PPODESUITE Parser

0.1beta

Generated by Doxygen 1.8.4

Mon Mar 24 2014 09:39:39

Contents

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:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 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

parent	Node where the *child* Node is to be appended to.
child	Node to be appended to the children of *parent*.

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

prevnodes	Pointer to a Node on which level the Node *newnode* should be added.
newnode	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

n1	First Node.
n2	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

n	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

num	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

op	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

varname	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

t	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

n	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

node	Node to set the identifier of.
identifier	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 ~= 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 Miscelaneous, internal usage.

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

TNOT NOT operation (!a, ~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

parent	Node where the *child* Node is to be appended to.
child	Node to be appended to the children of *parent*.

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

prevnodes	Pointer to a Node on which level the Node *newnode* should be added.
newnode	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

n1	First Node.
n2	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

n	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

num	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

op	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

varname	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

t	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

n	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

node	Node to set the identifier of.
identifier	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

message	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

message	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