</i> should be added.
<i>newnode</i>	Node to add to the linked list.

Returns

Pointer to a [Node](#) of the level where *newnode* was added.

5.6.2.3 int compareNodes (struct Node * n1, struct Node * n2)

Compare two Nodes and their children to check whether they are equivalent.

Compare the Nodes *n1* and *n2* and their children to check whether they are equivalent. All properties of the [Node](#) struct are compared, just like the complete structure the Nodes represent.

Parameters

<i>n1</i>	First Node .
<i>n2</i>	Second Node .

Returns

1 when the first and second [Node](#) are equivalent, 0 otherwise.

5.6.2.4 struct Node* copyNode (struct Node * n)

Copies the data of a [Node](#) to a new [Node](#) instance.

Copy the contents of [Node](#) *n* to a new [Node](#) and recursively does the same to all children. The neighbours of [Node](#) *n* will not be copied.

Parameters

<i>n</i>	Node to copy.
----------	-------------------------------

Returns

Pointer to the new [Node](#) with the same data as *n*.

5.6.2.5 struct Node* createConstant (double num)

Create a [Node](#) representing a constant number.

Create a [Node](#) representing the number *num*.

Parameters

<i>num</i>	The value of the constant.
------------	----------------------------

Returns

Pointer to the created [Node](#) instance.

5.6.2.6 struct Node* createOperation (enum NodeTag op)

Create a [Node](#) of the specified operation type.

Create a [Node](#) of the specified operation, whilst setting all other properties to zero or NULL.

Parameters

<i>op</i>	Operation type of the Node to be created.
-----------	---

Returns

The created [Node](#) instance.

5.6.2.7 `struct Node* createVariable (char * varname)`

Create a [Node](#) representing a variable.

Create a [Node](#) representing a variable with the name *varname*.

Parameters

<i>varname</i>	The name of the variable.
----------------	---------------------------

Returns

Pointer to the created [Node](#) instance.

5.6.2.8 `struct Node* findVariable (struct Node * n)`

Todo Describe this function

5.6.2.9 `struct Node* last (struct Node * t)`

Find the last [Node](#) on the same level as the specified [Node](#).

Search the last [Node](#) on the same level as [Node](#) *t*.

Parameters

<i>t</i>	Node of which to find the last neighbour/peer.
----------	--

Returns

The last [Node](#) on the same level as [Node](#) *t*.

5.6.2.10 `void removeNode (struct Node * n)`

Remove an [Node](#) instance and free the memory it used.

Removes [Node](#) *n* and all its children.

Parameters

<i>n</i>	Node to remove.
----------	---------------------------------

5.6.2.11 `void setIdentifier (struct Node * node, char * identifier)`

Set the identifier of a [Node](#).

Set the identifier ([Node::iname](#)) of [Node](#) *node* to *identifier*.

Parameters

<i>node</i>	Node to set the identifier of.
<i>identifier</i>	Name of the identifier.

5.7 simplify.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "simplify.h"
#include "jacobian.h"
#include "fortran.h"
#include "tree.h"
#include "node.h"
```

Macros

- [#define _GNU_SOURCE](#)

Functions

- void [simplifyStructure](#) (struct [Node](#) *t)
- void [S_zeroAssignments](#) (struct [Node](#) *t)
- void [registerZeroVar](#) (char *s, int zero)
- int [getVariableZero](#) (char *s)
- int [S_replaceZeroAssignments](#) (struct [Node](#) *t)
- void [S_removePlusZero](#) (struct [Node](#) *t)

5.7.1 Macro Definition Documentation

5.7.1.1 [#define _GNU_SOURCE](#)

5.7.2 Function Documentation

5.7.2.1 int [getVariableZero](#) (char * s)

5.7.2.2 void [registerZeroVar](#) (char * s, int zero)

5.7.2.3 void [S_removePlusZero](#) (struct [Node](#) * t)

5.7.2.4 int [S_replaceZeroAssignments](#) (struct [Node](#) * t)

5.7.2.5 void [S_zeroAssignments](#) (struct [Node](#) * t)

5.7.2.6 void [simplifyStructure](#) (struct [Node](#) * t)

5.8 simplify.h File Reference

Enumerations

- enum [SimplifyState](#) { [ZeroAssignments](#) = 0, [ReplaceZeroAssignments](#) = 1, [RemovePlusZero](#) = 2 }

Functions

- void [simplifyStructure](#) (struct [Node](#) *t)
- void [registerZeroVar](#) (char *name, int zero)
- int [getVariableZero](#) (char *s)
- void [S_zeroAssignments](#) (struct [Node](#) *t)
- int [S_replaceZeroAssignments](#) (struct [Node](#) *t)
- void [S_removePlusZero](#) (struct [Node](#) *t)

Variables

- enum [SimplifyState](#) [simplifyStateSize](#)

5.8.1 Enumeration Type Documentation

5.8.1.1 enum SimplifyState

Enumerator

ZeroAssignments

ReplaceZeroAssignments

RemovePlusZero

5.8.2 Function Documentation

5.8.2.1 int getVariableZero (char * s)

5.8.2.2 void registerZeroVar (char * name, int zero)

5.8.2.3 void S_removePlusZero (struct Node * t)

5.8.2.4 int S_replaceZeroAssignments (struct Node * t)

5.8.2.5 void S_zeroAssignments (struct Node * t)

5.8.2.6 void simplifyStructure (struct Node * t)

5.8.3 Variable Documentation

5.8.3.1 enum SimplifyState simplifyStateSize

5.9 tree.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "tree.h"
#include "fortran.h"
#include "jacobian.h"
#include "node.h"
```

Macros

- `#define _GNU_SOURCE`

Functions

- void * `emalloc` (size_t nbytes)
Nicer memory allocation function.
- void `fatalError` (char *message)
Fatal error function.
- void `depth` (int d)
- void `print_tree` (int d, struct `Node` *t)
Print a tree view of a `Node`.
- struct `Variable` * `registerVariable` (char *s, enum `VariableType` tp)
- char * `processIdentifier` (char *nm, enum `VariableType` tp)
- void `processFunctionHeader` (struct `Node` *f)
- void `processDependentVectorIdentifier` (char *s, struct `Node` *rel)
- struct `Node` * `getRelativeToY` (char *s)

5.9.1 Macro Definition Documentation

5.9.1.1 #define _GNU_SOURCE

5.9.2 Function Documentation

5.9.2.1 void depth (int d)

5.9.2.2 void* emalloc (size_t nbytes)

Nicer memory allocation function.

5.9.2.3 void fatalError (char * message)

Fatal error function.

Displays error *message* and then exits the application.

Parameters

<i>message</i>	Error message to display.
----------------	---------------------------

5.9.2.4 struct Node* getRelativeToY (char * s)

5.9.2.5 void print_tree (int d, struct Node * t)

Print a tree view of a `Node`.

5.9.2.6 void processDependentVectorIdentifier (char * s, struct Node * rel)

5.9.2.7 void processFunctionHeader (struct Node * f)

5.9.2.8 char* processIdentifier (char * nm, enum VariableType tp)

5.9.2.9 struct Variable* registerVariable (char * s, enum VariableType tp)

5.10 tree.h File Reference

```
#include "stdio.h"
```

Data Structures

- struct [Variable](#)
Datatype of a variable.
- struct [Function](#)
Datatype of a function definition.

Macros

- #define [MIN](#)(x, y) (((x) < (y)) ? (x) : (y))

Enumerations

- enum [VariableType](#) { [TDOUBLE](#), [TINT](#), [TDOUBLEARRAY](#) }
Variable types.

Functions

- void * [emalloc](#) (size_t nbytes)
Nicer memory allocation function.
- void [fatalError](#) (char *message)
Fatal error function.
- void [print_tree](#) (int d, struct [Node](#) *t)
Print a tree view of a [Node](#).
- struct [Variable](#) * [registerVariable](#) (char *s, enum [VariableType](#) tp)
- char * [processIdentifier](#) (char *nm, enum [VariableType](#) tp)
- void [processDependentVectorIdentifier](#) (char *s, struct [Node](#) *rel)
- void [processFunctionHeader](#) (struct [Node](#) *f)
- struct [Node](#) * [getRelativeToY](#) (char *s)

Variables

- FILE * [warn](#)
- FILE * [out](#)
- int [labelcount](#)
Counter for the label to use in the Fortran code.
- struct [Function](#) * [func](#)

Instance of the function definition.

- struct [Variable](#) * [vars](#)

Variables used in the ODE function.

5.10.1 Macro Definition Documentation

5.10.1.1 `#define MIN(x, y) (((x) < (y)) ? (x) : (y))`

5.10.2 Enumeration Type Documentation

5.10.2.1 `enum VariableType`

[Variable](#) types.

Enumerator

TDOUBLE Double precision floating point variable.

TINT Integer variable.

TDOUBLEARRAY Array of double precision variables.

5.10.3 Function Documentation

5.10.3.1 `void* emalloc (size_t nbytes)`

Nicer memory allocation function.

5.10.3.2 `void fatalError (char * message)`

Fatal error function.

Displays error *message* and then exits the application.

Parameters

<i>message</i>	Error message to display.
----------------	---------------------------

5.10.3.3 `struct Node* getRelativeToY (char * s)`

5.10.3.4 `void print_tree (int d, struct Node * t)`

Print a tree view of a [Node](#).

5.10.3.5 `void processDependentVectorIdentifier (char * s, struct Node * rel)`

5.10.3.6 `void processFunctionHeader (struct Node * f)`

5.10.3.7 `char* processIdentifier (char * nm, enum VariableType tp)`

5.10.3.8 `struct Variable* registerVariable (char * s, enum VariableType tp)`

5.10.4 Variable Documentation

5.10.4.1 struct Function* func

Instance of the function definition.

5.10.4.2 int labelcount

Counter for the label to use in the Fortran code.

5.10.4.3 FILE* out**5.10.4.4 struct Variable* vars**

Variables used in the ODE function.

5.10.4.5 FILE* warn

Index

`_GNU_SOURCE`
 `fortran.c`, 14
 `jacobian.c`, 18
 `simplify.c`, 27
 `tree.c`, 29

`appendChild`
 `node.c`, 20
 `node.h`, 24
`appendStatement`
 `node.c`, 20
 `node.h`, 24

`children`
 `Node`, 9
`compareNodes`
 `node.c`, 20
 `node.h`, 25
`copyNode`
 `node.c`, 21
 `node.h`, 25
`createConstant`
 `node.c`, 21
 `node.h`, 25
`createOperation`
 `node.c`, 21
 `node.h`, 25
`createVariable`
 `node.c`, 21
 `node.h`, 26

`D`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_arrayindex`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_assign`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_div`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_for`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_if`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_ifelse`

`jacobian.c`, 18
 `jacobian.h`, 19
`D_minus`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_mul`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_negative`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_plus`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_pow`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_var`
 `jacobian.c`, 18
 `jacobian.h`, 19
`D_zeros`
 `jacobian.c`, 18
 `jacobian.h`, 19
`depth`
 `tree.c`, 29
`derivative`
 `jacobian.c`, 18
 `jacobian.h`, 19
`dx`
 `Function`, 7

`emalloc`
 `tree.c`, 29
 `tree.h`, 31

`F_and`
 `fortran.c`, 14
 `fortran.h`, 16
`F_arrayindex`
 `fortran.c`, 14
 `fortran.h`, 16
`F_assign`
 `fortran.c`, 14
 `fortran.h`, 16
`F_combine`
 `fortran.c`, 14
 `fortran.h`, 16
`F_div`
 `fortran.c`, 14
 `fortran.h`, 16

- F_elseif
 - fortran.c, 14
 - fortran.h, 16
- F_eq_op
 - fortran.c, 14
 - fortran.h, 16
- F_for
 - fortran.c, 14
 - fortran.h, 16
- F_ge_op
 - fortran.c, 14
 - fortran.h, 16
- F_gt_op
 - fortran.c, 14
 - fortran.h, 16
- F_if
 - fortran.c, 14
 - fortran.h, 16
- F_ifelse
 - fortran.c, 14
 - fortran.h, 16
- F_ifelseif
 - fortran.c, 14
 - fortran.h, 16
- F_ifelseifelse
 - fortran.c, 14
 - fortran.h, 16
- F_indexrange
 - fortran.c, 14
 - fortran.h, 16
- F_le_op
 - fortran.c, 14
 - fortran.h, 16
- F_lt_op
 - fortran.c, 14
 - fortran.h, 16
- F_minus
 - fortran.c, 14
 - fortran.h, 16
- F_mul
 - fortran.c, 14
 - fortran.h, 16
- F_ne_op
 - fortran.c, 14
 - fortran.h, 16
- F_negative
 - fortran.c, 15
 - fortran.h, 16
- F_not
 - fortran.c, 15
 - fortran.h, 16
- F_or
 - fortran.c, 15
 - fortran.h, 16
- F_plus
 - fortran.c, 15
 - fortran.h, 16
- F_pow
 - fortran.c, 15
 - fortran.h, 17
- F_range
 - fortran.c, 15
 - fortran.h, 17
- F_while
 - fortran.c, 15
 - fortran.h, 17
- F_zeros
 - fortran.c, 15
 - fortran.h, 17
- fatalError
 - tree.c, 29
 - tree.h, 31
- findVariable
 - node.c, 22
 - node.h, 26
- fortran.c, 13
 - _GNU_SOURCE, 14
 - F_and, 14
 - F_arrayindex, 14
 - F_assign, 14
 - F_combine, 14
 - F_div, 14
 - F_elseif, 14
 - F_eq_op, 14
 - F_for, 14
 - F_ge_op, 14
 - F_gt_op, 14
 - F_if, 14
 - F_ifelse, 14
 - F_ifelseif, 14
 - F_ifelseifelse, 14
 - F_indexrange, 14
 - F_le_op, 14
 - F_lt_op, 14
 - F_minus, 14
 - F_mul, 14
 - F_ne_op, 14
 - F_negative, 15
 - F_not, 15
 - F_or, 15
 - F_plus, 15
 - F_pow, 15
 - F_range, 15
 - F_while, 15
 - F_zeros, 15
 - functionToFortran, 15
 - line, 15
 - printFortranFunction, 15
 - toFortran, 15
- fortran.h, 15
 - F_and, 16
 - F_arrayindex, 16
 - F_assign, 16
 - F_combine, 16
 - F_div, 16
 - F_elseif, 16

- F_eq_op, 16
- F_for, 16
- F_ge_op, 16
- F_gt_op, 16
- F_if, 16
- F_ifelse, 16
- F_ifelseif, 16
- F_ifelseifelse, 16
- F_indeXrange, 16
- F_le_op, 16
- F_lt_op, 16
- F_minus, 16
- F_mul, 16
- F_ne_op, 16
- F_negative, 16
- F_not, 16
- F_or, 16
- F_plus, 16
- F_pow, 17
- F_range, 17
- F_while, 17
- F_zeros, 17
- functionToFortran, 17
- line, 17
- printFortranFunction, 17
- toFortran, 17
- func
 - tree.h, 31
- Function, 7
 - dx, 7
 - j, 7
 - neq, 8
 - np, 8
 - p, 8
 - t, 8
 - x, 8
- functionToFortran
 - fortran.c, 15
 - fortran.h, 17
- functionToJacobian
 - jacobian.c, 18
 - jacobian.h, 19
- getRelativeToY
 - tree.c, 29
 - tree.h, 31
- getVariableZero
 - simplify.c, 27
 - simplify.h, 28
- ignore
 - Node, 9
- iname
 - Node, 9
 - Variable, 11
- ival
 - Node, 9
- j
 - Function, 7
 - jacobian.c, 17
 - _GNU_SOURCE, 18
 - D, 18
 - D_arrayindex, 18
 - D_assign, 18
 - D_div, 18
 - D_for, 18
 - D_if, 18
 - D_ifelse, 18
 - D_minus, 18
 - D_mul, 18
 - D_negative, 18
 - D_plus, 18
 - D_pow, 18
 - D_var, 18
 - D_zeros, 18
 - derivative, 18
 - functionToJacobian, 18
 - jacobian.h, 18
 - D, 19
 - D_arrayindex, 19
 - D_assign, 19
 - D_div, 19
 - D_for, 19
 - D_if, 19
 - D_ifelse, 19
 - D_minus, 19
 - D_mul, 19
 - D_negative, 19
 - D_plus, 19
 - D_pow, 19
 - D_var, 19
 - D_zeros, 19
 - derivative, 19
 - functionToJacobian, 19
 - main, 19
- labelcount
 - tree.h, 32
- last
 - node.c, 22
 - node.h, 26
- line
 - fortran.c, 15
 - fortran.h, 17
- MIN
 - tree.h, 31
- main
 - jacobian.h, 19
- neq
 - Function, 8
- next
 - Node, 10
 - Variable, 11
- Node, 8
 - children, 9

- ignore, 9
- iname, 9
- ival, 9
- next, 10
- parent, 10
- previous, 10
- tag, 10
- node.h
 - TAND, 24
 - TARRAYINDEX, 24
 - TASSIGN, 24
 - TCOMBINE, 24
 - TDIV, 23
 - TELSEIF, 24
 - TEQ_OP, 24
 - TFOR, 23
 - TFUNCDEC, 24
 - TGE_OP, 24
 - TGT_OP, 24
 - TIF, 24
 - TIFELSE, 24
 - TIFELSEIF, 24
 - TIFELSEIFELSE, 24
 - TLE_OP, 24
 - TLIST, 24
 - TLT_OP, 24
 - TMINUS, 23
 - TMISC, 24
 - TMUL, 23
 - TNE_OP, 24
 - TNEGATIVE, 24
 - TNOT, 24
 - TNUM, 23
 - TOR, 24
 - TPLUS, 23
 - TPOW, 23
 - TRANGE, 24
 - TVAR, 23
 - TWHILE, 24
- node.c, 19
 - appendChild, 20
 - appendStatement, 20
 - compareNodes, 20
 - copyNode, 21
 - createConstant, 21
 - createOperation, 21
 - createVariable, 21
 - findVariable, 22
 - last, 22
 - removeNode, 22
 - setIdentifier, 22
- node.h, 22
 - appendChild, 24
 - appendStatement, 24
 - compareNodes, 25
 - copyNode, 25
 - createConstant, 25
 - createOperation, 25
 - createVariable, 26
 - findVariable, 26
 - last, 26
 - NodeTag, 23
 - removeNode, 26
 - setIdentifier, 26
- NodeTag
 - node.h, 23
- np
 - Function, 8
- out
 - tree.h, 32
- p
 - Function, 8
- parent
 - Node, 10
- previous
 - Node, 10
 - Variable, 11
- print_tree
 - tree.c, 29
 - tree.h, 31
- printFortranFunction
 - fortran.c, 15
 - fortran.h, 17
- processDependentVectorIdentifier
 - tree.c, 29
 - tree.h, 31
- processFunctionHeader
 - tree.c, 30
 - tree.h, 31
- processIdentifier
 - tree.c, 30
 - tree.h, 31
- registerVariable
 - tree.c, 30
 - tree.h, 31
- registerZeroVar
 - simplify.c, 27
 - simplify.h, 28
- rel
 - Variable, 11
- RemovePlusZero
 - simplify.h, 28
- removeNode
 - node.c, 22
 - node.h, 26
- ReplaceZeroAssignments
 - simplify.h, 28
- S_removePlusZero
 - simplify.c, 27
 - simplify.h, 28
- S_replaceZeroAssignments
 - simplify.c, 27
 - simplify.h, 28

- S_zeroAssignments
 - simplify.c, [27](#)
 - simplify.h, [28](#)
- setIdentifier
 - node.c, [22](#)
 - node.h, [26](#)
- simplify.h
 - RemovePlusZero, [28](#)
 - ReplaceZeroAssignments, [28](#)
 - ZeroAssignments, [28](#)
- simplify.c, [27](#)
 - _GNU_SOURCE, [27](#)
 - getVariableZero, [27](#)
 - registerZeroVar, [27](#)
 - S_removePlusZero, [27](#)
 - S_replaceZeroAssignments, [27](#)
 - S_zeroAssignments, [27](#)
 - simplifyStructure, [27](#)
- simplify.h, [27](#)
 - getVariableZero, [28](#)
 - registerZeroVar, [28](#)
 - S_removePlusZero, [28](#)
 - S_replaceZeroAssignments, [28](#)
 - S_zeroAssignments, [28](#)
 - SimplifyState, [28](#)
 - simplifyStateSize, [28](#)
 - simplifyStructure, [28](#)
- SimplifyState
 - simplify.h, [28](#)
- simplifyStateSize
 - simplify.h, [28](#)
- simplifyStructure
 - simplify.c, [27](#)
 - simplify.h, [28](#)
- t
 - Function, [8](#)
- TAND
 - node.h, [24](#)
- TARRAYINDEX
 - node.h, [24](#)
- TASSIGN
 - node.h, [24](#)
- TCOMBINE
 - node.h, [24](#)
- TDIV
 - node.h, [23](#)
- TDOUBLE
 - tree.h, [31](#)
- TDOUBLEARRAY
 - tree.h, [31](#)
- TELSEIF
 - node.h, [24](#)
- TEQ_OP
 - node.h, [24](#)
- TFOR
 - node.h, [23](#)
- TFUNCDEC
 - node.h, [24](#)
- TGE_OP
 - node.h, [24](#)
- TGT_OP
 - node.h, [24](#)
- TIF
 - node.h, [24](#)
- TIFELSE
 - node.h, [24](#)
- TIFELSEIF
 - node.h, [24](#)
- TIFELSEIFELSE
 - node.h, [24](#)
- TINT
 - tree.h, [31](#)
- TLE_OP
 - node.h, [24](#)
- TLIST
 - node.h, [24](#)
- TLT_OP
 - node.h, [24](#)
- TMINUS
 - node.h, [23](#)
- TMISC
 - node.h, [24](#)
- TMUL
 - node.h, [23](#)
- TNE_OP
 - node.h, [24](#)
- TNEGATIVE
 - node.h, [24](#)
- TNOT
 - node.h, [24](#)
- TNUM
 - node.h, [23](#)
- TOR
 - node.h, [24](#)
- TPLUS
 - node.h, [23](#)
- TPOW
 - node.h, [23](#)
- TRANGE
 - node.h, [24](#)
- TVAR
 - node.h, [23](#)
- TWHILE
 - node.h, [24](#)
- tag
 - Node, [10](#)
- toFortran
 - fortran.c, [15](#)
 - fortran.h, [17](#)
- tree.h
 - TDOUBLE, [31](#)
 - TDOUBLEARRAY, [31](#)
 - TINT, [31](#)
- tree.c, [28](#)
 - _GNU_SOURCE, [29](#)
 - depth, [29](#)

- emalloc, [29](#)
- fatalError, [29](#)
- getRelativeToY, [29](#)
- print_tree, [29](#)
- processDependentVectorIdentifier, [29](#)
- processFunctionHeader, [30](#)
- processIdentifier, [30](#)
- registerVariable, [30](#)
- tree.h, [30](#)
 - emalloc, [31](#)
 - fatalError, [31](#)
 - func, [31](#)
 - getRelativeToY, [31](#)
 - labelcount, [32](#)
 - MIN, [31](#)
 - out, [32](#)
 - print_tree, [31](#)
 - processDependentVectorIdentifier, [31](#)
 - processFunctionHeader, [31](#)
 - processIdentifier, [31](#)
 - registerVariable, [31](#)
 - VariableType, [31](#)
 - vars, [32](#)
 - warn, [32](#)
- type
 - Variable, [11](#)
- Variable, [10](#)
 - iname, [11](#)
 - next, [11](#)
 - previous, [11](#)
 - rel, [11](#)
 - type, [11](#)
 - zero, [11](#)
- VariableType
 - tree.h, [31](#)
- vars
 - tree.h, [32](#)
- warn
 - tree.h, [32](#)
- x
 - Function, [8](#)
- zero
 - Variable, [11](#)
- ZeroAssignments
 - simplify.h, [28](#)