RelMDD

1.0

Generated by Doxygen 1.8.1

Tue Aug 7 2012 23:41:35

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

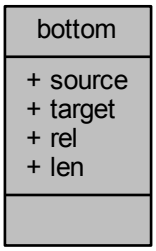
Chapter 3

Data Structure Documentation

3.1 bottom Struct Reference

`#include <structures.h>`

Collaboration diagram for bottom:



Data Fields

- char ** **source**
- char ** **target**
- double * **rel**
- int **len**

3.1.1 Detailed Description

Definition at line 18 of file structures.h.

3.1.2 Field Documentation

3.1.2.1 int len

Definition at line 23 of file structures.h.

Referenced by BottomRelation(), identityRelation(), and relxmlReadBottom().

3.1.2.2 double* rel

Definition at line 22 of file structures.h.

Referenced by BottomRelation(), identityRelation(), and relxmlReadBottom().

3.1.2.3 char** source

Definition at line 20 of file structures.h.

Referenced by relxmlReadBottom().

3.1.2.4 char** target

Definition at line 21 of file structures.h.

Referenced by relxmlReadBottom().

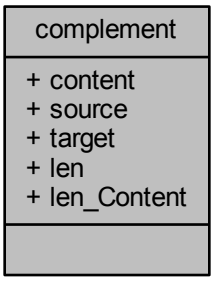
The documentation for this struct was generated from the following file:

- **structures.h**

3.2 complement Struct Reference

`#include <structures.h>`

Collaboration diagram for complement:



Data Fields

- double * **content**
- char ** **source**
- char ** **target**
- int **len**
- int * **len_Content**

3.2.1 Detailed Description

Definition at line 76 of file structures.h.

3.2.2 Field Documentation

3.2.2.1 double* content

Definition at line 77 of file structures.h.

Referenced by complementRelation(), and relxmlReadComplement().

3.2.2.2 int len

Definition at line 80 of file structures.h.

Referenced by complementRelation(), and relxmlReadComplement().

3.2.2.3 int* len_Content

Definition at line 81 of file structures.h.

Referenced by complementRelation(), and relxmlReadComplement().

3.2.2.4 char** source

Definition at line 78 of file structures.h.

Referenced by relxmlReadComplement().

3.2.2.5 char** target

Definition at line 79 of file structures.h.

Referenced by relxmlReadComplement().

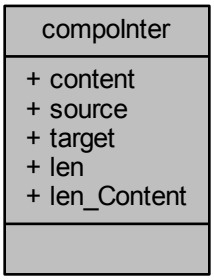
The documentation for this struct was generated from the following file:

- **structures.h**

3.3 compoInter Struct Reference

`#include <structures.h>`

Collaboration diagram for compoInter:



Data Fields

- double ∗ **content**
- char ∗∗ **source**
- char ∗∗ **target**
- int **len**
- int ∗ **len_Content**

3.3.1 Detailed Description

Definition at line 58 of file structures.h.

3.3.2 Field Documentation

3.3.2.1 double∗ content

Definition at line 59 of file structures.h.

Referenced by interserction_OperationComposition(), and relxmlReadCompostion_Inter().

3.3.2.2 int len

Definition at line 62 of file structures.h.

Referenced by interserction_OperationComposition(), and relxmlReadCompostion_Inter().

3.3.2.3 int∗ len_Content

Definition at line 63 of file structures.h.

Referenced by interserction_OperationComposition(), and relxmlReadCompostion_Inter().

3.3.2.4 char∗∗ source

Definition at line 60 of file structures.h.

Referenced by relxmlReadCompostion_Inter().

3.3.2.5 char** target

Definition at line 61 of file structures.h.

Referenced by relxmlReadCompostion_Inter().

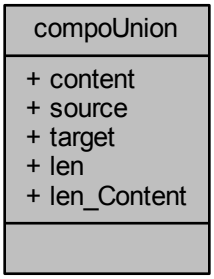
The documentation for this struct was generated from the following file:

- **structures.h**

3.4 compoUnion Struct Reference

`#include <structures.h>`

Collaboration diagram for compoUnion:



Data Fields

- double * **content**
- char ** **source**
- char ** **target**
- int **len**
- int * **len_Content**

3.4.1 Detailed Description

Definition at line 50 of file structures.h.

3.4.2 Field Documentation

3.4.2.1 double* content

Definition at line 51 of file structures.h.

Referenced by relxmlReadCompostion_Union(), and union_OperationComposition().

3.4.2.2 int len

Definition at line 54 of file structures.h.

Referenced by relxmlReadCompostion_Union(), and union_OperationComposition().

3.4.2.3 int* len_Content

Definition at line 55 of file structures.h.

Referenced by relxmlReadCompostion_Union(), and union_OperationComposition().

3.4.2.4 char** source

Definition at line 52 of file structures.h.

Referenced by relxmlReadCompostion_Union().

3.4.2.5 char** target

Definition at line 53 of file structures.h.

Referenced by relxmlReadCompostion_Union().

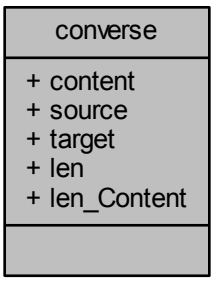
The documentation for this struct was generated from the following file:

- **structures.h**

3.5 converse Struct Reference

`#include <structures.h>`

Collaboration diagram for converse:



Data Fields

- double * **content**
- char ** **source**
- char ** **target**

- int **len**
- int * **len_Content**

3.5.1 Detailed Description

Definition at line 68 of file structures.h.

3.5.2 Field Documentation

3.5.2.1 double* content

Definition at line 69 of file structures.h.

Referenced by relxmlReadTransposition(), and transposeRelation().

3.5.2.2 int len

Definition at line 72 of file structures.h.

Referenced by relxmlReadTransposition(), and transposeRelation().

3.5.2.3 int* len_Content

Definition at line 73 of file structures.h.

Referenced by relxmlReadTransposition(), and transposeRelation().

3.5.2.4 char** source

Definition at line 70 of file structures.h.

Referenced by relxmlReadTransposition().

3.5.2.5 char** target

Definition at line 71 of file structures.h.

Referenced by relxmlReadTransposition().

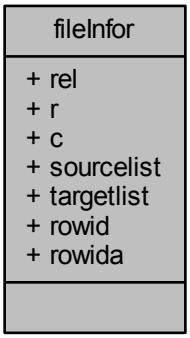
The documentation for this struct was generated from the following file:

- **structures.h**

3.6 fileInfor Struct Reference

`#include <RelMDD.h>`

Collaboration diagram for fileInfor:



Data Fields

- double ∗ **rel**
- int **r**
- int **c**
- char ∗ **sourcelist**
- char ∗ **targetlist**
- int ∗ **rowid**
- int ∗ **rowida**

3.6.1 Detailed Description

Definition at line 80 of file RelMDD.h.

3.6.2 Field Documentation

3.6.2.1 int c

Definition at line 83 of file RelMDD.h.

Referenced by main(), and RelMdd_ReadFile().

3.6.2.2 int r

Definition at line 82 of file RelMDD.h.

Referenced by main(), and RelMdd_ReadFile().

3.6.2.3 double∗ rel

Definition at line 81 of file RelMDD.h.

Referenced by main(), and RelMdd_ReadFile().

3.6.2.4 int* rowid

Definition at line 86 of file RelMDD.h.

Referenced by RelMdd_ReadFile().

3.6.2.5 int* rowida

Definition at line 87 of file RelMDD.h.

Referenced by RelMdd_ReadFile().

3.6.2.6 char* sourcelist

Definition at line 84 of file RelMDD.h.

Referenced by main(), and RelMdd_ReadFile().

3.6.2.7 char* targetlist

Definition at line 85 of file RelMDD.h.

Referenced by main(), and RelMdd_ReadFile().

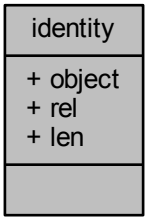
The documentation for this struct was generated from the following file:

- **RelMDD.h**

3.7 identity Struct Reference

`#include <structures.h>`

Collaboration diagram for identity:



Data Fields

- char ** **object**
- double * **rel**
- int **len**

3.7.1 Detailed Description

Definition at line 12 of file structures.h.

3.7.2 Field Documentation

3.7.2.1 int len

Definition at line 15 of file structures.h.

Referenced by identityRelation(), and relxmlReadIdentity().

3.7.2.2 char** object

Definition at line 13 of file structures.h.

Referenced by relxmlReadIdentity().

3.7.2.3 double* rel

Definition at line 14 of file structures.h.

Referenced by identityRelation(), and relxmlReadIdentity().

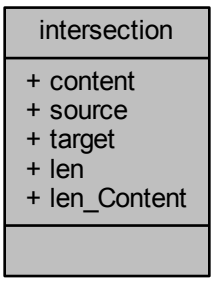
The documentation for this struct was generated from the following file:

- **structures.h**

3.8 intersection Struct Reference

`#include <structures.h>`

Collaboration diagram for intersection:



Data Fields

- double * **content**
- char ** **source**

- char ** **target**
- int **len**
- int * **len_Content**

3.8.1 Detailed Description

Definition at line 42 of file structures.h.

3.8.2 Field Documentation

3.8.2.1 double* content

Definition at line 43 of file structures.h.

Referenced by interrelation(), and relxmlReadIntersection().

3.8.2.2 int len

Definition at line 46 of file structures.h.

Referenced by interrelation(), and relxmlReadIntersection().

3.8.2.3 int* len_Content

Definition at line 47 of file structures.h.

Referenced by interrelation(), and relxmlReadIntersection().

3.8.2.4 char** source

Definition at line 44 of file structures.h.

Referenced by relxmlReadIntersection().

3.8.2.5 char** target

Definition at line 45 of file structures.h.

Referenced by relxmlReadIntersection().

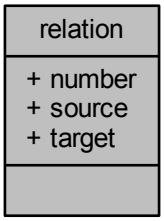
The documentation for this struct was generated from the following file:

- **structures.h**

3.9 relation Struct Reference

```
#include <structures.h>
```

Collaboration diagram for relation:



Data Fields

- int * **number**
- char ** **source**
- char ** **target**

3.9.1 Detailed Description

Definition at line 5 of file structures.h.

3.9.2 Field Documentation

3.9.2.1 int* number

Definition at line 6 of file structures.h.

Referenced by relxmlReadrelations().

3.9.2.2 char** source

Definition at line 7 of file structures.h.

Referenced by relxmlReadrelations().

3.9.2.3 char** target

Definition at line 8 of file structures.h.

Referenced by relxmlReadrelations().

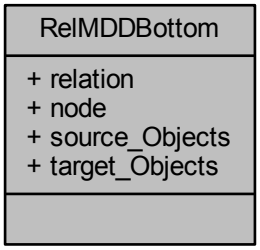
The documentation for this struct was generated from the following file:

- **structures.h**

3.10 RelMDDBottom Struct Reference

`#include <RelMDD.h>`

Collaboration diagram for RelMDDBottom:



Data Fields

- double * **relation**
- DdNode * **node**
- char * **source_Objects**
- char * **target_Objects**

3.10.1 Detailed Description

Definition at line 54 of file RelMDD.h.

3.10.2 Field Documentation

3.10.2.1 DdNode* node

Definition at line 56 of file RelMDD.h.

Referenced by RelMDDBottomRelation().

3.10.2.2 double* relation

Definition at line 55 of file RelMDD.h.

Referenced by RelMDDBottomRelation().

3.10.2.3 char* source_Objects

Definition at line 57 of file RelMDD.h.

Referenced by RelMDDBottomRelation().

3.10.2.4 char* target_Objects

Definition at line 58 of file RelMDD.h.

Referenced by RelMDDBottomRelation().

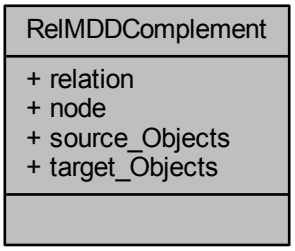
The documentation for this struct was generated from the following file:

- **RelMDD.h**

3.11 RelMDDComplement Struct Reference

`#include <RelMDD.h>`

Collaboration diagram for RelMDDComplement:



Data Fields

- double * **relation**
- DdNode * **node**
- char * **source_Objects**
- char * **target_Objects**

3.11.1 Detailed Description

Definition at line 30 of file RelMDD.h.

3.11.2 Field Documentation

3.11.2.1 DdNode* node

Definition at line 32 of file RelMDD.h.

Referenced by RelMDDComplement_Operation().

3.11.2.2 double* relation

Definition at line 31 of file RelMDD.h.

Referenced by RelMDDComplement_Operation().

3.11.2.3 char* source_Objects

Definition at line 33 of file RelMDD.h.

Referenced by RelMDDComplement_Operation().

3.11.2.4 char* target_Objects

Definition at line 34 of file RelMDD.h.

Referenced by RelMDDComplement_Operation().

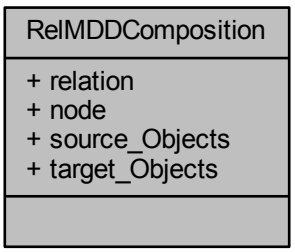
The documentation for this struct was generated from the following file:

- **RelMDD.h**

3.12 RelMDDComposition Struct Reference

`#include <RelMDD.h>`

Collaboration diagram for RelMDDComposition:



Data Fields

- double * **relation**
- DdNode * **node**
- char * **source_Objects**
- char * **target_Objects**

3.12.1 Detailed Description

Definition at line 71 of file RelMDD.h.

3.12.2 Field Documentation

3.12.2.1 DdNode* node

Definition at line 73 of file RelMDD.h.

Referenced by RelMDDCompositionOperation().

3.12.2.2 double* relation

Definition at line 72 of file RelMDD.h.

Referenced by RelMDDCompositionOperation().

3.12.2.3 char* source_Objects

Definition at line 74 of file RelMDD.h.

Referenced by RelMDDCompositionOperation().

3.12.2.4 char* target_Objects

Definition at line 75 of file RelMDD.h.

Referenced by RelMDDCompositionOperation().

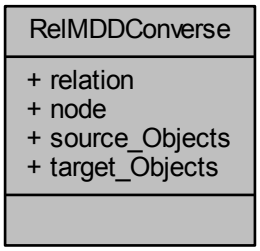
The documentation for this struct was generated from the following file:

- **RelMDD.h**

3.13 RelMDDConverse Struct Reference

`#include <RelMDD.h>`

Collaboration diagram for RelMDDConverse:



Data Fields

- double * **relation**
- DdNode * **node**
- char * **source_Objects**
- char * **target_Objects**

3.13.1 Detailed Description

Definition at line 39 of file RelMDD.h.

3.13.2 Field Documentation

3.13.2.1 DdNode* node

Definition at line 41 of file RelMDD.h.

Referenced by RelMDDConverse_Operation().

3.13.2.2 double* relation

Definition at line 40 of file RelMDD.h.

Referenced by RelMDDConverse_Operation().

3.13.2.3 char* source_Objects

Definition at line 42 of file RelMDD.h.

Referenced by RelMDDConverse_Operation().

3.13.2.4 char* target_Objects

Definition at line 43 of file RelMDD.h.

Referenced by RelMDDConverse_Operation().

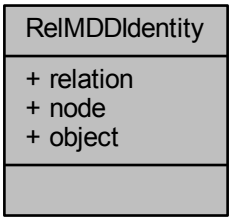
The documentation for this struct was generated from the following file:

- **RelMDD.h**

3.14 RelMDDIdentity Struct Reference

`#include <RelMDD.h>`

Collaboration diagram for RelMDDIdentity:



Data Fields

- double * **relation**
- DdNode * **node**
- char * **object**

3.14.1 Detailed Description

Definition at line 47 of file RelMDD.h.

3.14.2 Field Documentation

3.14.2.1 DdNode* node

Definition at line 49 of file RelMDD.h.

Referenced by RelMDDIdentityRelation().

3.14.2.2 char* object

Definition at line 50 of file RelMDD.h.

Referenced by RelMDDIdentityRelation().

3.14.2.3 double* relation

Definition at line 48 of file RelMDD.h.

Referenced by RelMDDIdentityRelation().

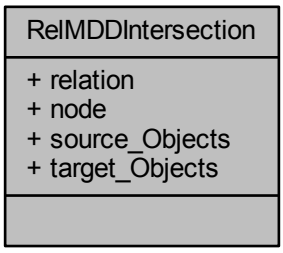
The documentation for this struct was generated from the following file:

- **RelMDD.h**

3.15 RelMDDIntersection Struct Reference

`#include <RelMDD.h>`

Collaboration diagram for RelMDDIntersection:



Data Fields

- double * **relation**
- DdNode * **node**
- char * **source_Objects**
- char * **target_Objects**

3.15.1 Detailed Description

Definition at line 21 of file RelMDD.h.

3.15.2 Field Documentation

3.15.2.1 DdNode* node

Definition at line 23 of file RelMDD.h.

Referenced by RelMDDintersectOperation().

3.15.2.2 double* relation

Definition at line 22 of file RelMDD.h.

Referenced by main(), and RelMDDintersectOperation().

3.15.2.3 char* source_Objects

Definition at line 24 of file RelMDD.h.

Referenced by RelMDDintersectOperation().

3.15.2.4 char* target_Objects

Definition at line 25 of file RelMDD.h.

Referenced by RelMDDintersectOperation().

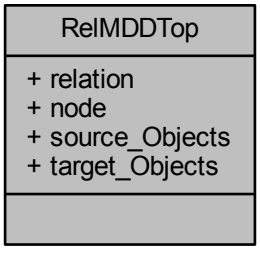
The documentation for this struct was generated from the following file:

- **RelMDD.h**

3.16 RelMDDTop Struct Reference

`#include <RelMDD.h>`

Collaboration diagram for RelMDDTop:



Data Fields

- double * **relation**
- DdNode * **node**

- char * **source_Objects**
- char * **target_Objects**

3.16.1 Detailed Description

Definition at line 62 of file RelMDD.h.

3.16.2 Field Documentation

3.16.2.1 DdNode* node

Definition at line 64 of file RelMDD.h.

Referenced by RelMDDTopRelation().

3.16.2.2 double* relation

Definition at line 63 of file RelMDD.h.

Referenced by RelMDDTopRelation().

3.16.2.3 char* source_Objects

Definition at line 65 of file RelMDD.h.

Referenced by RelMDDTopRelation().

3.16.2.4 char* target_Objects

Definition at line 66 of file RelMDD.h.

Referenced by RelMDDTopRelation().

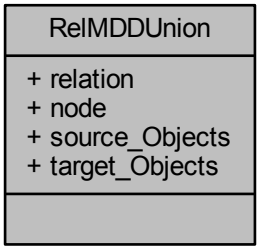
The documentation for this struct was generated from the following file:

- **RelMDD.h**

3.17 RelMDDUnion Struct Reference

`#include <RelMDD.h>`

Collaboration diagram for RelMDDUnion:



Data Fields

- double * **relation**
- DdNode * **node**
- char * **source_Objects**
- char * **target_Objects**

3.17.1 Detailed Description

Definition at line 13 of file RelMDD.h.

3.17.2 Field Documentation

3.17.2.1 DdNode* node

Definition at line 15 of file RelMDD.h.

Referenced by RelMDDUnionOperation().

3.17.2.2 double* relation

Definition at line 14 of file RelMDD.h.

Referenced by main(), and RelMDDUnionOperation().

3.17.2.3 char* source_Objects

Definition at line 16 of file RelMDD.h.

Referenced by RelMDDUnionOperation().

3.17.2.4 char* target_Objects

Definition at line 17 of file RelMDD.h.

Referenced by RelMDDUnionOperation().

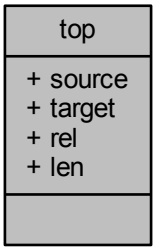
The documentation for this struct was generated from the following file:

- **RelMDD.h**

3.18 top Struct Reference

`#include <structures.h>`

Collaboration diagram for top:



Data Fields

- char ∗∗ **source**
- char ∗∗ **target**
- double ∗ **rel**
- int **len**

3.18.1 Detailed Description

Definition at line 26 of file structures.h.

3.18.2 Field Documentation

3.18.2.1 int len

Definition at line 31 of file structures.h.

Referenced by relxmlReadTop(), and topRelation().

3.18.2.2 double∗ rel

Definition at line 30 of file structures.h.

Referenced by relxmlReadTop(), and topRelation().

3.18.2.3 char∗∗ source

Definition at line 28 of file structures.h.

Referenced by relxmlReadTop().

3.18.2.4 char** target

Definition at line 29 of file structures.h.

Referenced by relxmlReadTop().

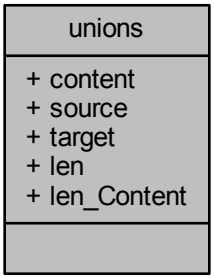
The documentation for this struct was generated from the following file:

- **structures.h**

3.19 unions Struct Reference

`#include <structures.h>`

Collaboration diagram for unions:



Data Fields

- double * **content**
- char ** **source**
- char ** **target**
- int **len**
- int * **len_Content**

3.19.1 Detailed Description

Definition at line 34 of file structures.h.

3.19.2 Field Documentation

3.19.2.1 double* content

Definition at line 35 of file structures.h.

Referenced by relxmlReadUnion(), and unrelation().

3.19.2.2 int len

Definition at line 38 of file structures.h.

Referenced by relxmlReadUnion(), and unrelation().

3.19.2.3 int* len_Content

Definition at line 39 of file structures.h.

Referenced by relxmlReadUnion(), and unrelation().

3.19.2.4 char** source

Definition at line 36 of file structures.h.

Referenced by relxmlReadUnion().

3.19.2.5 char** target

Definition at line 37 of file structures.h.

Referenced by relxmlReadUnion().

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

- **structures.h**

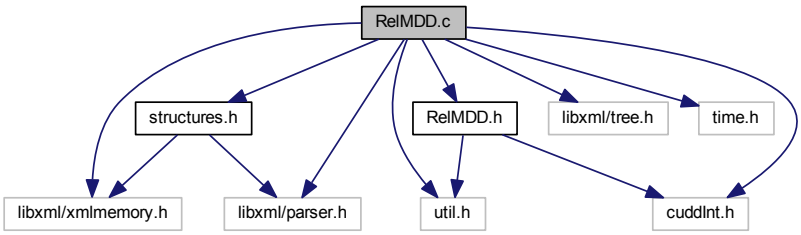
Chapter 4

File Documentation

4.1 RelMDD.c File Reference

```
#include "util.h"
#include "cuddInt.h"
#include "structures.h"
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>
#include <libxml/tree.h>
#include <time.h>
#include "RelMDD.h"
```

Include dependency graph for RelMDD.c:



Macros

- #define **MAXA** 500

Functions

- double **setSequence** (double value)
- DdNode ∗ **node_Complement** (DdNode ∗mat)
- void **RelMDD_Int** ()
- void **RelMDD_Quit** ()
- **RelMDDConversePtr RelMDDConverse_Operation** (int tra_rn, int tra_col, char ∗∗source_object, char ∗∗target_object, double ∗matrimx)
- CUDD_VALUE_TYPE **transposeRelation** (CUDD_VALUE_TYPE F)
- DdNode ∗ **Transpose** (DdManager ∗**dd**, DdNode ∗f)

- **RelMDDComplementPtr RelMDDComplement_Operation** (int com_Row, int com_Col, char ∗∗com_-Rowlist, char ∗∗com_Collist, double ∗matrimx)
- CUDD_VALUE_TYPE **complementRelation** (CUDD_VALUE_TYPE F)
- DdNode ∗ **Complement** (DdManager ∗**dd**, DdNode ∗f)
- **RelMDDIntersectionPtr RelMDDintersectOperation** (int no_rows, int no_cols, char ∗∗source_Object, char ∗∗target_object, double ∗matrix1, double ∗matrix2)
- CUDD_VALUE_TYPE **interrelation** (CUDD_VALUE_TYPE F, CUDD_VALUE_TYPE G)
- DdNode ∗ **Cudd_addIntersection** (DdManager ∗**dd**, DdNode ∗∗f, DdNode ∗∗g)
- **RelMDDUnionPtr RelMDDUnionOperation** (int no_rows, int no_cols, char ∗∗source_Object, char ∗∗target-_object, double ∗matrix1, double ∗matrix2)
- CUDD_VALUE_TYPE **unrelation** (CUDD_VALUE_TYPE F, CUDD_VALUE_TYPE G)
- DdNode ∗ **Cudd_addUnion_Meet** (DdManager ∗**dd**, DdNode ∗∗f, DdNode ∗∗g)
- DdNode ∗ **RelMDD_RenameRelation** (DdManager ∗**dd**, int roww, int collh)
- int ∗ **RelMDD_ListSequence** (DdNode ∗E, int roww, int coll)
- DdNode ∗ **RelMDD_SwapVariables** (DdManager ∗**dd**, int roww, int collh)
- **RelMDDIdentityPtr RelMDDIdentityRelation** (int row, int col, char ∗∗object)
- CUDD_VALUE_TYPE **identityRelation** (CUDD_VALUE_TYPE F)
- **RelMDDBottomPtr RelMDDBottomRelation** (int row, int col, char ∗∗source, char ∗∗target)
- CUDD_VALUE_TYPE **BottomRelation** (CUDD_VALUE_TYPE F)
- **RelMDDTopPtr RelMDDTopRelation** (int row, int col, char ∗∗source, char ∗∗target)
- CUDD_VALUE_TYPE **topRelation** (CUDD_VALUE_TYPE F)
- DdNode ∗ **RelMDD_CreateCompostionVariables** (DdManager ∗**dd**, int roww, int collh, int r, int c)
- double ∗∗ **multiplication** (int row, int col)
- **RelMDDCompositionPtr RelMDDCompositionOperation** (int no_rowofMat1, int no_colsofMat1, int no_-colsofMat2, char ∗∗source_Object, char ∗∗intermidiate_Object, char ∗∗target_Object2, double ∗matrix1, double ∗matrix2)

Variables

- DdManager ∗ **dd**
- DdNode ∗ **bck**
- static double ∗∗ **holdconverse**
- static double ∗∗ **holdcomplement**
- static double ∗∗ **holdMeet**
- static double ∗∗ **holdJoin**
- static double ∗∗ **holdId**
- static double ∗∗ **holdBottom**
- static double ∗∗ **holdTop**
- static double ∗∗ **value_comp**
- int ∗ **ini**
- static int **no_vars**
- static int **cno_vars**
- DdNode ∗ **tempNode**
- static char ∗∗ **idSource**
- static char ∗∗ **idTarget**
- static int **id_id_track**
- static int ∗ **inifor_id**
- static int **id_row**
- static int **id_col**
- static char ∗∗ **BottomSource**
- static char ∗∗ **BottomTarget**
- static int **Bottom_track**
- static int ∗ **inifor_Bottom**
- static int **Bottom_row**

- static int **Bottom_col**
- static char ** **topSource**
- static char ** **topTarget**
- static int **Top_track**
- static int * **inifor_Top**
- static int **Top_row**
- static int **Top_col**
- static int **mrow**
- static int **mcol**
- static int **tran_col**
- static int **tran_row**
- static char ** **compleSource**
- static char ** **compleTarget**
- static char ** **tran_Source**
- static char ** **tran_Target**
- static int **track**
- static int **track_trans**
- static int * **iniforcompl**
- static int * **iniforTrans**
- static double * **ComplementMat2**
- static double * **TranposeMat**
- static DdNode ** **summation_var**
- static int **track_union**
- static int **track_Inters**
- static int * **iniforUnion**
- static int * **iniforInters**
- static double * **unionMat1**
- static double * **UnionMat2**
- static double * **interMat1**
- static double * **interMat2**
- static char ** **unionSource**
- static char ** **unionTarget**
- static char ** **interSource**
- static char ** **interTarget**
- static int **urol**
- static int **ucol**
- static int **interrol**
- static int **interrcol**

4.1.1 Macro Definition Documentation

4.1.1.1 #define MAXA 500

CFile

Definition at line 22 of file RelMDD.c.

Referenced by RelMDDCompositionOperation().

4.1.2 Function Documentation

4.1.2.1 CUDD_VALUE_TYPE BottomRelation (CUDD_VALUE_TYPE *F*)

function:CUDD_VALUE_TYPE **BottomRelation(CUDD_VALUE_TYPE F)** (p. **??**) creates the the Bottom relation relation: returns the a value from the xml file internal file

Definition at line 1287 of file RelMDD.c.

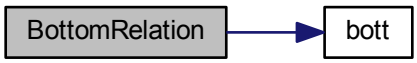
References bott(), Bottom_col, Bottom_row, Bottom_track, BottomSource, BottomTarget, holdBottom, inifor_-Bottom, bottom::len, and bottom::rel.

Referenced by RelMDDBottomRelation().

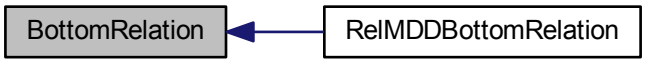
```
{

    int c;
    char *source_Object1[Bottom_row];
    char *target_Object2[Bottom_col];

    for (c = 0; c < Bottom_row; c++) {

        source_Object1[c] = BottomSource[c];

    }
    for (c = 0; c < Bottom_col; c++) {

        target_Object2[c] = BottomTarget[c];

    }

    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;

    CUDD_VALUE_TYPE value=0;
    int ste[listob1][listob2];
    bottomPtr het;
    het = (bottomPtr) bott();
    int size_bot = het->len;
    int  j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            ste[i][k] = yy;
            yy++;
            //printf("%d",ste[i][k]);
        }
    }

    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            //  if(ste[i][k]==Bottom_Bottom_trackt[Bottom_Bottom_track]){
            if (ste[i][k] == inifor_Bottom[Bottom_track]) {
                //printf("Bottom_Bottom_track%d%d\n", Bottom_track,
                //       inifor_Bottom[Bottom_track]);
                //printf("\n%d\t%d\n", i, k);
                rn = i;
                cn = k;         //
            }

        }
    }

    /*  if (rn == cn) {
       for (j = 0; j < size_struct; j++) {

       if (!(strcmp(source_Object1[cn], *ret[j].object))) {

       printf("Bottom \t%s \t %f\n ", *ret[j].object,
       ret->rel[j]);
       Bottom[rn][rn] = ret->rel[j];
       }

       }
       }
     */
    // else {
```

```
    for (j = 0; j < size_bot; j++) {
        if (!(strcmp(source_Object1[rn], *het[j].source))
            && !(strcmp(target_Object2[cn], *het[j].target))) {
           // printf(".........%s\t%s\n", *het[j].source, *het[j].target);
            // rel_list1[l] = (double)cur->rel[j];
            //perror("gome");
           // printf(".........%f\t\n", het->rel[j]);
            holdBottom[cn][rn] = het->rel[j];   //check the assignment for cn
       and rn
            // l++;

            // }
        }
    }
    Bottom_track--;

    return value;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.2 DdNode* Complement (DdManager * *dd,* DdNode * *f*)

function: *Complement(DdManager * dd, DdNode * f) It does the ADD manipulation of Complement operation. It is called by Cudd_addMonadicApply(); to compute the the complement of a relation. It returns the complement of an element from the file It is an Internal fucntion.

Definition at line 515 of file RelMDD.c.

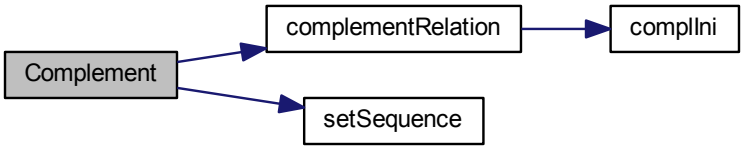
References bck, complementRelation(), and setSequence().

Referenced by node_Complement(), and RelMDDComplement_Operation().

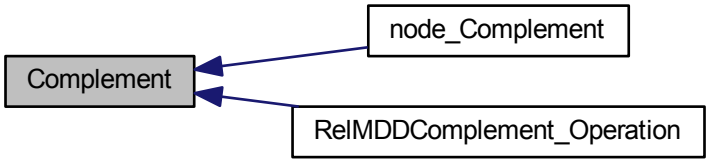
```
{
    CUDD_VALUE_TYPE value=0;
    DdNode *res;
    if (f == bck) {
        return (bck);
    }

    else {
        if (cuddIsConstant(f) || f == 0) {
            complementRelation(cuddV(f));
            value = setSequence(value);
```

```
            res = cuddUniqueConst(dd, value);
            return (res);
        }
    }
    return (NULL);

}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.3 CUDD_VALUE_TYPE complementRelation (CUDD_VALUE_TYPE *F*)

function: CUDD_VALUE_TYPE **complementRelation(CUDD_VALUE_TYPE F)** (p. **??**) This function does the complement operations by going into the file and selecting the complement of a given element matching it with the source and target object it is called by DdNode DdNode ∗Complement(DdManager ∗ dd, DdNode ∗ f) to perform manipulations of DD. it returns a CUDD_VALUE_TYPE of the relation or index It is an Internal fucntion.

Definition at line 424 of file RelMDD.c.

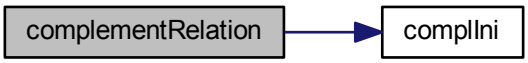
References ComplementMat2, compleSource, compleTarget, complIni(), complement::content, holdcomplement, iniforcompl, complement::len, complement::len_Content, mcol, mrow, and track.

Referenced by Complement().

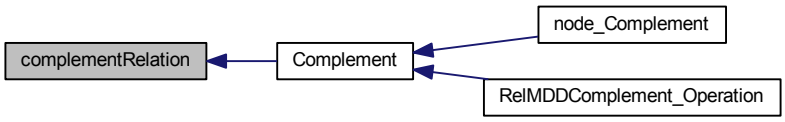
```
{

    CUDD_VALUE_TYPE value=0;
    int c;

    char *source_Object1[mrow];
    char *target_Object2[mcol];
    for (c = 0; c < mrow; c++) {

        source_Object1[c] = compleSource[c];
```

```
    }

    for (c = 0; c < mcol; c++) {

        target_Object2[c] = compleTarget[c];

    }

    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;
    int ste[listob1][listob2];

    complementPtr ret;
    ret = (complementPtr)complIni();
    int size_struct = ret->len;
    int  l, j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            ste[i][k] = yy;
            yy++;
            //printf("%d",ste[i][k]);
        }
    }

    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            //  if(ste[i][k]==trackt[track]){
            if (ste[i][k] == iniforcompl[track]) {
                //printf("track%d\n", track);
                //  printf("\n%d\t%d\n", i,k);
                rn = i;
                cn = k;          //
            }

        }
    }


    for (j = 0; j < size_struct; j++) {

        if (!(strcmp(source_Object1[rn], *ret[j].source))
            && !(strcmp(target_Object2[cn], *ret[j].target))) {
          //  printf("\n%s\t%s\n ", *ret[j].source, *ret[j].target);
            for (l = 0; l < ret->len_Content[j]; l++) {
                int ma = (int) (F - 1);
                if (l % 2 == 0) {
                    if (ret[j].content[l] == ComplementMat2[ma]) {
                        holdcomplement[rn][cn]= ret[j].content[l+1];
                        //printf("\n%f\n", holdcomplement[ma]);
                        value = F;
                        //printf("\n..%f \t%d\t %f\n", F, ma, pop);
                        track--;
                    }
                }
                //}

                //   printf("\n" );
            }

        }
    }                            //}

    return value;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.4 DdNode* Cudd_addIntersection (DdManager * *dd,* DdNode ** *f,* DdNode ** *g*)

function: DdNode *Cudd_addIntersection(DdManager * dd, DdNode ** f, DdNode ** g) It does the ADD manipulation of meet operation. It is called by Cudd_addApply(); to compute the the meet of a relation. It returns the meet of an element from the file It is an Internal fucntion.

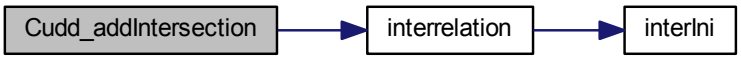
Definition at line 706 of file RelMDD.c.

References bck, and interrelation().

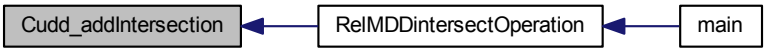
Referenced by RelMDDintersectOperation().

```
{
    DdNode *res;
    DdNode *F, *G;
    CUDD_VALUE_TYPE value=0;
    F = *f;
    G = *g;
    //if ((F == DD_ZERO(dd))||(G == DD_ZERO(dd))) return(DD_ZERO(dd));
    // if (G == DD_ZERO(dd)) return(F);
    if ((F == bck) || (G == bck)) {
        return (bck);
    }

    else {
        if (cuddIsConstant(F) && cuddIsConstant(G)) {
           value = interrelation(cuddV(F), cuddV(G));
            res = cuddUniqueConst(dd, value);
            return (res);
        }
    }
    if (F > G) {                /* swap f and g */
        *f = G;
        *g = F;
    }
    return (NULL);

}                               /* end of Cudd_addPlus */
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.5 DdNode* Cudd_addUnion_Meet (DdManager * *dd*, DdNode ** *f*, DdNode ** *g*)

function:DdNode *Cudd_addUnion_Meet(DdManager * dd, DdNode ** f, DdNode ** g) It does the ADD manipulation of meet operation. It is called by Cudd_addApply(); to compute the the join of a relation. It returns the join of an element from the file It is an Internal fucntion.

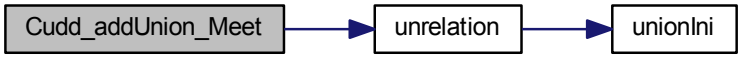
Definition at line 913 of file RelMDD.c.

References bck, and unrelation().

Referenced by RelMDDUnionOperation().

```
{
    DdNode *res;
    DdNode *F, *G;
    CUDD_VALUE_TYPE value=0;
    F = *f;
    G = *g;
    if ((F == bck) || (G == bck)) {
        return (bck);
    } else {
        if (cuddIsConstant(F) && cuddIsConstant(G)) {
            value = unrelation(cuddV(F), cuddV(G));
            res = cuddUniqueConst(dd, value);
            return (res);
        }
    }
    if (F > G) {                /* swap f and g */
        *f = G;
        *g = F;
    }
    return (NULL);

}                               /* end of Cudd_addPlus */
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.2.6 CUDD_VALUE_TYPE identityRelation (CUDD_VALUE_TYPE *F*)

function:CUDD_VALUE_TYPE **identityRelation(CUDD_VALUE_TYPE F)** (p. **??**) creates the indentity relation

Definition at line 1133 of file RelMDD.c.

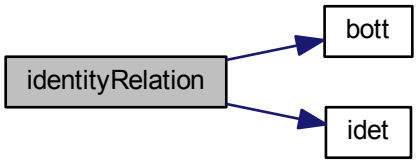
References bott(), holdId, id_col, id_id_track, id_row, idet(), idSource, idTarget, inifor_id, identity::len, bottom::len, identity::rel, and bottom::rel.

Referenced by RelMDDIdentityRelation().

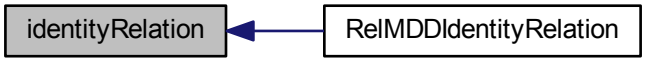
```
{

    int c;
    char *source_Object1[id_row];
    char *target_Object2[id_col];

    for (c = 0; c < id_row; c++) {
        source_Object1[c] = idSource[c];
    }
    for (c = 0; c < id_col; c++) {
        target_Object2[c] = idTarget[c];
    }
    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;
    CUDD_VALUE_TYPE value=0;
    int ste[listob1][listob2];
    identityPtr ret;
    ret = (identityPtr) idet();
    bottomPtr het;
    het = (bottomPtr) bott();
    int size_bot = het->len;
    int size_struct = ret->len;
    int j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            ste[i][k] = yy;
            yy++;
            //printf("%d",ste[i][k]);
        }
    }
```

```
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            //  if(ste[i][k]==id_id_trackt[id_id_track]){
            if (ste[i][k] == inifor_id[id_id_track]) {
                //printf("id_id_track%d%d\n", id_id_track,
                     //  inifor_id[id_id_track]);
                //printf("\n%d\t%d\n", i, k);
                rn = i;
                cn = k;          //
            }

        }
    }

    if (rn == cn) {
        for (j = 0; j < size_struct; j++) {

            if (!(strcmp(source_Object1[cn], *ret[j].object))) {
                //printf("id \t%s \t %f\n ", *ret[j].object, ret->rel[j]);
                 holdId[rn][rn] = ret->rel[j];
            }

        }
    }

    else {
        for (j = 0; j < size_bot; j++) {
            if (!(strcmp(source_Object1[rn], *het[j].source))
                && !(strcmp(target_Object2[cn], *het[j].target))) {
                //printf(".........%s\t%s\n", *het[j].source,
                    //   *het[j].target);
                // rel_list1[l] = (double)cur->rel[j];
                //perror("gome");
                //printf(".........%f\t\n", het->rel[j]);
                holdId[cn][rn] = het->rel[j];   //check the assignment for cn
       and rn
                // l++;

            }
        }
    }
    id_id_track--;

    return value;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.7 CUDD_VALUE_TYPE interrelation (CUDD_VALUE_TYPE *F*, CUDD_VALUE_TYPE *G*)

function: CUDD_VALUE_TYPE **interrelation(CUDD_VALUE_TYPE F, CUDD_VALUE_TYPE G)** (p. 28interrelationsubsubsection.4.1 This function does the meet operations by going into the file and selecting the complement of a given element matching it with the source and target object it is called by Cudd_addIntersection to perform manipulations of DD. it returns a CUDD_VALUE_TYPE of the relation or index It is an Internal fucntion.

Definition at line 625 of file RelMDD.c.

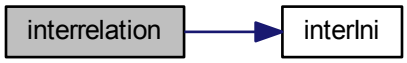
References intersection::content, holdMeet, iniforInters, interIni(), interMat1, interMat2, interrcol, interrol, interSource, interTarget, intersection::len, intersection::len_Content, and track_Inters.

Referenced by Cudd_addIntersection().

```
{
    int c;
    char *source_Object1[interrol];
    char *target_Object2[interrcol];
    for (c = 0; c < interrol; c++) {
        source_Object1[c] = interSource[c];
    }
    for (c = 0; c < interrcol; c++) {
        target_Object2[c] = interTarget[c];
    }
    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;
    int state[listob1][listob2];        // malloc here
    CUDD_VALUE_TYPE value=0;
    intersectionPtr ret;
    ret = (intersectionPtr) interIni();
    int size_struct = ret->len;
    int  l, j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {
        for (k = 0; k < listob2; k++) {
            state[i][k] = yy;
            yy++;
          //  printf("%d",state[i][k]);
        }
    }

    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {
            if (state[i][k] == iniforInters[track_Inters]) {
                //printf("track%d\n", track_Inters);
                //  printf("\n%d\t%d\n", i,k);
                rn = i;
                cn = k;         //
                //  printf("\ndd%fdd\n",result_Union[i]);
                 //printf("\n%s\t%s\n ", source_Object1[rn],
   target_Object2[cn]);
            }

        }

    }

    for (j = 0; j < size_struct; j++) {
```

```
        if (!(strcmp(source_Object1[rn], *ret[j].source))
            && !(strcmp(target_Object2[cn], *ret[j].target))) {
           // printf("\n%s\t%s\n ", *ret[j].source, *ret[j].target);}}

            // printf("\n%f\t%f\t%f\n" ,F,G, value);
            for (l = 0; l < ret->len_Content[j]; l++) {
                int ma = (int) (F - 1);
                int ka = (int) (G - 1);
                if (l % 3 == 0) {
                    if ((ret[j].content[l] == interMat1[ma])
                        && (ret[j].content[l + 1] == interMat2[ka])) {
                        holdMeet[rn][cn] = ret[j].content[l + 2];
                        //printf("\n%f\t%f\t%f\n", F, G, valufe);
                        value = F;
                        track_Inters--;

                    }

                }

            }

        }
    }
    return value;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.2.8 double** multiplication (int *row,* int *col*)

Definition at line 1590 of file RelMDD.c.

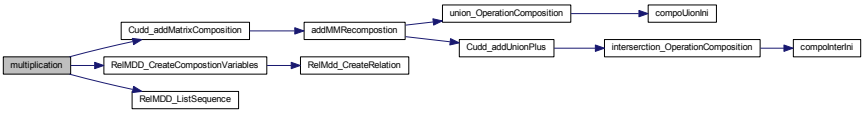
References cno_vars, col_trackComp, Cudd_addMatrixComposition(), dd, element_comp, RelMDD_CreateCompostionVariables(), RelMDD_ListSequence(), row_trackComp, summation_var, tempNode, and value_comp.

Referenced by RelMDDCompositionOperation().

```
                                        {
DdNode *M1;
DdNode *M2;
DdNode *nor;
M1= RelMDD_CreateCompostionVariables(dd,row,col,1,0);
DdNode **yy= summation_var;
M2= RelMDD_CreateCompostionVariables(dd,row,col,0,3);
nor=Cudd_addMatrixComposition(dd,M1,M2,yy,cno_vars);
//Cudd_PrintDebug(dd,nor,10,40);
int *list = RelMDD_ListSequence(nor,row, col);
tempNode =nor;
```

```
int i,r,c; double rel;

for(i=1;i<row*col;i++){

r=row_trackComp[list[i]];
c= col_trackComp[list[i]];
rel= element_comp[list[i]];
value_comp[r][c]=rel;
printf("<<%f>>",value_comp[r][c]);

}

free(row_trackComp);
free(col_trackComp);
free(element_comp);
return value_comp;

}
```

Here is the call graph for this function:



Here is the caller graph for this function:

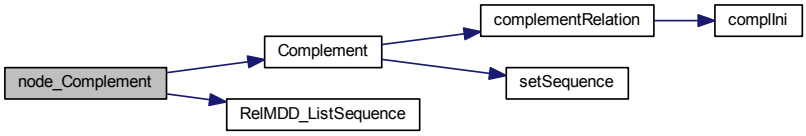
4.1.2.9 DdNode * node_Complement (DdNode * *mat*)

Definition at line 330 of file RelMDD.c.

References Complement(), dd, iniforcompl, mcol, mrow, and RelMDD_ListSequence().

```
{

    iniforcompl = RelMDD_ListSequence(mat, mrow, mcol);
    DdNode *tmp1 = Cudd_addMonadicApply(dd, Complement, mat);
    free(iniforcompl);          // check this if memory link
    return tmp1;
}
```

Here is the call graph for this function:



4.1.2.10 DdNode* RelMDD_CreateCompostionVariables (DdManager * *dd*, int *roww*, int *collh*, int *r*, int *c*)

Definition at line 1530 of file RelMDD.c.

References cno_vars, RelMdd_CreateRelation(), and summation_var.

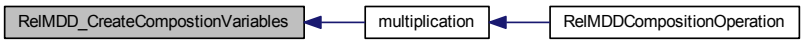
Referenced by multiplication().

```
{
    DdNode *E;
    DdNode **x;
    DdNode **y;
    DdNode **y1;
    DdNode **xn;
    DdNode **xn1;
    DdNode **yn_;
    DdNode **yn_1;
    int nx;
    int ny;
    int maxnx;
    int maxny;
    int m;
    int n;
    int ll;
    int f, j, ww = 0;
    // dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
    int rows[roww * collh];     //malloc
    int cols[roww * collh];     //malloc
    double id_relation[roww * collh];
     double count = 1;
    //int *colum = coll_id( roww);
    //int *rows=  rol_id(coll);
    //int uu [] ={0,0,1,1,2,2};// write methods for this
    //int vv [] ={0,1,0,1,0,1};// write methods for this
    for (f = 0; f < roww; f++) {
        for (j = 0; j < collh; j++) {
            //printf("\n%d%d\n",f,j);
            rows[ww] = f;
            cols[ww] = j;
            //printf("\n%d%d\n",rows[ww],cols[ww]);
            ww++;
        }

    }
    for (ll = 0; ll < roww * collh; ll++) {
        id_relation[ll] = count;
        count = count + 1.00;

    }
    x = y = xn = yn_ = y1 = xn1 = yn_1=  NULL;
    maxnx = maxny = 0;
    nx = maxnx;
    ny = maxny;
   RelMdd_CreateRelation(dd, &E, &x, &y, &xn, &yn_, &nx, &ny, &m, &n, r, roww,
      c, collh, roww, collh, rows, cols, id_relation);
   summation_var = y;
   // Cudd_RecursiveDeref(dd, E);
 // ok = RelMdd_CreateRelation(dd, &F, &y, &y1, &xn1, &yn_1, &nx1, &ny1, &m1,
       &n1, 0, 4, 3, 4, roww, collh, rows, cols, cant);
   //Cudd_PrintDebug(dd,E,nx + ny,40);
   //Cudd_PrintDebug(dd,F,nx1 + ny1,40);
     //
   //  if(ok)
    cno_vars = nx;
```

```
    // node_Complement( E);
    return E;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.1.2.11 void RelMDD_Int ( )

Function: void **RelMDD_int()** (p. **??**); To be used for the initialization of RelMDD by initialization the the cudd package and setting the background Value to PlusInfinity. To use RelMDD you must call this functions to initialize it before any other functions could be called. If this function is not called you will get error like segmentation fualt.

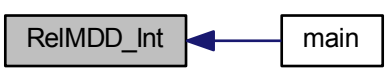
Definition at line 111 of file RelMDD.c.

References bck, and dd.

Referenced by main().

```
{
    dd = Cudd_Init(0, 0, CUDD_UNIQUE_SLOTS, CUDD_CACHE_SLOTS, 0);
    Cudd_AutodynDisable(dd);
    bck = Cudd_ReadPlusInfinity(dd);
    Cudd_SetBackground(dd, bck);
    //RelMDD_intt();
}/*End of void RelMDD_int()*/
```

Here is the caller graph for this function:



4.1.2.12 int* RelMDD_ListSequence (DdNode * *E*, int *roww*, int *coll*)

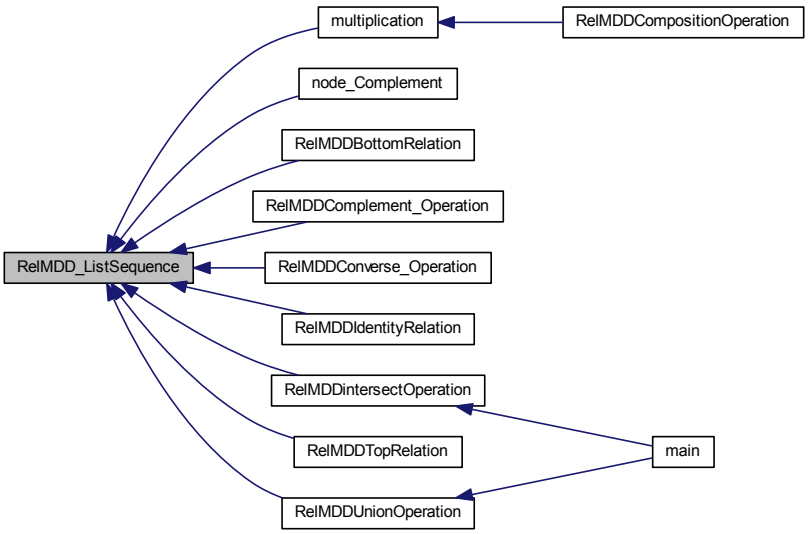
Definition at line 995 of file RelMDD.c.

References dd, ini, no_vars, and t.

Referenced by multiplication(), node_Complement(), RelMDDBottomRelation(), RelMDDComplement_Operation(), RelMDDConverse_Operation(), RelMDDIdentityRelation(), RelMDDintersectOperation(), RelMDDTopRelation(), and RelMDDUnionOperation().

```
{
    // dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
    ini = (int *) malloc(sizeof(int) * roww * coll);    // check here for
       memory leak
    int *cube;
    int t = 0;
    CUDD_VALUE_TYPE value;
    DdGen *gen;
    int q;
    //printf("Testing iterator on cubes:\n");
    Cudd_ForeachCube(dd, E, gen, cube, value) {
        for (q = 0; q < no_vars; q++) {
        }
        //(void) printf(" %g\t",value);
        // trackt[t] = value;
        ini[t] = value;
        t++;
    }
    return ini;
}
```

Here is the caller graph for this function:



4.1.2.13 void RelMDD_Quit ( )

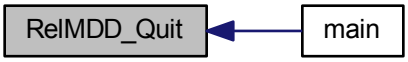
Definition at line 120 of file RelMDD.c.

References dd.

Referenced by main().

```
                {
Cudd_Quit(dd);
}
```

Here is the caller graph for this function:



4.1.2.14 DdNode* RelMDD_RenameRelation (DdManager * *dd,* int *roww,* int *collh*)

function:DdNode *DdNode *RelMDD_RenameRelation(DdManager * dd, int roww, int collh) This function renames relation since for instance 2 with objects A to B is not the same as 2 with object B to A Internal

Definition at line 942 of file RelMDD.c.

References no_vars, and RelMdd_CreateRelation().

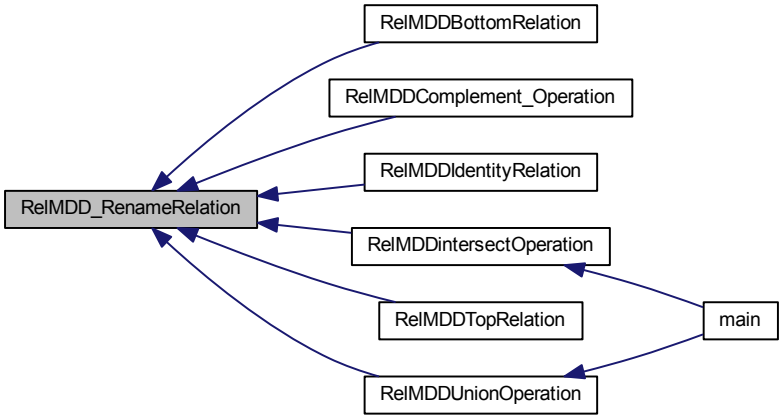
Referenced by RelMDDBottomRelation(), RelMDDComplement_Operation(), RelMDDIdentityRelation(), RelMDDintersectOperation(), RelMDDTopRelation(), and RelMDDUnionOperation().

```
{
    DdNode *E;
    DdNode **x;
    DdNode **y;
    DdNode **xn;
    DdNode **yn_;
    int nx;
    int ny;
    int maxnx;
    int maxny;
    int m;
    int n;
    int ok;
    int ll;
    int f, j, ww = 0;

    // dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
    int rows[roww * collh];     //malloc
    int cols[roww * collh];     //malloc
    double id_relation[roww * collh];
     double count = 1;
    //int *colum = coll_id( roww);
    //int *rows=  rol_id(coll);
    //int uu [] ={0,0,1,1,2,2};// write methods for this
    //int vv [] ={0,1,0,1,0,1};// write methods for this
    for (f = 0; f < roww; f++) {
        for (j = 0; j < collh; j++) {
            //printf("\n%d%d\n",f,j);
            rows[ww] = f;
            cols[ww] = j;
            //printf("\n%d%d\n",rows[ww],cols[ww]);
            ww++;
        }

    }
    for (ll = 0; ll < roww * collh; ll++) {
        id_relation[ll] = count;        //printf(" %f",id_relation[ll]);
        count = count + 1.00;

    }
    x = y = xn = yn_ = NULL;
    maxnx = maxny = 0;
    nx = maxnx;
    ny = maxny;
    ok = RelMdd_CreateRelation(dd, &E, &x, &y, &xn, &yn_, &nx, &ny, &m, &n, 0,
      2, 1, 2, roww, collh, rows, cols, id_relation);  //read matrix 1
     // Cudd_PrintDebug(dd,E,nx + ny,40); //Cudd_RecursiveDeref(dd, E);
     if(ok)
    no_vars = nx;
    // node_Complement( E);
    return E;
```

```
}
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.1.2.15 DdNode* RelMDD_SwapVariables (DdManager * *dd,* int *roww,* int *collh*)

Definition at line 1016 of file RelMDD.c.

References no_vars, and RelMdd_CreateRelation().

Referenced by RelMDDConverse_Operation().

```
{
    DdNode *E;
    DdNode **x;
    DdNode **y;
    DdNode **xn;
    DdNode **yn_;
    int nx;
    int ny;
    int maxnx;
    int maxny;
    int m;
    int n;
    int ll;
    int f, j, ww = 0;
    int cont=0;
    // dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
    int rows[roww * collh];     //malloc
    int cols[roww * collh];     //malloc
    double id_relation[roww * collh];
    int lll = 1;
```

```
    for (f = 0; f < roww; f++) {
        for (j = 0; j < collh; j++) {
            //printf("\n%d%d\n",f,j);
            rows[ww] = f;
            cols[ww] = j;
            //printf("\n%d%d\n",rows[ww],cols[ww]);
            ww++;
        }

    }
    for (ll = 0; ll < roww * collh; ll++) {
        id_relation[ll] = lll;  //printf(" %f",id_relation[ll]);
        lll = lll + 1;

    }
    x = y = xn = yn_ = NULL;
    maxnx = maxny = 0;
    nx = maxnx;
    ny = maxny;
   RelMdd_CreateRelation(dd, &E, &x, &y, &xn, &yn_, &nx, &ny, &m, &n, 0, 2, 1,
      2, roww, collh, rows, cols, id_relation);        //read matrix 1
     Cudd_PrintDebug(dd,E,nx + ny,40);
     //Cudd_RecursiveDeref(dd, E);
   // if(ok)

    if ( nx>ny){
    cont=ny;
    no_vars = ny;
    }

    else {
    cont = nx;
    no_vars = nx;
    }

    DdNode *P = Cudd_addSwapVariables(dd, E, x, y, cont);
    Cudd_PrintDebug(dd,P,nx + ny,40);
    return P;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.1.2.16 RelMDDBottomPtr RelMDDBottomRelation (int *row,* int *col,* char ** *source,* char ** *target*)

functionRelMDDBottomPtr RelMDDBottomRelation(int row, int col, char **source, char **target) Initializes the variables for the for creating the Bottom relation. This is an external functions. It calls other functions and uses them to create the bottom relation based on the object provided. It returns the list of relations, the objectd and the other

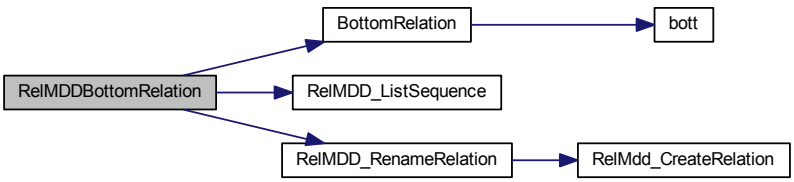
fields defined in the structure RelMDDBottomPtr: row is the size of row of the Matrix col is the size of column of the Matrix followed by list of source and target objects respectively

Definition at line 1225 of file RelMDD.c.

References Bottom_col, Bottom_row, Bottom_track, BottomRelation(), BottomSource, BottomTarget, dd, holdBottom, inifor_Bottom, RelMDDBottom::node, RelMDDBottom::relation, RelMDD_ListSequence(), RelMDD_RenameRelation(), RelMDDBottom::source_Objects, and RelMDDBottom::target_Objects.

```
{
    int move;
    holdBottom = (double **) calloc(row*col*2, sizeof(double*));
    for(move=0;move<row*col*2;move++){
    holdBottom[move] =(double *) calloc(row*col*2, sizeof(double*));
    }
    RelMDDBottomPtr ret;
    ret =(RelMDDBottomPtr) malloc(sizeof(RelMDDBottomPtr) * row*col);
    ret->relation = (double *) malloc(sizeof(double) * row*col*2);
    ret->source_Objects =(char *) malloc(sizeof(char) * row*col*2);
    ret->target_Objects =(char *) malloc(sizeof(char) * row*col*2);
    ret->node = (DdNode *) malloc(sizeof(DdNode) * row*col);
    Bottom_row = row;
    Bottom_col = col;
    Bottom_track = (row * col) - 1;

    BottomSource = (char **) malloc(sizeof(char) *row * col);
    BottomTarget = (char **) malloc(sizeof(char) * row * col);
    for (move = 0; move < Bottom_row; move++) {

        BottomSource[move] = source[move];
        ret[move].source_Objects=source[move];
        //printf("p%s\t%dp\n", BottomSource[move], move);

    }

    for (move = 0; move < Bottom_col; move++) {

        BottomTarget[move] = target[move];
        ret[move].target_Objects=target[move];
        //printf("p%s\t%dp\n", BottomTarget[move], move);

    }
    int i, j,u=0;

    inifor_Bottom =
        RelMDD_ListSequence(RelMDD_RenameRelation(dd, Bottom_row, Bottom_col),
      Bottom_row, Bottom_col);
   // for (i = 0; i < row * col; i++) {
        //printf("%d\t", inifor_Bottom[i]);

   // }
    for (i = 0; i < row * col; i++) {
        BottomRelation(inifor_Bottom[i]);
    }
    ret->node= RelMDD_RenameRelation(dd, Bottom_row, Bottom_col);
    for (j = 0; j < Bottom_row; j++) {
        for (i = 0; i < Bottom_col; i++) {
            ret->relation[u]=holdBottom[i][j];
            u++;
        }
        //printf("\n");
    }
    return ret;
}
```

Here is the call graph for this function:



4.1.2.17 RelMDDComplementPtr RelMDDComplement_Operation (int *com_Row,* int *com_Col,* char ** *com_Rowlist,* char ** *com_Collist,* double * *matrimx*)

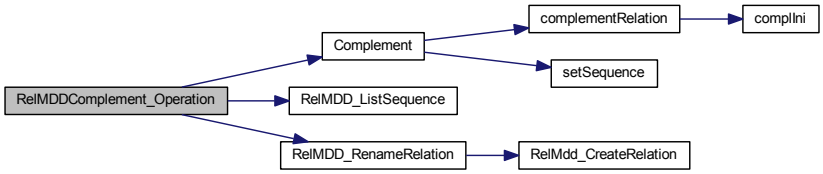
function:RelMDDComplementPtr RelMDDComplement_Operation(int com_Row, int com_Col, char **com_Rowlist, char **com_Collist, double *matrimx) Initializes the variables for the complement operations. This is an external functions. It calls other functions and uses them to compute the complement of a given relation. It returns the list of relations, the source and target objects and the other field defined in the structure RelMDDComplementPtr com_Row is the size of row of the Matrix com_Col is the size of col of the Matrix followed by the source and target object then finally the matrix itself

Definition at line 352 of file RelMDD.c.

References Complement(), ComplementMat2, compleSource, compleTarget, dd, holdcomplement, iniforcompl, mcol, mrow, RelMDDComplement::node, RelMDDComplement::relation, RelMDD_ListSequence(), RelMDD_RenameRelation(), RelMDDComplement::source_Objects, RelMDDComplement::target_Objects, and track.

```
{
    RelMDDComplementPtr ret;
    int move;
    mrow = com_Row;
    mcol = com_Col;
    ComplementMat2 = matrimx;
    track = (com_Row * com_Col) - 1;
    holdcomplement= (double **) calloc(mrow * mcol*2, sizeof(double*));
    for(move=0;move<mrow * mcol;move++){
    holdcomplement[move] =(double *) calloc(mrow * mcol, sizeof(double*));
    }
    ret = (RelMDDComplementPtr) malloc(sizeof(RelMDDComplementPtr) *  mrow *
      mcol*2);
    ret->source_Objects = (char *) malloc(sizeof(char) *  mrow * mcol*2);
    ret->target_Objects = (char *) malloc(sizeof(char)* mrow * mcol*2);
    ret->relation = (double *) malloc(sizeof(double) * mrow * mcol*2);

    compleSource = (char **) malloc(sizeof(char) * com_Row * com_Col*2);
    compleTarget = (char **) malloc(sizeof(char) * com_Row * com_Col*2);

    for (move = 0; move < com_Row; move++) {
        ret[move].source_Objects = com_Rowlist[move];
        //printf("%s\t%d\n", ret[move].source_Objects, move);

        //printf("%f\t%d\n", ComplementMat2[move], move);
        compleSource[move] = com_Rowlist[move];
        //printf("%s\t%d\n", compleSource[move], move);

    }

    for (move = 0; move < com_Col; move++) {

        ret[move].target_Objects = com_Rowlist[move];
        //printf("%s\t%d\n", ret[move].target_Objects, move);

        compleTarget[move] = com_Collist[move];
        //printf("%s\t%d\n", compleTarget[move], move);

    }


    DdNode *M2 = RelMDD_RenameRelation(dd, mrow,mcol);
    iniforcompl = RelMDD_ListSequence(M2,  mrow,   mcol);
    Cudd_addMonadicApply(dd, Complement, M2);
    ret->node=M2;

     int i, j,c =0;
     for(i=0;i<com_Row;i++){

     for(j=0;j<com_Col;j++){

         ret->relation[c]= holdcomplement[i][j];// return the relation here
         //printf("%f",ret->relation[c]);
         c++;
     }
        }
   // ret->relation= holdcomplement;

    return ret;
}/*End of  RelMDDComplementPtr RelMDDComplement_Operation()  */
```

Here is the call graph for this function:



4.1.2.18 RelMDDCompositionPtr RelMDDCompositionOperation (int *no_rowofMat1,* int *no_colsofMat1,* int *no_colsofMat2,* char ∗∗ *source_Object,* char ∗∗ *intermidiate_Object,* char ∗∗ *target_Object2,* double ∗ *matrix1,* double ∗ *matrix2*)

function:RelMDDCompositionPtr **RelMDDCompositionOperation(int no_rowofMat1, int no_colsofMat1, int no_colsofMat2, char ∗∗source_Object, char ∗∗intermidiate_Object, char ∗∗target_Object2, double ∗matrix1, double ∗matrix2)** (p. **??**) Initializes the variables for the compostion operations. This is an external functions. It calls other functions and uses them to compute the join of two given relation or matrices. It returns the list of relations, the source and target objects and the other fields defined in the structure **RelMDDComposition** (p. 13RelMDDComposition Struct Referencesection.3.12): no_rowofMat1 is the size of row of the first Matrix no_colsofMat1, is the size of col of first the Matrix no_colsofMat2, is the size of col of second the Matrix followed by the source and target object first matrix followed by the target object second matrix then finally the first matrix and second matrix respectively

Definition at line 1632 of file RelMDD.c.

References col_comp, col_trackComp, element_comp, mat1, mat2, MAXA, multiplication(), RelMDDComposition::node, RelMDDComposition::relation, row_comp, row_trackComp, source_comp, RelMDDComposition::source_Objects, target_Comp, RelMDDComposition::target_Objects, tempNode, and value_comp.

```
                                                        {
 element_comp = (double *) malloc(sizeof(double) * no_rowofMat1*no_colsofMat1*
      MAXA);
 row_trackComp = (int  *) malloc(sizeof(int) * no_rowofMat1*no_colsofMat1*MAXA)
      ;
col_trackComp = (int *) malloc(sizeof(int) * no_rowofMat1*no_colsofMat1*MAXA);
RelMDDCompositionPtr ret;
    ret =(RelMDDCompositionPtr) malloc(sizeof(RelMDDComposition) * no_rowofMat1
       * no_colsofMat2*2);
    ret->relation = (double *) malloc(sizeof(double) * no_rowofMat1 *
      no_colsofMat2*2);
    ret->source_Objects =(char *) malloc(sizeof(char) * no_rowofMat1 *
      no_colsofMat2*2);
    ret->target_Objects =(char *) malloc(sizeof(char) * no_rowofMat1 *
      no_colsofMat2*2);
    ret->node = (DdNode *) malloc(sizeof(DdNode) * no_rowofMat1 * no_colsofMat2
      *2);
int move;
    value_comp= (double **) calloc(no_rowofMat1*no_colsofMat1+1, sizeof(double
      *));
    for(move=0;move<no_rowofMat1*no_colsofMat1+1;move++){
    value_comp[move] =(double *) calloc(no_rowofMat1*no_colsofMat1+1, sizeof(
      double*));
    }
    for (move = 0; move < no_rowofMat1; move++) {
        ret[move].source_Objects = source_Object[move];
    }
    for (move = 0; move <no_colsofMat2; move++) {
        ret[move].target_Objects =target_Object2[move];
    }
mat1= matrix1;
mat2=matrix2;
source_comp= source_Object;
target_Comp= target_Object2;
row_comp=no_rowofMat1;
col_comp=no_colsofMat2;
double **temp=multiplication(no_rowofMat1 , no_colsofMat2);
```

```
  int i, j, l=0;
 for (j = 0; j < no_rowofMat1; j++) {
        for (i = 0; i < no_colsofMat2; i++) {
          ret->relation[l]=temp[i][j];
          l++;
        }

    }

 ret->node= tempNode;
 return ret;
}
```

Here is the call graph for this function:

4.1.2.19 RelMDDConversePtr RelMDDConverse_Operation (int *tra_rn*, int *tra_col*, char ** *source_object*, char ** *target_object*, double * *matrimx*)

Prototypes decalations of functions in this file. function: **RelMDDConverse_Operation(int tra_rn, int tra_col, char **srelation, char **trelation, double *matrimx)** (p. **??**) Initializes the variables for the converse operations. This is an external functions. It calls other functions and uses them to compute the converse of a given relation. It returns the list of relations, the source and target objects and the other field defined in the structure **RelMDDConverse** (p. 14RelMDDConverse Struct Referencesection.3.13). tran_rn is the size of row of the Matrix tran_col is the size of col of the Matrix followed by the source and target object then finally the matrix itself

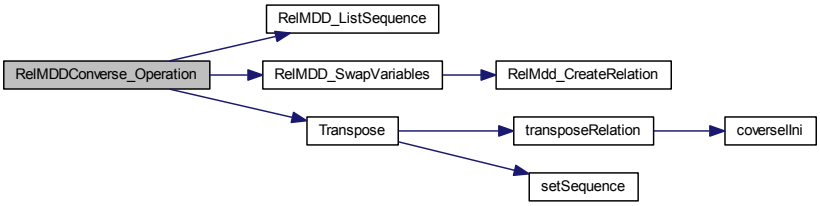
Definition at line 135 of file RelMDD.c.

References dd, holdconverse, iniforTrans, RelMDDConverse::node, RelMDDConverse::relation, RelMDD_ListSequence(), RelMDD_SwapVariables(), RelMDDConverse::source_Objects, RelMDDConverse::target_Objects, track_trans, tran_col, tran_row, tran_Source, tran_Target, TranposeMat, and Transpose().

```
{

    int move;
    int  row = tra_rn;
    int col = tra_col;
    RelMDDConversePtr ret= NULL;
  //  holdconverse = (double *) malloc(sizeof(double) * row *col *2);
    holdconverse= (double **) calloc(row*col*2, sizeof(double*));
    for(move=0;move<row*col*2;move++){
    holdconverse[move] =(double *) calloc(row*col*2, sizeof(double*));
    }
    ret = (RelMDDConversePtr) malloc(sizeof(RelMDDConversePtr)*row *col *2);
    ret->source_Objects= (char*)malloc(sizeof(char)*row *col *2);
    ret->target_Objects= (char*)malloc(sizeof(char)*row *col *2);
    ret->relation= (double*)malloc(sizeof(double)*row *col *2);

    tran_row = tra_rn;
    tran_col = tra_col;
    TranposeMat = matrimx;
    track_trans = (tra_rn * tra_col) - 1;

    tran_Source = (char **) malloc(sizeof(char) * tra_rn*tran_col*2);
    tran_Target = (char **) malloc(sizeof(char) * tra_rn*tran_col*2);
    for (move = 0; move < tra_rn; move++) {
        tran_Source[move] =source_object[move];
        //printf("%s\t%d\n", tran_Source[move], move);
        //ret[move].source_Objects = source_object[move];
        // printf ("%s\t%d\n", tran_Source[move],move);
    }
    for (move = 0; move < tra_col; move++) {

        tran_Target[move] = target_object[move];
        //printf("%s\t%d\n", tran_Target[move], move);

    }
```

```
    DdNode *Matt2;
    DdNode *Matt = RelMDD_SwapVariables(dd, tra_rn, tra_col);// the transpose
       matrix
   //Cudd_PrintDebug(dd,Matt,3,40);
        iniforTrans = RelMDD_ListSequence(Matt, tra_rn, tra_col);

   Matt2= Cudd_addMonadicApply(dd, Transpose, Matt); //creating the new
       relations from the xml file
    ret->node= Matt2; //return the ADD here


    for (move = 0; move < tran_col; move++) {
        ret[move].source_Objects = target_object[move]; // printf ("%s\t%d\n",
       ret[move].source_Objects,move);

         }
     for (move = 0; move <tran_row; move++) {

        ret[move].target_Objects = source_object[move];  //printf ("%s\t%d\n",
       ret[move].target_Objects,move);
         }
     int i, j,c =0;
     for(i=0;i<tran_col;i++){

     for(j=0;j<tran_row;j++){

         ret->relation[c]= holdconverse[i][j];// return the relation here
        //printf("%f",ret->relation[c]);
         c++;
     }
        }
    return ret ;
}/*End of RelMDDConverse_Operation*/
```

Here is the call graph for this function:



4.1.2.20 RelMDDIdentityPtr RelMDDIdentityRelation (int *row,* int *col,* char ** *object*)

function:**RelMDDIdentityRelation(int row, int col, char **object)** (p. **??**) Initializes the variables for the for creating the identity relation. This is an external functions. It calls other functions and uses them to create the identity relation based on the object provided. It returns the list of relations, the objectd and the other fields defined in the structure RelMDDIdentityPtr: row is the size of row of the Matrix col is the size of column of the Matrix followed by list of object

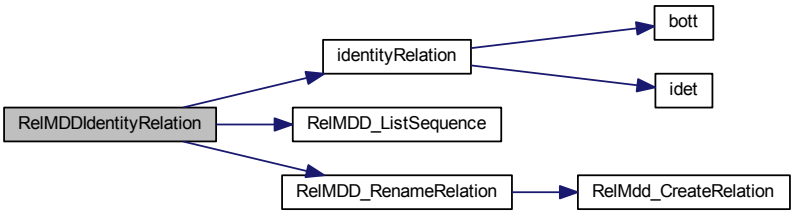
Definition at line 1085 of file RelMDD.c.

References dd, holdId, id_col, id_id_track, id_row, identityRelation(), idSource, idTarget, inifor_id, RelMDDIdentity::node, RelMDDIdentity::object, RelMDDIdentity::relation, RelMDD_ListSequence(), and RelMDD_RenameRelation().

```
{
    int move;
    holdId = (double **) calloc(row*col*2, sizeof(double*));
    for(move=0;move<row*col*2;move++){
    holdId[move] =(double *) calloc(row*col*2, sizeof(double*));
    }
    RelMDDIdentityPtr ret;
    ret =(RelMDDIdentityPtr) malloc(sizeof(RelMDDIdentityPtr) * row*col);
    ret->relation = (double *) malloc(sizeof(double) * row*col*2);
    ret->object =(char *) malloc(sizeof(char) * row*col*2);
    ret->node = (DdNode *) malloc(sizeof(DdNode) * row*col);
```

```
    id_row = row;
    id_col = col;
    id_id_track = (row*col) - 1;
    idSource = (char **) malloc(sizeof(char) * row*col);
    idTarget = (char **) malloc(sizeof(char) * row*col);
    for (move = 0; move<row; move++) {
        idSource[move] = object[move];
        //printf("p%s\t%dp\n", idSource[move], move);
    }
    for (move = 0; move < col; move++) {
        idTarget[move] = object[move];
        ret[move].object = object[move];
        //printf("p%s\t%dp\n", idTarget[move], move);
    }
    int i, j,y=0;

    inifor_id = RelMDD_ListSequence(RelMDD_RenameRelation(dd, id_row, id_col),
      id_row,id_col);
    ret->node=RelMDD_RenameRelation(dd, id_row, id_col);
    for (i = 0; i < row*col; i++) {
        identityRelation(inifor_id[i]);
    }
    for (j = 0; j < row; j++) {
        for (i = 0; i < col; i++) {
            //printf(">>%f>>\t", holdId[i][j]);
            ret->relation[y] =holdId[i][j];
            y++;
        }
        //printf("\n");
    }
    return ret;
}/*End of RelMDDIdentityRelation(int row, int col, char **object)*/
```

Here is the call graph for this function:



4.1.2.21 RelMDDIntersectionPtr RelMDDintersectOperation (int *no_rows,* int *no_cols,* char ** *source_Object,* char ** *target_object,* double * *matrix1,* double * *matrix2*)

function:RelMDDIntersectionPtr RelMDDintersectOperation(int no_rows, int no_cols, char **ssrelation, char **ttrelation, double *matrix1, double *matrix2) Initializes the variables for the intersection operations. This is an external functions. It calls other functions and uses them to compute the meet of two given relation or matrices. It returns the list of relations, the source and target objects and the other fields defined in the structure **RelMDDIntersection** (p. 16RelMDDIntersection Struct Referencesection.3.15): no_rows is the size of row of the Matrix no_cols is the size of col of the Matrix followed by the source and target object then finally the first matrix and second matrix respectively

Definition at line 554 of file RelMDD.c.

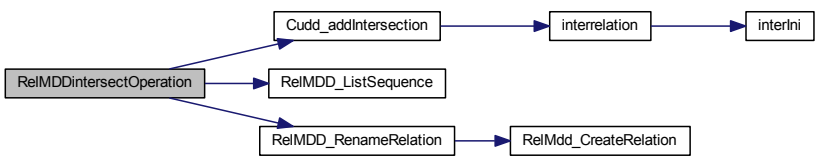
References Cudd_addIntersection(), dd, holdMeet, iniforInters, interMat1, interMat2, interrcol, interrol, interSource, interTarget, RelMDDIntersection::node, RelMDDIntersection::relation, RelMDD_ListSequence(), RelMDD_RenameRelation(), RelMDDIntersection::source_Objects, RelMDDIntersection::target_Objects, and track_Inters.

Referenced by main().

```
{
```

```
    int mrow1= no_rows;
    int mcol1 = no_cols;
    int move;
    holdMeet= (double **) calloc(mrow1 * mcol1*2, sizeof(double*));
    for(move=0;move< mrow1 * mcol1*2;move++){
    holdMeet[move] =(double *) calloc( mrow1 * mcol1*2, sizeof(double*));
    }
    RelMDDIntersectionPtr ret;
    ret =(RelMDDIntersectionPtr) malloc(sizeof(RelMDDIntersection) *mrow1 *
      mcol1*2);
    ret->relation = (double *) malloc(sizeof(double) * mrow1 * mcol1*2);
    ret->source_Objects =(char *) malloc(sizeof(char) * mrow1 * mcol1*2);
    ret->target_Objects =(char *) malloc(sizeof(char) * mrow1 * mcol1*2);
    ret->node = (DdNode *) malloc(sizeof(DdNode) *mrow1 * mcol1*2);
    DdNode *M1;
    DdNode *M2;
    DdNode *tmp;
    int move1;
    track_Inters = (no_cols * no_rows) - 1;
    interMat1 = matrix1;
    interMat2 = matrix2;
    interrol = no_rows;
    interrcol = no_cols;
    interSource = (char **) malloc(sizeof(char) * mrow1 * mcol1*2);
    interTarget = (char **) malloc(sizeof(char) * mrow1 * mcol1*2);
    for (move1 = 0; move1 < mrow1; move1++) {
        interSource[move1] = source_Object[move1];      //
      printf("\n..%s\t%d..\n",interSource[move1],move1);
        ret[move1].source_Objects = source_Object[move1];
    }

      for (move1 = 0; move1 < mcol1; move1++) {
        interTarget[move1] = target_object[move1];      //
      printf("\n..%s\t%d..\n",interTarget[move1],move1);
        ret[move1].target_Objects = target_object[move1];
    }


    // DD mamupulation
    M1 = RelMDD_RenameRelation(dd, no_rows, no_cols);
    Cudd_Ref(M1);
    M2 = RelMDD_RenameRelation(dd, no_rows, no_cols);
    iniforInters = RelMDD_ListSequence(RelMDD_RenameRelation(dd, no_rows,
      no_cols), no_rows, no_cols);
    tmp = Cudd_addApply(dd, Cudd_addIntersection, M1, M2);
    ret->node= tmp;

    int i, j,c =0;
     for(i=0;i<mrow1;i++){

     for(j=0;j<mcol1;j++){

         ret->relation[c]= holdMeet[i][j];// return the relation here
         //printf("%f",holdMeet[i][j]);
         c++;
     }
     //printf("\n");
   }
    return ret;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.2.22 RelMDDTopPtr RelMDDTopRelation (int *row,* int *col,* char ** *source,* char ** *target*)

function: RelMDDBottomPtr RelMDDTopRelation(int row, int col, char **source, char **target) Initializes the variables for the for creating the top relation. This is an external functions. It calls other functions and uses them to create the top relation based on the object provided. It returns the list of relations, the object and the other field members defined in the structure **RelMDDTop** (p. 16RelMDDTop Struct Referencesection.3.16): row is the size of row of the Matrix col is the size of column of the Matrix followed by list of source and target objects respectively

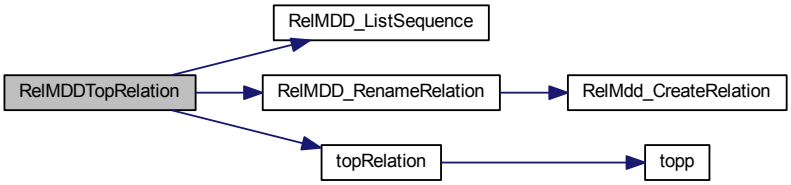
Definition at line 1383 of file RelMDD.c.

References dd, holdTop, inifor_Top, RelMDDTop::node, RelMDDTop::relation, RelMDD_ListSequence(), RelMDD-_RenameRelation(), RelMDDTop::source_Objects, RelMDDTop::target_Objects, Top_col, Top_row, Top_track, top-Relation(), topSource, and topTarget.

```
{
    int move;
    holdTop= (double **) calloc(row*col*2, sizeof(double*));
    for(move=0;move<row*col*2;move++){
    holdTop[move] =(double *) calloc(row*col*2, sizeof(double*));
    }
    RelMDDTopPtr ret;
    ret =(RelMDDTopPtr) malloc(sizeof(RelMDDTopPtr) * row*col);
    ret->relation = (double *) malloc(sizeof(double) * row*col*2);
    ret->source_Objects =(char *) malloc(sizeof(char) * row*col*2);
    ret->target_Objects =(char *) malloc(sizeof(char) * row*col*2);
    ret->node = (DdNode *) malloc(sizeof(DdNode) * row*col);
    Top_row = row;
    Top_col = col;
    Top_track = ( row*col) - 1;
    topSource = (char **) malloc(sizeof(char) *  row*col);
    topTarget = (char **) malloc(sizeof(char) * row*col);
    for (move = 0; move <  row; move++) {

        topSource[move] = source[move];
        ret[move].source_Objects=source[move];
        //printf("p%s\t%dp\n", topSource[move], move);

    }

    for (move = 0; move <  col; move++) {

        topTarget[move] = target[move];
        ret[move].target_Objects=target[move];
        //printf("p%s\t%dp\n", topTarget[move], move);

    }
    int i, j,l=0;

    inifor_Top =
        RelMDD_ListSequence(RelMDD_RenameRelation(dd,Top_row , Top_col ),Top_row
       , Top_col);
    for (i = 0; i < row*col; i++) {
        topRelation(inifor_Top[i]);
    }
     ret->node=RelMDD_RenameRelation(dd,Top_row , Top_col );
    for (j = 0; j < row; j++) {
        for (i = 0; i < col; i++) {
          ret->relation[l]=holdTop[i][j];
          l++;
        }
```

```
    }
    return ret;
}
```

Here is the call graph for this function:



4.1.2.23 RelMDDUnionPtr RelMDDUnionOperation (int *no_rows*, int *no_cols*, char ** *source_Object*, char ** *target_object*, double * *matrix1*, double * *matrix2*)

function:RelMDDUnionPtr RelMDDUnionOperation(int no_rows, int no_cols, char **ssrelation, char **ttrelation, double *matrix1, double *matrix2) Initializes the variables for the join operations. This is an external functions. It calls other functions and uses them to compute the join of two given relation or matrices. It returns the list of relations, the source and target objects and the other fields defined in the structure **RelMDDUnion** (p. 17RelMDD-Union Struct Referencesection.3.17): no_rows is the size of row of the Matrix no_cols is the size of col of the Matrix followed by the source and target object then finally the first matrix and second matrix respectively

Definition at line 744 of file RelMDD.c.

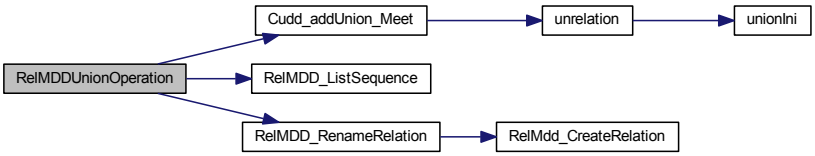
References Cudd_addUnion_Meet(), dd, holdJoin, iniforUnion, RelMDDUnion::node, RelMDDUnion::relation, RelMDD_ListSequence(), RelMDD_RenameRelation(), RelMDDUnion::source_Objects, RelMDDUnion::target_-Objects, track_union, ucol, unionMat1, UnionMat2, unionSource, unionTarget, and urol.

Referenced by main().

```
{
    int unrow = no_rows;
    int uncol = no_cols;
  //  holdJoin = (double *) malloc(sizeof(double) * unrow*uncol*2);
    int move;
    holdJoin= (double **) calloc(unrow * uncol*2, sizeof(double*));
    for(move=0;move< unrow * uncol*2;move++){
    holdJoin[move] =(double *) calloc( unrow * uncol*2, sizeof(double*));
    }
    RelMDDUnionPtr ret;

    ret =
        (RelMDDUnionPtr) malloc(sizeof(RelMDDUnionPtr) * unrow *uncol *2);
    ret->relation = (double *) malloc(sizeof(double) * unrow*uncol*2);
    ret->source_Objects =
        (char *) malloc(sizeof(char) * unrow*uncol);
    ret->target_Objects =
        (char *) malloc(sizeof(char) * unrow*uncol);
    ret->node = (DdNode *) malloc(sizeof(DdNode) *unrow*uncol*2);
    int move1;
    track_union = (unrow*uncol) - 1;
    unionMat1 = matrix1;
    UnionMat2 = matrix2;
    urol = no_rows;
    ucol = no_cols;
   // track_union = (no_rows * no_cols) - 1;
    unionSource = (char **) malloc(sizeof(char) * unrow*uncol);
    unionTarget = (char **) malloc(sizeof(char) * unrow*uncol);
    for (move1 = 0; move1 < unrow; move1++) {
        unionSource[move1] = source_Object[move1];      //
      printf("%s\t%d\n",unionSource[move1],move1);
        ret[move1].source_Objects = source_Object[move1];       // printf
       ("%s\t%d\n", ret[move1].source_Objects,move1);
```

```
    }
    for (move1 = 0; move1 < uncol; move1++) {
        unionTarget[move1] = target_object[move1];      //
      printf("%s\t%d\n",unionTarget[move1],move1);
        ret[move1].target_Objects = target_object[move1];       // printf
       ("%s\t%d\n", ret[move].target_Objects,move);

    }

    DdNode *M1;
    DdNode *M2;
    DdNode *tmp;
    M1 = RelMDD_RenameRelation(dd, unrow, uncol);
    Cudd_Ref(M1);
    //Cudd_RecursiveDeref(manager,f);
    M2 = RelMDD_RenameRelation(dd,  unrow, uncol);
    iniforUnion =
        RelMDD_ListSequence(RelMDD_RenameRelation(dd,  unrow, uncol),
                            unrow, uncol);
    tmp = Cudd_addApply(dd, Cudd_addUnion_Meet, M1, M2);
    ret->node = tmp;

     int i, j,c =0;
     for(i=0;i<unrow;i++){

     for(j=0;j<uncol;j++){

         ret->relation[c]= holdJoin[i][j];// return the relation here
        printf("%f",ret->relation[c]);
         c++;
     }
        }

     if (ret == NULL) {
        printf("\n out of memory\n");
        return NULL;

    }
    return ret;
}/*End RelMDDUnionOperation */
```

Here is the call graph for this function:



Here is the caller graph for this function:

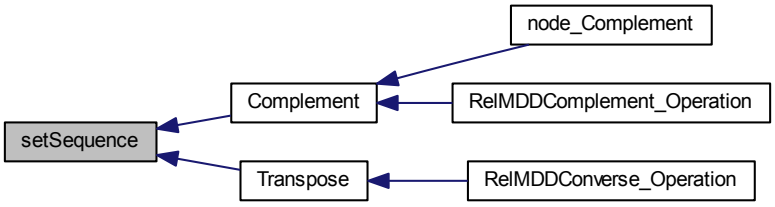
4.1.2.24 double setSequence (double *value*)

Definition at line 535 of file RelMDD.c.

Referenced by Complement(), and Transpose().

```
{

    return value;

}
```

Here is the caller graph for this function:



4.1.2.25 CUDD_VALUE_TYPE topRelation (CUDD_VALUE_TYPE *F*)

function: CUDD_VALUE_TYPE **topRelation(CUDD_VALUE_TYPE F)** (p. **??**) creates the the top relation relation: returns the a value from the xml file internal file

Definition at line 1442 of file RelMDD.c.

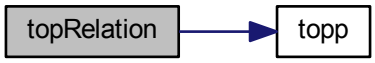
References holdTop, inifor_Top, top::len, top::rel, Top_col, Top_row, Top_track, topp(), topSource, and topTarget.

Referenced by RelMDDTopRelation().

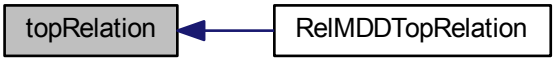
```
{

    int c;
    char *source_Object1[Top_row];
    char *target_Object2[Top_col];

    for (c = 0; c < Top_row; c++) {

        source_Object1[c] = topSource[c];

    }
    for (c = 0; c < Top_col; c++) {

        target_Object2[c] = topTarget[c];

    }

    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;

    CUDD_VALUE_TYPE value=0;
    int ste[listob1][listob2];
    topPtr het;
    het = (topPtr) topp();
    int size_bot = het->len;
    int  j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            ste[i][k] = yy;
            yy++;
            //printf("%d",ste[i][k]);
        }
```

```
    }

    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            //  if(ste[i][k]==Top_Top_trackt[Top_Top_track]){
            if (ste[i][k] == inifor_Top[Top_track]) {
                //printf("Top_Top_track%d%d\n", Top_track,
                     //  inifor_Top[Top_track]);
                //printf("\n%d\t%d\n", i, k);
                rn = i;
                cn = k;          //
            }

        }
    }

    /*    if(rn==cn) {
       for(j = 0; j<size_struct; j++){

       if( !(strcmp(source_Object1[cn], *ret[j].object  ))){

       printf("Top \t%s \t %f\n ",*ret[j].object, ret->rel[j]);
       toppp[rn][rn]=ret->rel[j];
       }

       }}
     */
//                              else{
    for (j = 0; j < size_bot; j++) {
        if (!(strcmp(source_Object1[rn], *het[j].source))
            && !(strcmp(target_Object2[cn], *het[j].target))) {
           // printf(".........%s\t%s\n", *het[j].source, *het[j].target);
            // rel_list1[l] = (double)cur->rel[j];
            //perror("gome");
           // printf(".........%f\t\n", het->rel[j]);
            holdTop[cn][rn] = het->rel[j];      //check the assignment for cn
       and rn
            // l++;

        }
    }
//}
    Top_track--;

    return value;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.26 DdNode* Transpose (DdManager * *dd*, DdNode * *f*)

function: *Transpose(DdManager * dd, DdNode * f) It does the ADD manipulation of Coverse operation. It is called by Cudd_addMonadicApply(); to compute the the transpose of a relation. It is an Internal fucntion.

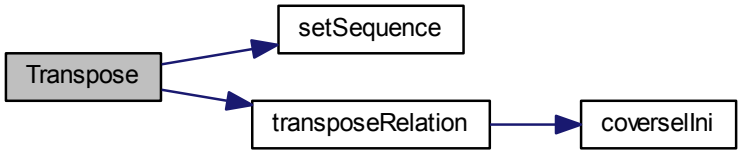
Definition at line 307 of file RelMDD.c.

References bck, setSequence(), and transposeRelation().

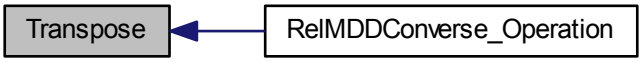
Referenced by RelMDDConverse_Operation().

```
{
    CUDD_VALUE_TYPE value=0;
    DdNode *res;
    if (f == bck) {
        return (bck);
    }

    else {
        if (cuddIsConstant(f) || f == 0) {
           transposeRelation(cuddV(f));
           //printf("<<%f>>", cuddV(f));
            value = setSequence(value);
            res = cuddUniqueConst(dd, value);
            return (res);
        }
    }
    return (NULL);

}/*Tranpose*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.27 CUDD_VALUE_TYPE transposeRelation (CUDD_VALUE_TYPE *F*)

function: **transposeRelation(CUDD_VALUE_TYPE F)** (p. **??**) This function does the converse operations by going into the file and selecting the converse of a given element matching it with the source and target object it is called by DdNode *Transpose(DdManager * dd, DdNode * f) to perform manipulation of DD. It is an Internal fucntion.

Definition at line 212 of file RelMDD.c.

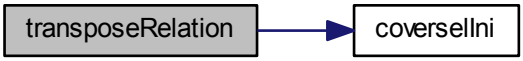
References converse::content, coverselIni(), holdconverse, iniforTrans, converse::len, converse::len_Content, track_trans, tran_col, tran_row, tran_Source, tran_Target, and TranposeMat.

Referenced by Transpose().

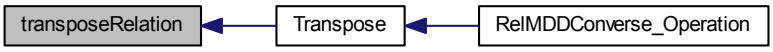
```
{
    int c;

    char *source_Object1[tran_col];//source object
    char *target_Object2[tran_row];// target object
    for (c = 0; c < tran_col; c++) {

        source_Object1[c] = tran_Target[c]; // initialization of source
        //printf("\n++%s++\n ", source_Object1[c]);

    }
    for (c = 0; c < tran_row ; c++) {

        target_Object2[c] = tran_Source[c]; // initialization of target
      // printf("\n++%s++\n ",target_Object2[c] );
    }

//char *source_Object1[]={"A","B","A","A"};
//char *target_Object2[]={"A","A","B","B"};

    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;
    CUDD_VALUE_TYPE value=0;
    int ste[listob2][listob1];
    conversePtr ret;
    ret = (conversePtr) coverselIni();
    int size_struct = ret->len;
    int  l, j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
   for (i = 0; i < listob2; i++) {

        for (k = 0; k < listob1; k++) {

            ste[i][k] = yy;
            yy++;
        // printf("\nN%d%d%dN\n",i,k,ste[i][k]);
        }
    }

    for (i = 0; i < listob2; i++) {

        for (k = 0; k < listob1; k++) {

            //  if(ste[i][k]==trackt[track]){
            if (ste[i][k] == iniforTrans[track_trans]) {
                //printf("track%d\n", track_trans);
                // printf("\n%d\t%d\n", i,k);

                rn = i;
                cn = k;
                //printf("\n%d\t%d\t%f\n", k,i, F);
                //printf("\n++%s\t%s++\n ", source_Object1[cn],
       target_Object2[rn]);
            }

        }
    }

    for (j = 0; j < size_struct; j++) {

        if (!(strcmp(source_Object1[cn], *ret[j].source))
            && !(strcmp(target_Object2[rn], *ret[j].target))) {
            //printf("\n++%s\t%s++\n ", *ret[j].source, *ret[j].target);
            for (l = 0; l < ret->len_Content[j]; l++) {
                int ma = (int) (F - 1);
                if (l % 2 == 0) {
                    if (ret[j].content[l] == TranposeMat[ma]) {
                    // printf("<<%f\t%f>>",F,TranposeMat[ma]);
                        holdconverse[cn][rn]= ret[j].content[l+1];
                        // printf("<<%d\t%f\t>>",ma,holdconverse[cn][rn]);
                        value = F;
                        track_trans--;
                    }
                }
                //}

                //   printf("\n" );
```

```
            }

        }
    }

//track_trans--;
    return value;
}/*End of transposeRelation(CUDD_VALUE_TYPE F)*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.28 CUDD_VALUE_TYPE unrelation (CUDD_VALUE_TYPE *F*, CUDD_VALUE_TYPE *G*)

function: CUDD_VALUE_TYPE **unrelation(CUDD_VALUE_TYPE F, CUDD_VALUE_TYPE G)** (p. **??**) This function does the join operations by going into the file and selecting the join of a given element matching it with the source and target object it is called by DdNode ∗Cudd_addUnion_Meet(DdManager ∗ dd, DdNode ∗∗ f, DdNode ∗∗ g) to perform manipulations of DD. it returns a CUDD_VALUE_TYPE of the relation or index It is an Internal fucntion.

Definition at line 827 of file RelMDD.c.

References unions::content, holdJoin, iniforUnion, unions::len, unions::len_Content, track_union, ucol, unionIni(), unionMat1, UnionMat2, unionSource, unionTarget, and urol.

Referenced by Cudd_addUnion_Meet().

```
{
    int c;
    char *source_Object1[urol];
    char *target_Object2[ucol];
    for (c = 0; c < urol; c++) {

        source_Object1[c] = unionSource[c];

    }
    for (c = 0; c < ucol; c++) {

        target_Object2[c] = unionTarget[c];

    }

    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;
    int ste[listob1][listob2];
    CUDD_VALUE_TYPE value=0;
```

```
    unionsPtr ret;
    ret = (unionsPtr)unionIni();
    int size_struct = ret->len;
    int  l, j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            ste[i][k] = yy;
            yy++;
            //printf("%d",ste[i][k]);
        }
    }

    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            //  if(ste[i][k]==trackt[track]){
            if (ste[i][k] == iniforUnion[track_union]) {
                //printf("track%d\n", track_union);
                //  printf("\n%d\t%d\n", i,k);
                rn = i;
                cn = k;          //

            }

        }
    }

    for (j = 0; j < size_struct; j++) {

        if (!(strcmp(source_Object1[rn], *ret[j].source))
            && !(strcmp(target_Object2[cn], *ret[j].target))) {
           // printf("\n%s\t%s\n ", *ret[j].source, *ret[j].target);
            // printf("\n%f\t%f\t%f\n" ,F,G, value);
            for (l = 0; l < ret->len_Content[j]; l++) {
                int ma = (int) (F - 1);
                int ka = (int) (G - 1);
                if (l % 3 == 0) {
                    if ((ret[j].content[l] == unionMat1[ma])
                        && (ret[j].content[l + 1] == UnionMat2[ka])) {
                        holdJoin[rn][cn] = ret[j].content[l + 2];
                        value = F;
                        track_union--;

                    }

                }

            }

        }
    }
    return value;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.1.3 Variable Documentation

4.1.3.1 DdNode* bck

Definition at line 24 of file RelMDD.c.

Referenced by Complement(), Cudd_addIntersection(), Cudd_addUnion_Meet(), RelMDD_int(), RelMDD_Int(), and Transpose().

4.1.3.2 int Bottom_col [static]

Definition at line 53 of file RelMDD.c.

Referenced by BottomRelation(), and RelMDDBottomRelation().

4.1.3.3 int Bottom_row [static]

Definition at line 52 of file RelMDD.c.

Referenced by BottomRelation(), and RelMDDBottomRelation().

4.1.3.4 int Bottom_track [static]

Definition at line 49 of file RelMDD.c.

Referenced by BottomRelation(), and RelMDDBottomRelation().

4.1.3.5 char** BottomSource [static]

Definition at line 47 of file RelMDD.c.

Referenced by BottomRelation(), and RelMDDBottomRelation().

4.1.3.6 char** BottomTarget [static]

Definition at line 48 of file RelMDD.c.

Referenced by BottomRelation(), and RelMDDBottomRelation().

4.1.3.7 int cno_vars [static]

Definition at line 35 of file RelMDD.c.

Referenced by multiplication(), and RelMDD_CreateCompostionVariables().

4.1.3.8 double* ComplementMat2 [static]

Definition at line 78 of file RelMDD.c.

Referenced by complementRelation(), and RelMDDComplement_Operation().

4.1.3.9 char** compleSource [static]

Definition at line 70 of file RelMDD.c.

Referenced by complementRelation(), and RelMDDComplement_Operation().

4.1.3.10 char** compleTarget [static]

Definition at line 71 of file RelMDD.c.

Referenced by complementRelation(), and RelMDDComplement_Operation().

4.1.3.11 DdManager* dd

Definition at line 23 of file RelMDD.c.

Referenced by multiplication(), node_Complement(), RelMDD_Int(), RelMDD_int(), RelMDD_ListSequence(), RelMDD_Quit(), RelMDDBottomRelation(), RelMDDComplement_Operation(), RelMDDConverse_Operation(), RelMDDIdentityRelation(), RelMDDintersectOperation(), RelMDDTopRelation(), and RelMDDUnionOperation().

4.1.3.12 double** holdBottom [static]

Definition at line 30 of file RelMDD.c.

Referenced by BottomRelation(), and RelMDDBottomRelation().

4.1.3.13 double** holdcomplement [static]

Definition at line 26 of file RelMDD.c.

Referenced by complementRelation(), and RelMDDComplement_Operation().

4.1.3.14 double** holdconverse [static]

Definition at line 25 of file RelMDD.c.

Referenced by RelMDDConverse_Operation(), and transposeRelation().

4.1.3.15 double** holdId [static]

Definition at line 29 of file RelMDD.c.

Referenced by identityRelation(), and RelMDDIdentityRelation().

4.1.3.16 double** holdJoin [static]

Definition at line 28 of file RelMDD.c.

Referenced by RelMDDUnionOperation(), and unrelation().

4.1.3.17 double** holdMeet [static]

Definition at line 27 of file RelMDD.c.

Referenced by interrelation(), and RelMDDintersectOperation().

4.1.3.18 double** holdTop [static]

Definition at line 31 of file RelMDD.c.

Referenced by RelMDDTopRelation(), and topRelation().

4.1.3.19 int id_col [static]

Definition at line 44 of file RelMDD.c.

Referenced by identityRelation(), and RelMDDIdentityRelation().

4.1.3.20 int id_id_track [static]

Definition at line 41 of file RelMDD.c.

Referenced by identityRelation(), and RelMDDIdentityRelation().

4.1.3.21 int id_row [static]

Definition at line 43 of file RelMDD.c.

Referenced by identityRelation(), and RelMDDIdentityRelation().

4.1.3.22 char** idSource [static]

Definition at line 39 of file RelMDD.c.

Referenced by identityRelation(), and RelMDDIdentityRelation().

4.1.3.23 char** idTarget [static]

Definition at line 40 of file RelMDD.c.

Referenced by identityRelation(), and RelMDDIdentityRelation().

4.1.3.24 int* ini

Definition at line 33 of file RelMDD.c.

Referenced by RelMDD_ListSequence().

4.1.3.25 int* inifor_Bottom [static]

Definition at line 50 of file RelMDD.c.

Referenced by BottomRelation(), and RelMDDBottomRelation().

4.1.3.26 int* inifor_id [static]

Definition at line 42 of file RelMDD.c.

Referenced by identityRelation(), and RelMDDIdentityRelation().

4.1.3.27 int* inifor_Top [static]

Definition at line 60 of file RelMDD.c.

Referenced by RelMDDTopRelation(), and topRelation().

4.1.3.28 int* iniforcompl [static]

Definition at line 76 of file RelMDD.c.

Referenced by complementRelation(), node_Complement(), and RelMDDComplement_Operation().

4.1.3.29 int* iniforInters [static]

Definition at line 87 of file RelMDD.c.

Referenced by interrelation(), and RelMDDintersectOperation().

4.1.3.30 int* iniforTrans [static]

Definition at line 77 of file RelMDD.c.

Referenced by RelMDDConverse_Operation(), and transposeRelation().

4.1.3.31 int* iniforUnion [static]

Definition at line 86 of file RelMDD.c.

Referenced by RelMDDUnionOperation(), and unrelation().

4.1.3.32 double* interMat1 [static]

Definition at line 90 of file RelMDD.c.

Referenced by interrelation(), and RelMDDintersectOperation().

4.1.3.33 double* interMat2 [static]

Definition at line 91 of file RelMDD.c.

Referenced by interrelation(), and RelMDDintersectOperation().

4.1.3.34 int interrcol [static]

Definition at line 99 of file RelMDD.c.

Referenced by interrelation(), and RelMDDintersectOperation().

4.1.3.35 int interrol [static]

Definition at line 98 of file RelMDD.c.

Referenced by interrelation(), and RelMDDintersectOperation().

4.1.3.36 char** interSource [static]

Definition at line 94 of file RelMDD.c.

Referenced by interrelation(), and RelMDDintersectOperation().

4.1.3.37 char** interTarget [static]

Definition at line 95 of file RelMDD.c.

Referenced by interrelation(), and RelMDDintersectOperation().

4.1.3.38 int mcol [static]

Definition at line 67 of file RelMDD.c.

Referenced by complementRelation(), node_Complement(), and RelMDDComplement_Operation().

4.1.3.39 int mrow [static]

Definition at line 66 of file RelMDD.c.

Referenced by complementRelation(), node_Complement(), and RelMDDComplement_Operation().

4.1.3.40 int no_vars [static]

Definition at line 34 of file RelMDD.c.

Referenced by RelMDD_ListSequence(), RelMDD_RenameRelation(), and RelMDD_SwapVariables().

4.1.3.41 DdNode** summation_var [static]

Definition at line 81 of file RelMDD.c.

Referenced by multiplication(), and RelMDD_CreateCompostionVariables().

4.1.3.42 DdNode* tempNode

Definition at line 36 of file RelMDD.c.

Referenced by multiplication(), and RelMDDCompositionOperation().

4.1.3.43 int Top_col [static]

Definition at line 62 of file RelMDD.c.

Referenced by RelMDDTopRelation(), and topRelation().

4.1.3.44 int Top_row [static]

Definition at line 61 of file RelMDD.c.

Referenced by RelMDDTopRelation(), and topRelation().

4.1.3.45 int Top_track [static]

Definition at line 59 of file RelMDD.c.

Referenced by RelMDDTopRelation(), and topRelation().

4.1.3.46 char** topSource [static]

Definition at line 57 of file RelMDD.c.

Referenced by RelMDDTopRelation(), and topRelation().

4.1.3.47 char** topTarget [static]

Definition at line 58 of file RelMDD.c.

Referenced by RelMDDTopRelation(), and topRelation().

4.1.3.48 int track [static]

Definition at line 74 of file RelMDD.c.

Referenced by complementRelation(), and RelMDDComplement_Operation().

4.1.3.49 int track_Inters [static]

Definition at line 85 of file RelMDD.c.

Referenced by interrelation(), and RelMDDintersectOperation().

4.1.3.50 int track_trans [static]

Definition at line 75 of file RelMDD.c.

Referenced by RelMDDConverse_Operation(), and transposeRelation().

4.1.3.51 int track_union [static]

Definition at line 84 of file RelMDD.c.

Referenced by RelMDDUnionOperation(), and unrelation().

4.1.3.52 int tran_col [static]

Definition at line 68 of file RelMDD.c.

Referenced by RelMDDConverse_Operation(), and transposeRelation().

4.1.3.53 int tran_row [static]

Definition at line 69 of file RelMDD.c.

Referenced by RelMDDConverse_Operation(), and transposeRelation().

4.1.3.54 char** tran_Source [static]

Definition at line 72 of file RelMDD.c.

Referenced by RelMDDConverse_Operation(), and transposeRelation().

4.1.3.55 char** tran_Target [static]

Definition at line 73 of file RelMDD.c.

Referenced by RelMDDConverse_Operation(), and transposeRelation().

4.1.3.56 double* TranposeMat [static]

Definition at line 79 of file RelMDD.c.

Referenced by RelMDDConverse_Operation(), and transposeRelation().

4.1.3.57 int ucol [static]

Definition at line 97 of file RelMDD.c.

Referenced by RelMDDUnionOperation(), and unrelation().

4.1.3.58 double* unionMat1 [static]

Definition at line 88 of file RelMDD.c.

Referenced by RelMDDUnionOperation(), and unrelation().

4.1.3.59 double* UnionMat2 [static]

Definition at line 89 of file RelMDD.c.

Referenced by RelMDDUnionOperation(), and unrelation().

4.1.3.60 char** unionSource [static]

Definition at line 92 of file RelMDD.c.

Referenced by RelMDDUnionOperation(), and unrelation().

4.1.3.61 char** unionTarget [static]

Definition at line 93 of file RelMDD.c.

Referenced by RelMDDUnionOperation(), and unrelation().

4.1.3.62 int urol [static]

Definition at line 96 of file RelMDD.c.

Referenced by RelMDDUnionOperation(), and unrelation().

4.1.3.63 double** value_comp [static]

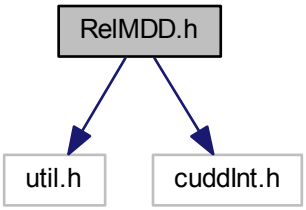
Definition at line 32 of file RelMDD.c.

Referenced by multiplication(), and RelMDDCompositionOperation().

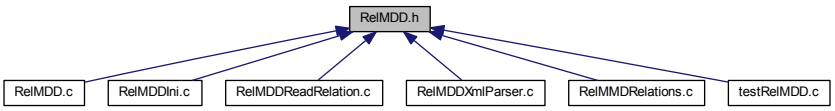
4.2 RelMDD.h File Reference

```
#include "util.h"
#include "cuddInt.h"
```

Include dependency graph for RelMDD.h:



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



Data Structures

- struct **RelMDDUnion**
- struct **RelMDDIntersection**
- struct **RelMDDComplement**
- struct **RelMDDConverse**
- struct **RelMDDIdentity**
- struct **RelMDDBottom**
- struct **RelMDDTop**
- struct **RelMDDComposition**
- struct **fileInfor**

Typedefs

- typedef struct **RelMDDUnion RelMDDUnion**
- typedef struct **RelMDDUnion** ∗ **RelMDDUnionPtr**
- typedef struct **RelMDDIntersection RelMDDIntersection**
- typedef struct **RelMDDIntersection** ∗ **RelMDDIntersectionPtr**
- typedef struct **RelMDDComplement RelMDDComplement**
- typedef struct **RelMDDComplement** ∗ **RelMDDComplementPtr**
- typedef struct **RelMDDConverse RelMDDConverse**
- typedef struct **RelMDDConverse** ∗ **RelMDDConversePtr**
- typedef struct **RelMDDIdentity RelMDDIdentity**
- typedef struct **RelMDDIdentity** ∗ **RelMDDIdentityPtr**
- typedef struct **RelMDDBottom RelMDDBottom**
- typedef struct **RelMDDBottom** ∗ **RelMDDBottomPtr**
- typedef struct **RelMDDTop RelMDDTop**
- typedef struct **RelMDDTop** ∗ **RelMDDTopPtr**
- typedef struct **RelMDDComposition RelMDDComposition**
- typedef struct **RelMDDComposition** ∗ **RelMDDCompositionPtr**
- typedef struct **fileInfor fileInfor**
- typedef struct **fileInfor** ∗ **fileInforPtr**

Functions

- void **RelMDD_Int** ()
- void **RelMDD_Quit** ()
- DdNode ∗ **Transpose** (DdManager ∗**dd**, DdNode ∗f)
- CUDD_VALUE_TYPE **transposeRelation** (CUDD_VALUE_TYPE F)
- DdNode ∗ **RelMDD_SwapVariables** (DdManager ∗**dd**, int roww, int collh)
- DdNode ∗ **Complement** (DdManager ∗**dd**, DdNode ∗f)
- CUDD_VALUE_TYPE **complementRelation** (CUDD_VALUE_TYPE F)
- **RelMDDComplementPtr RelMDDComplement_Operation** (int toi, int gom, char ∗∗srelation, char ∗∗trelation, double ∗matrimx)
- **RelMDDConversePtr RelMDDConverse_Operation** (int tra_rn, int tra_col, char ∗∗source_Object, char ∗∗target_object, double ∗matrimx)
- **fileInforPtr RelMdd_ReadFile** (char ∗filename)
- double **setSequence** (double value)
- DdNode ∗ **RelMDD_RenameRelation** (DdManager ∗**dd**, int roww, int collh)
- int ∗ **RelMDD_ListSequence** (DdNode ∗E, int roww, int coll)
- DdNode ∗ **Cudd_addUnion_Meet** (DdManager ∗**dd**, DdNode ∗∗f, DdNode ∗∗g)
- DdNode ∗ **Cudd_addIntersection** (DdManager ∗**dd**, DdNode ∗∗f, DdNode ∗∗g)
- **RelMDDIntersectionPtr RelMDDintersectOperation** (int no_rows, int no_cols, char ∗∗source_Object, char ∗∗target_object, double ∗matrix1, double ∗matrix2)
- **RelMDDUnionPtr RelMDDUnionOperation** (int no_rows, int no_cols, char ∗∗source_Object, char ∗∗target_object, double ∗matrix1, double ∗matrix2)
- DdNode ∗ **node_Complement** (DdNode ∗mat)
- CUDD_VALUE_TYPE **unrelation** (CUDD_VALUE_TYPE F, CUDD_VALUE_TYPE G)
- int **RelMdd_CreateRelation** (DdManager ∗**dd**, DdNode ∗∗E, DdNode ∗∗∗x, DdNode ∗∗∗y, DdNode ∗∗∗xn, DdNode ∗∗∗yn_, int ∗nx, int ∗ny, int ∗m, int ∗n, int bx, int sx, int by, int sy, int row_nn, int col_nn, int ∗row_n, int ∗col_n, double ∗id_relation)
- CUDD_VALUE_TYPE **identityRelation** (CUDD_VALUE_TYPE F)
- **RelMDDIdentityPtr RelMDDIdentityRelation** (int row, int col, char ∗∗object)
- **RelMDDBottomPtr RelMDDBottomRelation** (int row, int col, char ∗∗source, char ∗∗target)
- CUDD_VALUE_TYPE **BottomRelation** (CUDD_VALUE_TYPE F)
- **RelMDDTopPtr RelMDDTopRelation** (int row, int col, char ∗∗source, char ∗∗target)
- CUDD_VALUE_TYPE **topRelation** (CUDD_VALUE_TYPE F)

- DdNode * **RelMDD_CreateCompostionVariables** (DdManager *__dd__, int roww, int collh, int r, int c)
- **RelMDDCompositionPtr RelMDDCompositionOperation** (int no_rowofMat1, int no_colsofMat1, int no_colsofMat2, char **source_Object, char **intermidiate_Object, char **target_Object2, double *matrix1, double *matrix2)
- double ** **multiplication** (int row, int col)
- void **poo** ()
- **compoUnionPtr compoUionIni** ()
- **conversePtr coverselIni** ()
- **compoInterPtr compoInterIni** ()
- **complementPtr complIni** ()
- **intersectionPtr interIni** ()
- **unionsPtr unionIni** ()
- **compoInterPtr test** ()
- **identityPtr idet** ()
- **bottomPtr bott** ()
- **topPtr topp** ()

Variables

- DdManager * **dd**
- DdNode * **bck**
- unsigned int **cacheSize**
- unsigned int **nvars**
- unsigned int **nslots**
- DdNode * **tempNode**

4.2.1 Typedef Documentation

4.2.1.1 typedef struct fileInfor fileInfor

4.2.1.2 typedef struct fileInfor * fileInforPtr

4.2.1.3 typedef struct RelMDDBottom RelMDDBottom

4.2.1.4 typedef struct RelMDDBottom * RelMDDBottomPtr

4.2.1.5 typedef struct RelMDDComplement RelMDDComplement

4.2.1.6 typedef struct RelMDDComplement * RelMDDComplementPtr

4.2.1.7 typedef struct RelMDDComposition RelMDDComposition

4.2.1.8 typedef struct RelMDDComposition * RelMDDCompositionPtr

4.2.1.9 typedef struct RelMDDConverse RelMDDConverse

4.2.1.10 typedef struct RelMDDConverse * RelMDDConversePtr

4.2.1.11 typedef struct RelMDDIdentity RelMDDIdentity

4.2.1.12 typedef struct RelMDDIdentity * RelMDDIdentityPtr

4.2.1.13 typedef struct RelMDDIntersection RelMDDIntersection

4.2.1.14 typedef struct RelMDDIntersection * RelMDDIntersectionPtr

4.2.1.15 typedef struct RelMDDTop RelMDDTop

4.2.1.16 typedef struct RelMDDTop * RelMDDTopPtr

4.2.1.17 typedef struct RelMDDUnion RelMDDUnion

4.2.1.18 typedef struct RelMDDUnion * RelMDDUnionPtr

4.2.2 Function Documentation

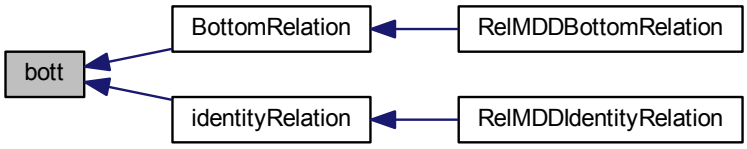
4.2.2.1 bottomPtr bott ( )

Definition at line 1207 of file RelMDDXmlParser.c.

Referenced by BottomRelation(), and identityRelation().

```
{

   // xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    bottomPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadBottom(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the caller graph for this function:



4.2.2.2 CUDD_VALUE_TYPE BottomRelation (CUDD_VALUE_TYPE *F*)

function:CUDD_VALUE_TYPE **BottomRelation(CUDD_VALUE_TYPE F)** (p. **??**) creates the the Bottom relation relation: returns the a value from the xml file internal file

Definition at line 1287 of file RelMDD.c.

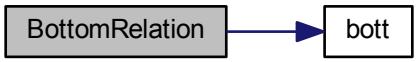
References bott(), Bottom_col, Bottom_row, Bottom_track, BottomSource, BottomTarget, holdBottom, inifor_-Bottom, bottom::len, and bottom::rel.

Referenced by RelMDDBottomRelation().

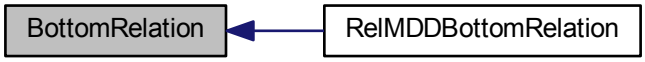
```
{

    int c;
    char *source_Object1[Bottom_row];
    char *target_Object2[Bottom_col];

    for (c = 0; c < Bottom_row; c++) {

        source_Object1[c] = BottomSource[c];

    }
    for (c = 0; c < Bottom_col; c++) {

        target_Object2[c] = BottomTarget[c];

    }

    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;

    CUDD_VALUE_TYPE value=0;
    int ste[listob1][listob2];
    bottomPtr het;
    het = (bottomPtr) bott();
    int size_bot = het->len;
    int  j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            ste[i][k] = yy;
            yy++;
            //printf("%d",ste[i][k]);
        }
    }

    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            //  if(ste[i][k]==Bottom_Bottom_trackt[Bottom_Bottom_track]){
            if (ste[i][k] == inifor_Bottom[Bottom_track]) {
                //printf("Bottom_Bottom_track%d%d\n", Bottom_track,
                //       inifor_Bottom[Bottom_track]);
                //printf("\n%d\t%d\n", i, k);
                rn = i;
                cn = k;         //
            }

        }
    }

    /*  if (rn == cn) {
       for (j = 0; j < size_struct; j++) {

       if (!(strcmp(source_Object1[cn], *ret[j].object))) {

       printf("Bottom \t%s \t %f\n ", *ret[j].object,
       ret->rel[j]);
       Bottom[rn][rn] = ret->rel[j];
       }

       }
       }
     */
    // else {
    for (j = 0; j < size_bot; j++) {
        if (!(strcmp(source_Object1[rn], *het[j].source))
            && !(strcmp(target_Object2[cn], *het[j].target))) {
           // printf(".........%s\t%s\n", *het[j].source, *het[j].target);
            // rel_list1[l] = (double)cur->rel[j];
            //perror("gome");
           // printf(".........%f\t\n", het->rel[j]);
            holdBottom[cn][rn] = het->rel[j];   //check the assignment for cn
       and rn
            // l++;

            // }
```

```
        }
    }
    Bottom_track--;

    return value;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.2.3 DdNode* Complement (DdManager * *dd,* DdNode * *f*)

function: *Complement(DdManager * dd, DdNode * f) It does the ADD manipulation of Complement operation. It is called by Cudd_addMonadicApply(); to compute the the complement of a relation. It returns the complement of an element from the file It is an Internal fucntion.

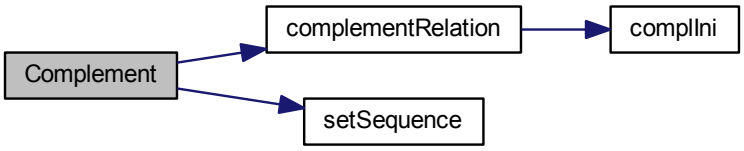
Definition at line 515 of file RelMDD.c.

References bck, complementRelation(), and setSequence().

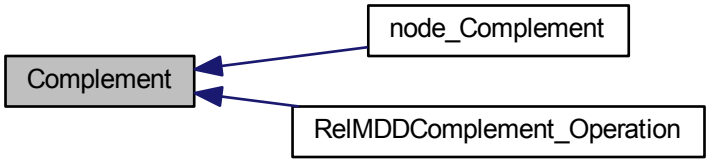
Referenced by node_Complement(), and RelMDDComplement_Operation().

```
{
    CUDD_VALUE_TYPE value=0;
    DdNode *res;
    if (f == bck) {
        return (bck);
    }

    else {
        if (cuddIsConstant(f) || f == 0) {
            complementRelation(cuddV(f));
            value = setSequence(value);
            res = cuddUniqueConst(dd, value);
            return (res);
        }
    }
    return (NULL);

}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.2.4 CUDD_VALUE_TYPE complementRelation (CUDD_VALUE_TYPE *F*)

function: CUDD_VALUE_TYPE **complementRelation(CUDD_VALUE_TYPE F)** (p. **??**) This function does the complement operations by going into the file and selecting the complement of a given element matching it with the source and target object it is called by DdNode DdNode ∗Complement(DdManager ∗ dd, DdNode ∗ f) to perform manipulations of DD. it returns a CUDD_VALUE_TYPE of the relation or index It is an Internal fucntion.

Definition at line 424 of file RelMDD.c.

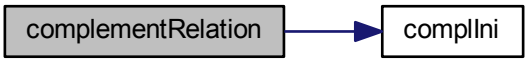
References ComplementMat2, compleSource, compleTarget, complIni(), complement::content, holdcomplement, iniforcompl, complement::len, complement::len_Content, mcol, mrow, and track.

Referenced by Complement().

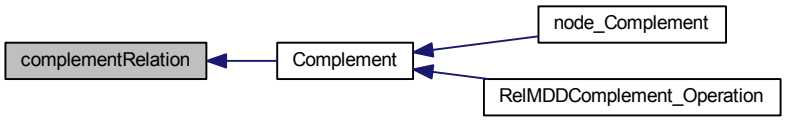
```
{

    CUDD_VALUE_TYPE value=0;
    int c;

    char *source_Object1[mrow];
    char *target_Object2[mcol];
    for (c = 0; c < mrow; c++) {

        source_Object1[c] = compleSource[c];

    }

    for (c = 0; c < mcol; c++) {

        target_Object2[c] = compleTarget[c];

    }

    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
```

```
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;
    int ste[listob1][listob2];

    complementPtr ret;
    ret = (complementPtr)complIni();
    int size_struct = ret->len;
    int  l, j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            ste[i][k] = yy;
            yy++;
            //printf("%d",ste[i][k]);
        }
    }

    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            //  if(ste[i][k]==trackt[track]){
            if (ste[i][k] == iniforcompl[track]) {
                //printf("track%d\n", track);
                //  printf("\n%d\t%d\n", i,k);
                rn = i;
                cn = k;          //
            }

        }
    }


    for (j = 0; j < size_struct; j++) {

        if (!(strcmp(source_Object1[rn], *ret[j].source))
            && !(strcmp(target_Object2[cn], *ret[j].target))) {
          //  printf("\n%s\t%s\n ", *ret[j].source, *ret[j].target);
            for (l = 0; l < ret->len_Content[j]; l++) {
                int ma = (int) (F - 1);
                if (l % 2 == 0) {
                    if (ret[j].content[l] == ComplementMat2[ma]) {
                        holdcomplement[rn][cn]= ret[j].content[l+1];
                        //printf("\n%f\n", holdcomplement[ma]);
                        value = F;
                        //printf("\n..%f \t%d\t %f\n", F, ma, pop);
                        track--;
                    }
                }
                //}

                //   printf("\n" );
            }

        }
    }                            //}

    return value;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



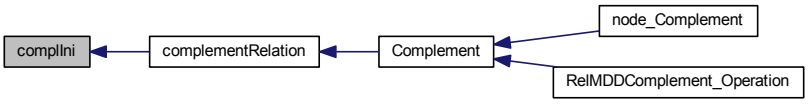
4.2.2.5 complementPtr complIni ( )

Definition at line 1086 of file RelMDDXmlParser.c.

Referenced by complementRelation().

```
{

    //xmlDocPtr doc;
    //doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    complementPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadComplement(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.....................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the caller graph for this function:



4.2.2.6 compoInterPtr compoInterIni ( )

Definition at line 1303 of file RelMDDXmlParser.c.

Referenced by interserction_OperationComposition().

```
{
```

```
//xmlDocPtr doc;
 // doc = xmlParseFile("w3.xml");

  if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL ;
  root = xmlDocGetRootElement(xmlDocument);



   compoInterPtr ret;


   // memset(ret, 0, sizeof(identityPtr));
 ret=  relxmlReadCompostion_Inter(  root,xmlDocument);
  //int k,kk;
// printf("\n.................. in main.................................\n ");

 //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}
 if( !root ||
      !root->name ||
      xmlStrcmp(root->name, (xmlChar *)"RelationBasis") )
  {
     xmlFreeDoc(xmlDocument);
     return NULL;
  }

return ret;
 }
```

Here is the caller graph for this function:

4.2.2.7 compoUnionPtr compoUionIni ( )

Definition at line 1264 of file RelMDDXmlParser.c.

Referenced by union_OperationComposition().

```
                          {

//xmlDocPtr doc;
  //doc = xmlParseFile("w3.xml");

  if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL ;
  root = xmlDocGetRootElement(xmlDocument);



   compoUnionPtr ret;


   // memset(ret, 0, sizeof(identityPtr));
 ret=  relxmlReadCompostion_Union(  root,xmlDocument);
  //int k,kk;
// printf("\n.................. in main.................................\n ");

 //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}
 if( !root ||
      !root->name ||
      xmlStrcmp(root->name,(xmlChar *)"RelationBasis") )
  {
```

```
    xmlFreeDoc(xmlDocument);
    return NULL;
  }

return ret;
 }
```

Here is the caller graph for this function:

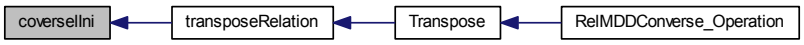
4.2.2.8 conversePtr coverselIni ( )

Definition at line 1058 of file RelMDDXmlParser.c.

Referenced by transposeRelation().

```
{

    //xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument== NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    conversePtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadTransposition(root,xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the caller graph for this function:



4.2.2.9 DdNode* Cudd_addIntersection (DdManager * *dd*, DdNode ** *f*, DdNode ** *g*)

function: DdNode *Cudd_addIntersection(DdManager * dd, DdNode ** f, DdNode ** g) It does the ADD manipulation of meet operation. It is called by Cudd_addApply(); to compute the the meet of a relation. It returns the meet of an element from the file It is an Internal fucntion.

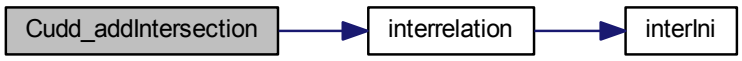
Definition at line 706 of file RelMDD.c.

References bck, and interrelation().

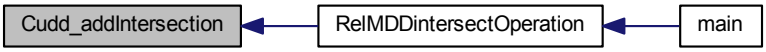
Referenced by RelMDDintersectOperation().

```
{
    DdNode *res;
    DdNode *F, *G;
    CUDD_VALUE_TYPE value=0;
    F = *f;
    G = *g;
    //if ((F == DD_ZERO(dd))||(G == DD_ZERO(dd))) return(DD_ZERO(dd));
    // if (G == DD_ZERO(dd)) return(F);
    if ((F == bck) || (G == bck)) {
        return (bck);
    }

    else {
        if (cuddIsConstant(F) && cuddIsConstant(G)) {
           value = interrelation(cuddV(F), cuddV(G));
            res = cuddUniqueConst(dd, value);
            return (res);
        }
    }
    if (F > G) {                /* swap f and g */
        *f = G;
        *g = F;
    }
    return (NULL);

}                               /* end of Cudd_addPlus */
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.2.10 DdNode* Cudd_addUnion_Meet (DdManager * *dd*, DdNode ** *f*, DdNode ** *g*)

function:DdNode *Cudd_addUnion_Meet(DdManager * dd, DdNode ** f, DdNode ** g) It does the ADD manipulation of meet operation. It is called by Cudd_addApply(); to compute the the join of a relation. It returns the join of an element from the file It is an Internal fucntion.

Definition at line 913 of file RelMDD.c.

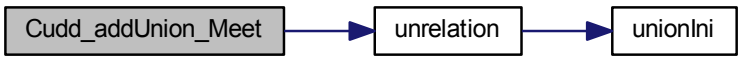
References bck, and unrelation().

Referenced by RelMDDUnionOperation().

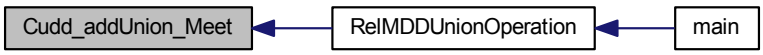
```
{
    DdNode *res;
    DdNode *F, *G;
    CUDD_VALUE_TYPE value=0;
    F = *f;
    G = *g;
```

```
    if ((F == bck) || (G == bck)) {
        return (bck);
    } else {
        if (cuddIsConstant(F) && cuddIsConstant(G)) {
            value = unrelation(cuddV(F), cuddV(G));
            res = cuddUniqueConst(dd, value);
            return (res);
        }
    }
    if (F > G) {                /* swap f and g */
        *f = G;
        *g = F;
    }
    return (NULL);

}                               /* end of Cudd_addPlus */
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.2.11 CUDD_VALUE_TYPE identityRelation (CUDD_VALUE_TYPE *F*)

function:CUDD_VALUE_TYPE **identityRelation(CUDD_VALUE_TYPE F)** (p. **??**) creates the indentity relation

Definition at line 1133 of file RelMDD.c.

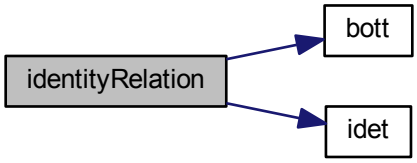
References bott(), holdId, id_col, id_id_track, id_row, idet(), idSource, idTarget, inifor_id, identity::len, bottom::len, identity::rel, and bottom::rel.

Referenced by RelMDDIdentityRelation().

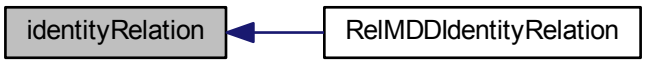
```
{

    int c;
    char *source_Object1[id_row];
    char *target_Object2[id_col];

    for (c = 0; c < id_row; c++) {
        source_Object1[c] = idSource[c];
    }
    for (c = 0; c < id_col; c++) {
        target_Object2[c] = idTarget[c];
    }
    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;
    CUDD_VALUE_TYPE value=0;
    int ste[listob1][listob2];
    identityPtr ret;
```

```
    ret = (identityPtr) idet();
    bottomPtr het;
    het = (bottomPtr) bott();
    int size_bot = het->len;
    int size_struct = ret->len;
    int j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            ste[i][k] = yy;
            yy++;
            //printf("%d",ste[i][k]);
        }
    }

    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            //  if(ste[i][k]==id_id_trackt[id_id_track]){
            if (ste[i][k] == inifor_id[id_id_track]) {
                //printf("id_id_track%d%d\n", id_id_track,
                     //  inifor_id[id_id_track]);
                //printf("\n%d\t%d\n", i, k);
                rn = i;
                cn = k;          //
            }

        }
    }

    if (rn == cn) {
        for (j = 0; j < size_struct; j++) {

            if (!(strcmp(source_Object1[cn], *ret[j].object))) {
                //printf("id \t%s \t %f\n ", *ret[j].object, ret->rel[j]);
                 holdId[rn][rn] = ret->rel[j];
            }

        }
    }

    else {
        for (j = 0; j < size_bot; j++) {
            if (!(strcmp(source_Object1[rn], *het[j].source))
                && !(strcmp(target_Object2[cn], *het[j].target))) {
                //printf(".........%s\t%s\n", *het[j].source,
                    //   *het[j].target);
                // rel_list1[l] = (double)cur->rel[j];
                //perror("gome");
                //printf(".........%f\t\n", het->rel[j]);
                holdId[cn][rn] = het->rel[j];   //check the assignment for cn
       and rn
                // l++;

            }
        }
    }
    id_id_track--;

    return value;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.2.12 identityPtr idet ( )

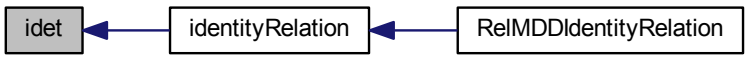
xmlDocPtr doc;

Definition at line 1179 of file RelMDDXmlParser.c.

Referenced by identityRelation().

```
{

    //doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    identityPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadIdentity(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.....................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the caller graph for this function:



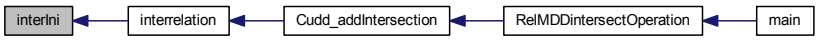
4.2.2.13 intersectionPtr interIni ( )

Definition at line 1115 of file RelMDDXmlParser.c.

Referenced by interrelation().

```
{

  //  xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL;
    root = xmlDocGetRootElement(xmlDocument);

    intersectionPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadIntersection(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}
    if (!root || !root->name || xmlStrcmp(root->name, (xmlChar *)"RelationBasis
      ")) {
        xmlFreeDoc(xmlDocument);
        return NULL;
    }

    return ret;
}
```

Here is the caller graph for this function:



4.2.2.14 double** multiplication (int *row*, int *col*)

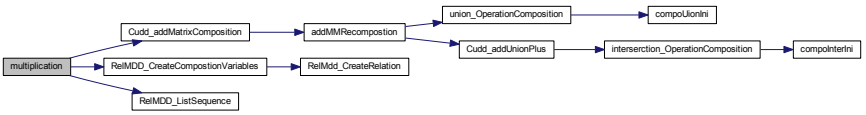
Definition at line 1590 of file RelMDD.c.

References cno_vars, col_trackComp, Cudd_addMatrixComposition(), dd, element_comp, RelMDD_CreateCompostionVariables(), RelMDD_ListSequence(), row_trackComp, summation_var, tempNode, and value_comp.

Referenced by RelMDDCompositionOperation().

```
                                                {
DdNode *M1;
DdNode *M2;
DdNode *nor;
M1= RelMDD_CreateCompostionVariables(dd,row,col,1,0);
DdNode **yy= summation_var;
M2= RelMDD_CreateCompostionVariables(dd,row,col,0,3);
nor=Cudd_addMatrixComposition(dd,M1,M2,yy,cno_vars);
//Cudd_PrintDebug(dd,nor,10,40);
int *list = RelMDD_ListSequence(nor,row, col);
tempNode =nor;
int i,r,c; double rel;

for(i=1;i<row*col;i++){

r=row_trackComp[list[i]];
c= col_trackComp[list[i]];
rel= element_comp[list[i]];
value_comp[r][c]=rel;
printf("<<%f>>",value_comp[r][c]);

}

free(row_trackComp);
free(col_trackComp);
free(element_comp);
return value_comp;

}
```

Here is the call graph for this function:



Here is the caller graph for this function:

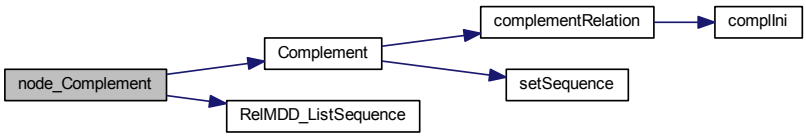
4.2.2.15 DdNode* node_Complement (DdNode * *mat*)

Definition at line 330 of file RelMDD.c.

References Complement(), dd, iniforcompl, mcol, mrow, and RelMDD_ListSequence().

```
{

    iniforcompl = RelMDD_ListSequence(mat, mrow, mcol);
    DdNode *tmp1 = Cudd_addMonadicApply(dd, Complement, mat);
    free(iniforcompl);          // check this if memory link
    return tmp1;
}
```

Here is the call graph for this function:



4.2.2.16 void poo ( )

4.2.2.17 DdNode* RelMDD_CreateCompostionVariables (DdManager * *dd,* int *roww,* int *collh,* int *r,* int *c*)

Definition at line 1530 of file RelMDD.c.

References cno_vars, RelMdd_CreateRelation(), and summation_var.

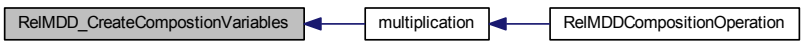
Referenced by multiplication().

```
{
    DdNode *E;
    DdNode **x;
    DdNode **y;
    DdNode **y1;
    DdNode **xn;
    DdNode **xn1;
    DdNode **yn_;
    DdNode **yn_1;
    int nx;
    int ny;
    int maxnx;
    int maxny;
    int m;
    int n;
    int ll;
    int f, j, ww = 0;
    // dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
    int rows[roww * collh];     //malloc
    int cols[roww * collh];     //malloc
    double id_relation[roww * collh];
     double count = 1;
    //int *colum = coll_id( roww);
    //int *rows=  rol_id(coll);
    //int uu [] ={0,0,1,1,2,2};// write methods for this
    //int vv [] ={0,1,0,1,0,1};// write methods for this
    for (f = 0; f < roww; f++) {
        for (j = 0; j < collh; j++) {
            //printf("\n%d%d\n",f,j);
            rows[ww] = f;
            cols[ww] = j;
            //printf("\n%d%d\n",rows[ww],cols[ww]);
            ww++;
        }

    }
    for (ll = 0; ll < roww * collh; ll++) {
        id_relation[ll] = count;
        count = count + 1.00;

    }
    x = y = xn = yn_ = y1 = xn1 = yn_1=  NULL;
    maxnx = maxny = 0;
    nx = maxnx;
    ny = maxny;
   RelMdd_CreateRelation(dd, &E, &x, &y, &xn, &yn_, &nx, &ny, &m, &n, r, roww,
      c, collh, roww, collh, rows, cols, id_relation);
   summation_var = y;
   // Cudd_RecursiveDeref(dd, E);
 // ok = RelMdd_CreateRelation(dd, &F, &y, &y1, &xn1, &yn_1, &nx1, &ny1, &m1,
       &n1, 0, 4, 3, 4, roww, collh, rows, cols, cant);
   //Cudd_PrintDebug(dd,E,nx + ny,40);
   //Cudd_PrintDebug(dd,F,nx1 + ny1,40);
```

```
    //
  //  if(ok)
    cno_vars = nx;
    // node_Complement( E);
    return E;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.2.2.18 int RelMdd_CreateRelation (DdManager * *dd*, DdNode ** *E*, DdNode *** *x*, DdNode *** *y*, DdNode *** *xn*, DdNode *** *yn_*, int * *nx*, int * *ny*, int * *m*, int * *n*, int *bx*, int *sx*, int *by*, int *sy*, int *row_nn*, int *col_nn*, int * *row_n*, int * *col_n*, double * *id_relation*)

CFile

Definition at line 27 of file RelMMDRelations.c.

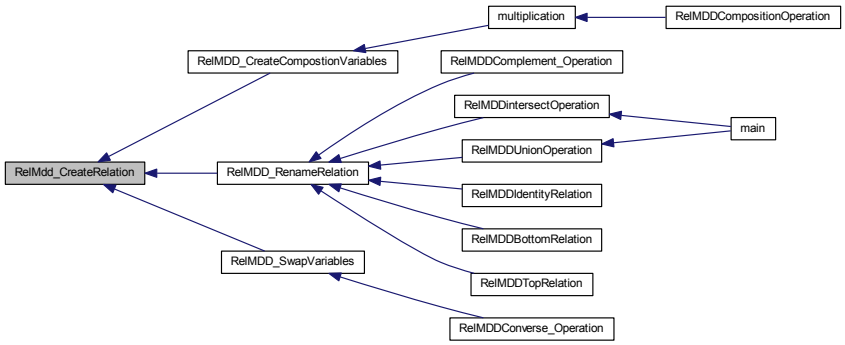
Referenced by RelMDD_CreateCompostionVariables(), RelMDD_RenameRelation(), and RelMDD_SwapVariables().

```
{

    // dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
    DdNode *one, *zero;
    DdNode *w, *neW;
    DdNode *minterm1;
    int u, v, i, nv;
    int lnx, lny;
    //CUDD_VALUE_TYPE val;
    // CUDD_VALUE_TYPE *val1;
    //val1 =(double *) malloc(sizeof(double));
    // val1 = RelMDD_Identity();
    //  save;

    // CUDD_VALUE_TYPE val1 []={1,0,0, 0,0,2,0, 0,0,0,3, 0,0,0,4};
    //CUDD_VALUE_TYPE *val1=RelMDD_Identity();
    CUDD_VALUE_TYPE *val1 = id_relation;
    int kkk=0;

    // int uu [] ={0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4};
    int *uu;
    uu = row_n;
    // int vv [] ={0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4};
    int *vv;
    vv = col_n;
    int pp = 0;
    int ppp = 0;
    DdNode **lx, **ly, **lxn, **lyn;

    one = DD_ONE(dd);
```

```
zero = DD_ZERO(dd);

/* err = fscanf(fp, "%d %d", &u, &v);
   if (err == EOF) {
   return(0);
   } else if (err != 2) {
   return(0);
   } */
u = row_nn;
v = col_nn;
*m = u;
/* Compute the number of x variables. */
lx = *x;
lxn = *xn;
u--;                          /* row and column numbers start from 0 */
for (lnx = 0; u > 0; lnx++) {
    u >>= 1;
}
/* Here we rely on the fact that REALLOC of a null pointer is
 ** translates to an ALLOC.
 */
if (lnx > *nx) {
    *x = lx = REALLOC(DdNode *, *x, lnx);
    if (lx == NULL) {
        dd->errorCode = CUDD_MEMORY_OUT;
        return (0);
    }
    *xn = lxn = REALLOC(DdNode *, *xn, lnx);
    if (lxn == NULL) {
        dd->errorCode = CUDD_MEMORY_OUT;
        return (0);
    }
}

*n = v;
/* Compute the number of y variables. */
ly = *y;
lyn = *yn_;
v--;                          /* row and column numbers start from 0 */
for (lny = 0; v > 0; lny++) {
    v >>= 1;
}
/* Here we rely on the fact that REALLOC of a null pointer is
 ** translates to an ALLOC.
 */
if (lny > *ny) {
    *y = ly = REALLOC(DdNode *, *y, lny);
    if (ly == NULL) {
        dd->errorCode = CUDD_MEMORY_OUT;
        return (0);
    }
    *yn_ = lyn = REALLOC(DdNode *, *yn_, lny);
    if (lyn == NULL) {
        dd->errorCode = CUDD_MEMORY_OUT;
        return (0);
    }
}

/* Create all new variables. */
for (i = *nx, nv = bx + (*nx) * sx; i < lnx; i++, nv += sx) {
    do {
        dd->reordered = 0;
        lx[i] = cuddUniqueInter(dd, nv, one, zero);
    } while (dd->reordered == 1);
    if (lx[i] == NULL)
        return (0);
    cuddRef(lx[i]);
    do {
        dd->reordered = 0;
        lxn[i] = cuddUniqueInter(dd, nv, zero, one);
    } while (dd->reordered == 1);
    if (lxn[i] == NULL)
        return (0);
    cuddRef(lxn[i]);
}
for (i = *ny, nv = by + (*ny) * sy; i < lny; i++, nv += sy) {
    do {
        dd->reordered = 0;
        ly[i] = cuddUniqueInter(dd, nv, one, zero);
    } while (dd->reordered == 1);
    if (ly[i] == NULL)
        return (0);
    cuddRef(ly[i]);
    do {
        dd->reordered = 0;
        lyn[i] = cuddUniqueInter(dd, nv, zero, one);
    } while (dd->reordered == 1);
```

```
        if (lyn[i] == NULL)
            return (0);
        cuddRef(lyn[i]);
    }
    *nx = lnx;
    *ny = lny;

    *E = dd->background;        /* this call will never cause reordering */
    cuddRef(*E);

    while (val1) {
        /* = fscanf(fp, "%d %d %lf", &u, &v, &val);
           if (err == EOF) {
           break;
           } else if (err != 3) {
           return(0);
           } */
        if (uu[pp] >= *m || vv[ppp] >= *n || uu[pp] < 0 || vv[ppp] < 0) {
            return (0);
        }

        minterm1 = one;
        cuddRef(minterm1);

        /* Build minterm1 corresponding to this arc */
        for (i = lnx - 1; i >= 0; i--) {
            if (uu[pp] & 1) {
                w = Cudd_addApply(dd, Cudd_addTimes, minterm1, lx[i]);
            } else {
                w = Cudd_addApply(dd, Cudd_addTimes, minterm1, lxn[i]);
            }
            if (w == NULL) {
                Cudd_RecursiveDeref(dd, minterm1);
                return (0);
            }
            cuddRef(w);
            Cudd_RecursiveDeref(dd, minterm1);
            minterm1 = w;
            uu[pp] >>= 1;
        }
        for (i = lny - 1; i >= 0; i--) {
            if (vv[ppp] & 1) {
                w = Cudd_addApply(dd, Cudd_addTimes, minterm1, ly[i]);
            } else {
                w = Cudd_addApply(dd, Cudd_addTimes, minterm1, lyn[i]);
            }
            if (w == NULL) {
                Cudd_RecursiveDeref(dd, minterm1);
                return (0);
            }
            cuddRef(w);
            Cudd_RecursiveDeref(dd, minterm1);
            minterm1 = w;
            vv[ppp] >>= 1;
        }
        /* Create new constant node if necessary.
         ** This call will never cause reordering.
         */
        neW = cuddUniqueConst(dd, val1[kkk]);
        if (neW == NULL) {
            Cudd_RecursiveDeref(dd, minterm1);
            return (0);
        }
        cuddRef(neW);

        w = Cudd_addIte(dd, minterm1, neW, *E);
        if (w == NULL) {
            Cudd_RecursiveDeref(dd, minterm1);
            Cudd_RecursiveDeref(dd, neW);
            return (0);
        }
        cuddRef(w);
        Cudd_RecursiveDeref(dd, minterm1);
        Cudd_RecursiveDeref(dd, neW);
        Cudd_RecursiveDeref(dd, *E);
        *E = w;
        kkk++;
        pp++;
        ppp++;
    }
    return (1);

}
```

Here is the caller graph for this function:



4.2.2.19 void RelMDD_Int ( )

Function: void **RelMDD_int()** (p. **??**); To be used for the initialization of RelMDD by initialization the the cudd package and setting the background Value to PlusInfinity. To use RelMDD you must call this functions to initialize it before any other functions could be called. If this function is not called you will get error like segmentation fualt.

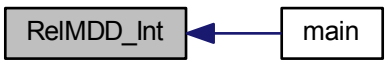
Definition at line 111 of file RelMDD.c.

References bck, and dd.

Referenced by main().

```
{
    dd = Cudd_Init(0, 0, CUDD_UNIQUE_SLOTS, CUDD_CACHE_SLOTS, 0);
    Cudd_AutodynDisable(dd);
    bck = Cudd_ReadPlusInfinity(dd);
    Cudd_SetBackground(dd, bck);
    //RelMDD_intt();
}/*End of void RelMDD_int()*/
```

Here is the caller graph for this function:



4.2.2.20 int* RelMDD_ListSequence (DdNode * *E,* int *roww,* int *coll*)

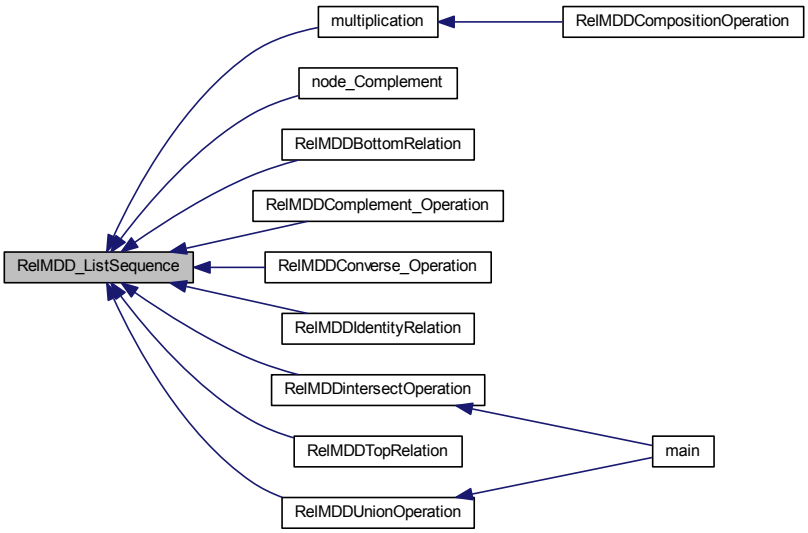
Definition at line 995 of file RelMDD.c.

References dd, ini, no_vars, and t.

Referenced by multiplication(), node_Complement(), RelMDDBottomRelation(), RelMDDComplement_Operation(), RelMDDConverse_Operation(), RelMDDIdentityRelation(), RelMDDintersectOperation(), RelMDDTopRelation(), and RelMDDUnionOperation().

```
{
    // dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
    ini = (int *) malloc(sizeof(int) * roww * coll);    // check here for
       memory leak
    int *cube;
    int t = 0;
    CUDD_VALUE_TYPE value;
    DdGen *gen;
    int q;
    //printf("Testing iterator on cubes:\n");
    Cudd_ForeachCube(dd, E, gen, cube, value) {
        for (q = 0; q < no_vars; q++) {
        }
        //(void) printf(" %g\t",value);
        // trackt[t] = value;
        ini[t] = value;
        t++;
    }
    return ini;
}
```

Here is the caller graph for this function:



4.2.2.21 void RelMDD_Quit ( )

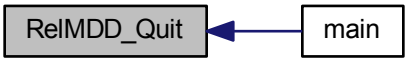
Definition at line 120 of file RelMDD.c.

References dd.

Referenced by main().

```
                {
Cudd_Quit(dd);
}
```

Here is the caller graph for this function:



4.2.2.22 fileInforPtr RelMdd_ReadFile (char * *filename*)

CFile

RelMDD has a special format to read in matrices from files. This format adhere to the format of Harwell-Boeing benchmark suite. This format specifies how matrices should be stored and read from a file. In our case the item on the file specifies the number of rows and column of the matrix. The second and third items on the file specify the list of source and target objects respectively. This is followed by the matrix in which the row and column indices of each element of the matrix is also specified. Refer to the appendix for sample file.

Definition at line 22 of file RelMDDReadRelation.c.

References fileInfor::c, fileInfor::r, fileInfor::rel, fileInfor::rowid, fileInfor::rowida, fileInfor::sourcelist, and fileInfor::targetlist.

Referenced by main().

```
{
    FILE *ifp;

    int u, v, err, i;


    double val;

    ifp = fopen(filename, "r");
    if (ifp == NULL) {
        fprintf(stderr, "Can't open input file in.list!\n");
        exit(1);
    }

    err = fscanf(ifp, "%d %d", &u, &v);
    //printf("\n%d \t %d\n", u,  v);
    fileInforPtr ret;
    ret = (fileInforPtr) malloc(sizeof(fileInfor) * u * v);

    ret->r = u;
    ret->c = v;
    // printf("\n%d \t %d\n", ret->r,   ret->c);
    if (err == EOF) {
        fprintf(stderr, "Can't open input file in.list!\n");
        exit(1);
    } else if (err != 2) {
        fprintf(stderr, "Can't open input file in.list!\n");
        exit(1);
    }

    char *rowlist = NULL;
    rowlist = (char *) malloc(u * 3);

    err = fscanf(ifp, "%s", rowlist);

    if (err == EOF) {
        return (0);
    } else if (err != 1) {

        return (0);
    }

    char *collist = NULL;
    collist = (char *) malloc(v * 3);
```

```
    err = fscanf(ifp, "%s", collist);

    if (err == EOF) {
        return (0);
    } else if (err != 1) {

        return (0);
    }

    ret->sourcelist = (char *) malloc(sizeof(char));

    char *pch;
    int l = 0;

    pch = strtok(rowlist, "%[^]% [],.-");
    while (pch != NULL) {
        //  printf ("%s\n",pch);
        ret[l].sourcelist = pch;
        l++;
        pch = strtok(NULL, "[] ,%[^]%n.-");
    }


    ret->targetlist = (char *) malloc(sizeof(char));

    char *pchs;
    int ll = 0;

    pchs = strtok(collist, " [],.-");
    while (pchs != NULL) {
        ret[ll].targetlist = pchs;
        ll++;
        pchs = strtok(NULL, "[] ,.-");
    }

    ret->rel = (double *) malloc(sizeof(double) * u * v);

    ret->rowid = (int *) malloc(sizeof(int) * u * v);

    ret->rowida = (int *) malloc(sizeof(int) * u * v);

    i = 0;
    while (!feof(ifp)) {

        err = fscanf(ifp, "%d %d %lf", &u, &v, &val);
        // printf("\n%d \t %d \t%f\n ", u,  v, val);
        //printf("%d",err);
        if (err == EOF) {
            break;
        }
        //else if () {
        //   break;
        else if (err != 3) {

            break;
        }
        //else if (u >= m || v >= n || u < 0 || v < 0) {
        //   break;

        //}
        ret->rowid[i] = u;      //printf("%d\t ", ret->rowid[i]);
        ret->rowida[i] = v;     // printf("%d\t ", ret->rowida[i]);
        //ret->colid[i]=u; printf("%d\t ", ret->colid[i]);
        ret->rel[i] = val;      //printf("%f\n ", ret->rel[i]);
        i++;
    }

    fclose(ifp);

    return ret;
}
```

Here is the caller graph for this function:

4.2.2.23 DdNode* RelMDD_RenameRelation (DdManager * *dd,* int *roww,* int *collh*)

function:DdNode *DdNode *RelMDD_RenameRelation(DdManager * dd, int roww, int collh) This function renames relation since for instance 2 with objects A to B is not the same as 2 with object B to A Internal

Definition at line 942 of file RelMDD.c.

References no_vars, and RelMdd_CreateRelation().

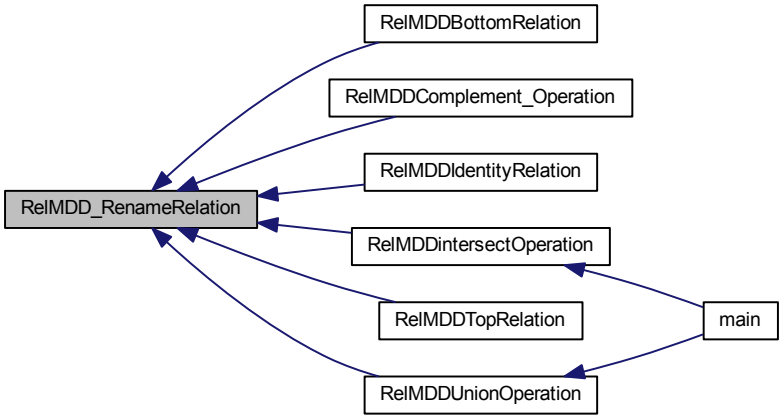
Referenced by RelMDDBottomRelation(), RelMDDComplement_Operation(), RelMDDIdentityRelation(), RelMDDintersectOperation(), RelMDDTopRelation(), and RelMDDUnionOperation().

```
{
    DdNode *E;
    DdNode **x;
    DdNode **y;
    DdNode **xn;
    DdNode **yn_;
    int nx;
    int ny;
    int maxnx;
    int maxny;
    int m;
    int n;
    int ok;
    int ll;
    int f, j, ww = 0;

    // dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
    int rows[roww * collh];     //malloc
    int cols[roww * collh];     //malloc
    double id_relation[roww * collh];
     double count = 1;
    //int *colum = coll_id( roww);
    //int *rows=  rol_id(coll);
    //int uu [] ={0,0,1,1,2,2};// write methods for this
    //int vv [] ={0,1,0,1,0,1};// write methods for this
    for (f = 0; f < roww; f++) {
        for (j = 0; j < collh; j++) {
            //printf("\n%d%d\n",f,j);
            rows[ww] = f;
            cols[ww] = j;
            //printf("\n%d%d\n",rows[ww],cols[ww]);
            ww++;
        }

    }
    for (ll = 0; ll < roww * collh; ll++) {
        id_relation[ll] = count;        //printf(" %f",id_relation[ll]);
        count = count + 1.00;

    }
    x = y = xn = yn_ = NULL;
    maxnx = maxny = 0;
    nx = maxnx;
    ny = maxny;
    ok = RelMdd_CreateRelation(dd, &E, &x, &y, &xn, &yn_, &nx, &ny, &m, &n, 0,
      2, 1, 2, roww, collh, rows, cols, id_relation);  //read matrix 1
     // Cudd_PrintDebug(dd,E,nx + ny,40); //Cudd_RecursiveDeref(dd, E);
     if(ok)
    no_vars = nx;
    // node_Complement( E);
    return E;
```

```
}
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.2.2.24 DdNode* RelMDD_SwapVariables (DdManager * *dd,* int *roww,* int *collh*)

Definition at line 1016 of file RelMDD.c.

References no_vars, and RelMdd_CreateRelation().

Referenced by RelMDDConverse_Operation().

```
{
    DdNode *E;
    DdNode **x;
    DdNode **y;
    DdNode **xn;
    DdNode **yn_;
    int nx;
    int ny;
    int maxnx;
    int maxny;
    int m;
    int n;
    int ll;
    int f, j, ww = 0;
    int cont=0;
    // dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
    int rows[roww * collh];     //malloc
    int cols[roww * collh];     //malloc
    double id_relation[roww * collh];
    int lll = 1;
```

```
    for (f = 0; f < roww; f++) {
        for (j = 0; j < collh; j++) {
            //printf("\n%d%d\n",f,j);
            rows[ww] = f;
            cols[ww] = j;
            //printf("\n%d%d\n",rows[ww],cols[ww]);
            ww++;
        }

    }
    for (ll = 0; ll < roww * collh; ll++) {
        id_relation[ll] = lll;  //printf(" %f",id_relation[ll]);
        lll = lll + 1;

    }
    x = y = xn = yn_ = NULL;
    maxnx = maxny = 0;
    nx = maxnx;
    ny = maxny;
   RelMdd_CreateRelation(dd, &E, &x, &y, &xn, &yn_, &nx, &ny, &m, &n, 0, 2, 1,
      2, roww, collh, rows, cols, id_relation);        //read matrix 1
     Cudd_PrintDebug(dd,E,nx + ny,40);
     //Cudd_RecursiveDeref(dd, E);
   // if(ok)

    if ( nx>ny){
    cont=ny;
    no_vars = ny;
    }

    else {
    cont = nx;
    no_vars = nx;
    }

    DdNode *P = Cudd_addSwapVariables(dd, E, x, y, cont);
    Cudd_PrintDebug(dd,P,nx + ny,40);
    return P;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.2.2.25 RelMDDBottomPtr RelMDDBottomRelation (int *row,* int *col,* char ** *source,* char ** *target*)

functionRelMDDBottomPtr RelMDDBottomRelation(int row, int col, char **source, char **target) Initializes the variables for the for creating the Bottom relation. This is an external functions. It calls other functions and uses them to create the bottom relation based on the object provided. It returns the list of relations, the objectd and the other

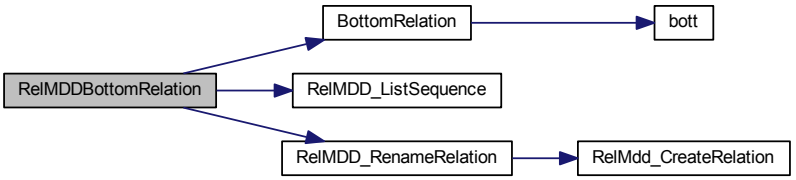
fields defined in the structure RelMDDBottomPtr: row is the size of row of the Matrix col is the size of column of the Matrix followed by list of source and target objects respectively

Definition at line 1225 of file RelMDD.c.

References Bottom_col, Bottom_row, Bottom_track, BottomRelation(), BottomSource, BottomTarget, dd, holdBottom, inifor_Bottom, RelMDDBottom::node, RelMDDBottom::relation, RelMDD_ListSequence(), RelMDD_RenameRelation(), RelMDDBottom::source_Objects, and RelMDDBottom::target_Objects.

```
{
    int move;
    holdBottom = (double **) calloc(row*col*2, sizeof(double*));
    for(move=0;move<row*col*2;move++){
    holdBottom[move] =(double *) calloc(row*col*2, sizeof(double*));
    }
    RelMDDBottomPtr ret;
    ret =(RelMDDBottomPtr) malloc(sizeof(RelMDDBottomPtr) * row*col);
    ret->relation = (double *) malloc(sizeof(double) * row*col*2);
    ret->source_Objects =(char *) malloc(sizeof(char) * row*col*2);
    ret->target_Objects =(char *) malloc(sizeof(char) * row*col*2);
    ret->node = (DdNode *) malloc(sizeof(DdNode) * row*col);
    Bottom_row = row;
    Bottom_col = col;
    Bottom_track = (row * col) - 1;

    BottomSource = (char **) malloc(sizeof(char) *row * col);
    BottomTarget = (char **) malloc(sizeof(char) * row * col);
    for (move = 0; move < Bottom_row; move++) {

        BottomSource[move] = source[move];
        ret[move].source_Objects=source[move];
        //printf("p%s\t%dp\n", BottomSource[move], move);

    }

    for (move = 0; move < Bottom_col; move++) {

        BottomTarget[move] = target[move];
        ret[move].target_Objects=target[move];
        //printf("p%s\t%dp\n", BottomTarget[move], move);

    }
    int i, j,u=0;

    inifor_Bottom =
        RelMDD_ListSequence(RelMDD_RenameRelation(dd, Bottom_row, Bottom_col),
      Bottom_row, Bottom_col);
   // for (i = 0; i < row * col; i++) {
        //printf("%d\t", inifor_Bottom[i]);

   // }
    for (i = 0; i < row * col; i++) {
        BottomRelation(inifor_Bottom[i]);
    }
    ret->node= RelMDD_RenameRelation(dd, Bottom_row, Bottom_col);
    for (j = 0; j < Bottom_row; j++) {
        for (i = 0; i < Bottom_col; i++) {
            ret->relation[u]=holdBottom[i][j];
            u++;
        }
        //printf("\n");
    }
    return ret;
}
```

Here is the call graph for this function:



4.2.2.26 RelMDDComplementPtr RelMDDComplement_Operation (int *com_Row,* int *com_Col,* char ** *com_Rowlist,* char ** *com_Collist,* double * *matrimx*)

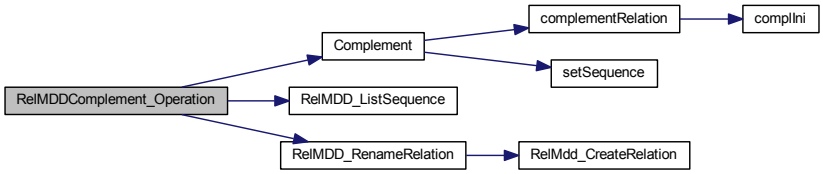
function:RelMDDComplementPtr RelMDDComplement_Operation(int com_Row, int com_Col, char **com_Rowlist, char **com_Collist, double *matrimx) Initializes the variables for the complement operations. This is an external functions. It calls other functions and uses them to compute the complement of a given relation. It returns the list of relations, the source and target objects and the other field defined in the structure RelMDDComplementPtr com_Row is the size of row of the Matrix com_Col is the size of col of the Matrix followed by the source and target object then finally the matrix itself

Definition at line 352 of file RelMDD.c.

References Complement(), ComplementMat2, compleSource, compleTarget, dd, holdcomplement, iniforcompl, mcol, mrow, RelMDDComplement::node, RelMDDComplement::relation, RelMDD_ListSequence(), RelMDD_RenameRelation(), RelMDDComplement::source_Objects, RelMDDComplement::target_Objects, and track.

```
{
    RelMDDComplementPtr ret;
    int move;
    mrow = com_Row;
    mcol = com_Col;
    ComplementMat2 = matrimx;
    track = (com_Row * com_Col) - 1;
    holdcomplement= (double **) calloc(mrow * mcol*2, sizeof(double*));
    for(move=0;move<mrow * mcol;move++){
    holdcomplement[move] =(double *) calloc(mrow * mcol, sizeof(double*));
    }
    ret = (RelMDDComplementPtr) malloc(sizeof(RelMDDComplementPtr) *  mrow *
      mcol*2);
    ret->source_Objects = (char *) malloc(sizeof(char) *  mrow * mcol*2);
    ret->target_Objects = (char *) malloc(sizeof(char)* mrow * mcol*2);
    ret->relation = (double *) malloc(sizeof(double) * mrow * mcol*2);

    compleSource = (char **) malloc(sizeof(char) * com_Row * com_Col*2);
    compleTarget = (char **) malloc(sizeof(char) * com_Row * com_Col*2);

    for (move = 0; move < com_Row; move++) {
        ret[move].source_Objects = com_Rowlist[move];
        //printf("%s\t%d\n", ret[move].source_Objects, move);

        //printf("%f\t%d\n", ComplementMat2[move], move);
        compleSource[move] = com_Rowlist[move];
        //printf("%s\t%d\n", compleSource[move], move);

    }

    for (move = 0; move < com_Col; move++) {

        ret[move].target_Objects = com_Rowlist[move];
        //printf("%s\t%d\n", ret[move].target_Objects, move);

        compleTarget[move] = com_Collist[move];
        //printf("%s\t%d\n", compleTarget[move], move);

    }


    DdNode *M2 = RelMDD_RenameRelation(dd, mrow,mcol);
    iniforcompl = RelMDD_ListSequence(M2,  mrow,   mcol);
    Cudd_addMonadicApply(dd, Complement, M2);
    ret->node=M2;

     int i, j,c =0;
     for(i=0;i<com_Row;i++){

     for(j=0;j<com_Col;j++){

         ret->relation[c]= holdcomplement[i][j];// return the relation here
         //printf("%f",ret->relation[c]);
         c++;
     }
        }
   // ret->relation= holdcomplement;

    return ret;
}/*End of  RelMDDComplementPtr RelMDDComplement_Operation()  */
```

Here is the call graph for this function:



4.2.2.27 RelMDDCompositionPtr RelMDDCompositionOperation (int *no_rowofMat1,* int *no_colsofMat1,* int *no_colsofMat2,* char ∗∗ *source_Object,* char ∗∗ *intermidiate_Object,* char ∗∗ *target_Object2,* double ∗ *matrix1,* double ∗ *matrix2*)

function:RelMDDCompositionPtr **RelMDDCompositionOperation(int no_rowofMat1, int no_colsofMat1, int no_colsofMat2, char ∗∗source_Object, char ∗∗intermidiate_Object, char ∗∗target_Object2, double ∗matrix1, double ∗matrix2)** (p. **??**) Initializes the variables for the compostion operations. This is an external functions. It calls other functions and uses them to compute the join of two given relation or matrices. It returns the list of relations, the source and target objects and the other fields defined in the structure **RelMDDComposition** (p. 13RelMDDComposition Struct Referencesection.3.12): no_rowofMat1 is the size of row of the first Matrix no_colsofMat1, is the size of col of first the Matrix no_colsofMat2, is the size of col of second the Matrix followed by the source and target object first matrix followed by the target object second matrix then finally the first matrix and second matrix respectively

Definition at line 1632 of file RelMDD.c.

References col_comp, col_trackComp, element_comp, mat1, mat2, MAXA, multiplication(), RelMDDComposition::node, RelMDDComposition::relation, row_comp, row_trackComp, source_comp, RelMDDComposition::source_Objects, target_Comp, RelMDDComposition::target_Objects, tempNode, and value_comp.

```
                                                        {
 element_comp = (double *) malloc(sizeof(double) * no_rowofMat1*no_colsofMat1*
      MAXA);
 row_trackComp = (int  *) malloc(sizeof(int) * no_rowofMat1*no_colsofMat1*MAXA)
      ;
col_trackComp = (int *) malloc(sizeof(int) * no_rowofMat1*no_colsofMat1*MAXA);
RelMDDCompositionPtr ret;
    ret =(RelMDDCompositionPtr) malloc(sizeof(RelMDDComposition) * no_rowofMat1
       * no_colsofMat2*2);
    ret->relation = (double *) malloc(sizeof(double) * no_rowofMat1 *
      no_colsofMat2*2);
    ret->source_Objects =(char *) malloc(sizeof(char) * no_rowofMat1 *
      no_colsofMat2*2);
    ret->target_Objects =(char *) malloc(sizeof(char) * no_rowofMat1 *
      no_colsofMat2*2);
    ret->node = (DdNode *) malloc(sizeof(DdNode) * no_rowofMat1 * no_colsofMat2
      *2);
int move;
    value_comp= (double **) calloc(no_rowofMat1*no_colsofMat1+1, sizeof(double
      *));
    for(move=0;move<no_rowofMat1*no_colsofMat1+1;move++){
    value_comp[move] =(double *) calloc(no_rowofMat1*no_colsofMat1+1, sizeof(
      double*));
    }
    for (move = 0; move < no_rowofMat1; move++) {
        ret[move].source_Objects = source_Object[move];
    }
    for (move = 0; move <no_colsofMat2; move++) {
        ret[move].target_Objects =target_Object2[move];
    }
mat1= matrix1;
mat2=matrix2;
source_comp= source_Object;
target_Comp= target_Object2;
row_comp=no_rowofMat1;
col_comp=no_colsofMat2;
double **temp=multiplication(no_rowofMat1 , no_colsofMat2);
```

```
  int i, j, l=0;
 for (j = 0; j < no_rowofMat1; j++) {
        for (i = 0; i < no_colsofMat2; i++) {
          ret->relation[l]=temp[i][j];
          l++;
        }

    }

 ret->node= tempNode;
 return ret;
}
```

Here is the call graph for this function:

4.2.2.28 RelMDDConversePtr RelMDDConverse_Operation (int *tra_rn*, int *tra_col*, char ** *source_object*, char ** *target_object*, double * *matrimx*)

Prototypes decalations of functions in this file. function: **RelMDDConverse_Operation(int tra_rn, int tra_col, char **srelation, char **trelation, double *matrimx)** (p. **??**) Initializes the variables for the converse operations. This is an external functions. It calls other functions and uses them to compute the converse of a given relation. It returns the list of relations, the source and target objects and the other field defined in the structure **RelMDDConverse** (p. 14RelMDDConverse Struct Referencesection.3.13). tran_rn is the size of row of the Matrix tran_col is the size of col of the Matrix followed by the source and target object then finally the matrix itself

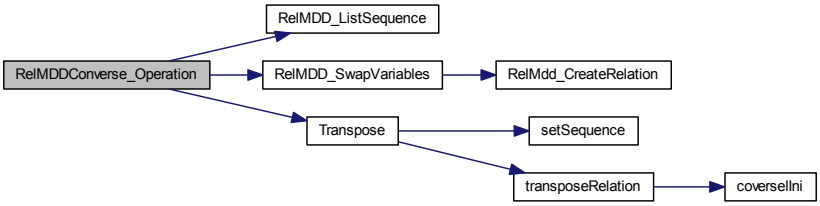
Definition at line 135 of file RelMDD.c.

References dd, holdconverse, iniforTrans, RelMDDConverse::node, RelMDDConverse::relation, RelMDD_ListSequence(), RelMDD_SwapVariables(), RelMDDConverse::source_Objects, RelMDDConverse::target_Objects, track_trans, tran_col, tran_row, tran_Source, tran_Target, TranposeMat, and Transpose().

```
{

    int move;
    int  row = tra_rn;
    int col = tra_col;
    RelMDDConversePtr ret= NULL;
  //  holdconverse = (double *) malloc(sizeof(double) * row *col *2);
    holdconverse= (double **) calloc(row*col*2, sizeof(double*));
    for(move=0;move<row*col*2;move++){
    holdconverse[move] =(double *) calloc(row*col*2, sizeof(double*));
    }
    ret = (RelMDDConversePtr) malloc(sizeof(RelMDDConversePtr)*row *col *2);
    ret->source_Objects= (char*)malloc(sizeof(char)*row *col *2);
    ret->target_Objects= (char*)malloc(sizeof(char)*row *col *2);
    ret->relation= (double*)malloc(sizeof(double)*row *col *2);

    tran_row = tra_rn;
    tran_col = tra_col;
    TranposeMat = matrimx;
    track_trans = (tra_rn * tra_col) - 1;

    tran_Source = (char **) malloc(sizeof(char) * tra_rn*tran_col*2);
    tran_Target = (char **) malloc(sizeof(char) * tra_rn*tran_col*2);
    for (move = 0; move < tra_rn; move++) {
        tran_Source[move] =source_object[move];
        //printf("%s\t%d\n", tran_Source[move], move);
        //ret[move].source_Objects = source_object[move];
        // printf ("%s\t%d\n", tran_Source[move],move);
    }
    for (move = 0; move < tra_col; move++) {

        tran_Target[move] = target_object[move];
        //printf("%s\t%d\n", tran_Target[move], move);

    }
```

```
    DdNode *Matt2;
    DdNode *Matt = RelMDD_SwapVariables(dd, tra_rn, tra_col);// the transpose
       matrix
   //Cudd_PrintDebug(dd,Matt,3,40);
        iniforTrans = RelMDD_ListSequence(Matt, tra_rn, tra_col);

   Matt2= Cudd_addMonadicApply(dd, Transpose, Matt); //creating the new
       relations from the xml file
    ret->node= Matt2; //return the ADD here


    for (move = 0; move < tran_col; move++) {
        ret[move].source_Objects = target_object[move]; // printf ("%s\t%d\n",
       ret[move].source_Objects,move);

         }
     for (move = 0; move <tran_row; move++) {

        ret[move].target_Objects = source_object[move];  //printf ("%s\t%d\n",
       ret[move].target_Objects,move);
         }
     int i, j,c =0;
     for(i=0;i<tran_col;i++){

     for(j=0;j<tran_row;j++){

         ret->relation[c]= holdconverse[i][j];// return the relation here
        //printf("%f",ret->relation[c]);
         c++;
     }
        }
    return ret ;
}/*End of RelMDDConverse_Operation*/
```

Here is the call graph for this function:



4.2.2.29 RelMDDIdentityPtr RelMDDIdentityRelation (int *row,* int *col,* char ** *object*)

function:**RelMDDIdentityRelation(int row, int col, char **object)** (p. **??**) Initializes the variables for the for creating the identity relation. This is an external functions. It calls other functions and uses them to create the identity relation based on the object provided. It returns the list of relations, the objectd and the other fields defined in the structure RelMDDIdentityPtr: row is the size of row of the Matrix col is the size of column of the Matrix followed by list of object

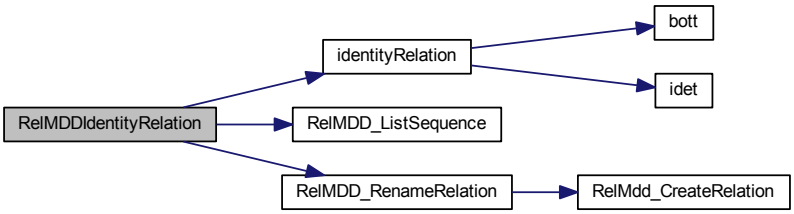
Definition at line 1085 of file RelMDD.c.

References dd, holdId, id_col, id_id_track, id_row, identityRelation(), idSource, idTarget, inifor_id, RelMDDIdentity::node, RelMDDIdentity::object, RelMDDIdentity::relation, RelMDD_ListSequence(), and RelMDD_RenameRelation().

```
{
    int move;
    holdId = (double **) calloc(row*col*2, sizeof(double*));
    for(move=0;move<row*col*2;move++){
    holdId[move] =(double *) calloc(row*col*2, sizeof(double*));
    }
    RelMDDIdentityPtr ret;
    ret =(RelMDDIdentityPtr) malloc(sizeof(RelMDDIdentityPtr) * row*col);
    ret->relation = (double *) malloc(sizeof(double) * row*col*2);
    ret->object =(char *) malloc(sizeof(char) * row*col*2);
    ret->node = (DdNode *) malloc(sizeof(DdNode) * row*col);
```

```
    id_row = row;
    id_col = col;
    id_id_track = (row*col) - 1;
    idSource = (char **) malloc(sizeof(char) * row*col);
    idTarget = (char **) malloc(sizeof(char) * row*col);
    for (move = 0; move<row; move++) {
        idSource[move] = object[move];
        //printf("p%s\t%dp\n", idSource[move], move);
    }
    for (move = 0; move < col; move++) {
        idTarget[move] = object[move];
        ret[move].object = object[move];
        //printf("p%s\t%dp\n", idTarget[move], move);
    }
    int i, j,y=0;

    inifor_id = RelMDD_ListSequence(RelMDD_RenameRelation(dd, id_row, id_col),
      id_row,id_col);
    ret->node=RelMDD_RenameRelation(dd, id_row, id_col);
    for (i = 0; i < row*col; i++) {
        identityRelation(inifor_id[i]);
    }
    for (j = 0; j < row; j++) {
        for (i = 0; i < col; i++) {
            //printf(">>%f>>\t", holdId[i][j]);
            ret->relation[y] =holdId[i][j];
            y++;
        }
        //printf("\n");
    }
    return ret;
}/*End of RelMDDIdentityRelation(int row, int col, char **object)*/
```

Here is the call graph for this function:



4.2.2.30 RelMDDIntersectionPtr RelMDDintersectOperation (int *no_rows,* int *no_cols,* char ** *source_Object,* char ** *target_object,* double * *matrix1,* double * *matrix2*)

function:RelMDDIntersectionPtr RelMDDintersectOperation(int no_rows, int no_cols, char **ssrelation, char **ttrelation, double *matrix1, double *matrix2) Initializes the variables for the intersection operations. This is an external functions. It calls other functions and uses them to compute the meet of two given relation or matrices. It returns the list of relations, the source and target objects and the other fields defined in the structure **RelMDDIntersection** (p. 16RelMDDIntersection Struct Referencesection.3.15): no_rows is the size of row of the Matrix no_cols is the size of col of the Matrix followed by the source and target object then finally the first matrix and second matrix respectively

Definition at line 554 of file RelMDD.c.

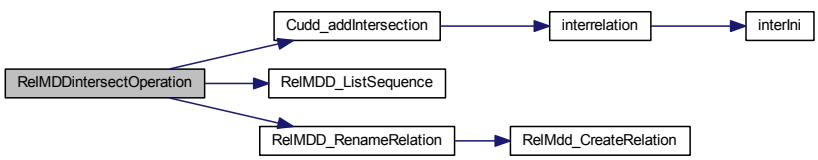
References Cudd_addIntersection(), dd, holdMeet, iniforInters, interMat1, interMat2, interrcol, interrol, interSource, interTarget, RelMDDIntersection::node, RelMDDIntersection::relation, RelMDD_ListSequence(), RelMDD_RenameRelation(), RelMDDIntersection::source_Objects, RelMDDIntersection::target_Objects, and track_Inters.

Referenced by main().

```
{
```

```
    int mrow1= no_rows;
    int mcol1 = no_cols;
    int move;
    holdMeet= (double **) calloc(mrow1 * mcol1*2, sizeof(double*));
    for(move=0;move< mrow1 * mcol1*2;move++){
    holdMeet[move] =(double *) calloc( mrow1 * mcol1*2, sizeof(double*));
    }
    RelMDDIntersectionPtr ret;
    ret =(RelMDDIntersectionPtr) malloc(sizeof(RelMDDIntersection) *mrow1 *
      mcol1*2);
    ret->relation = (double *) malloc(sizeof(double) * mrow1 * mcol1*2);
    ret->source_Objects =(char *) malloc(sizeof(char) * mrow1 * mcol1*2);
    ret->target_Objects =(char *) malloc(sizeof(char) * mrow1 * mcol1*2);
    ret->node = (DdNode *) malloc(sizeof(DdNode) *mrow1 * mcol1*2);
    DdNode *M1;
    DdNode *M2;
    DdNode *tmp;
    int move1;
    track_Inters = (no_cols * no_rows) - 1;
    interMat1 = matrix1;
    interMat2 = matrix2;
    interrol = no_rows;
    interrcol = no_cols;
    interSource = (char **) malloc(sizeof(char) * mrow1 * mcol1*2);
    interTarget = (char **) malloc(sizeof(char) * mrow1 * mcol1*2);
    for (move1 = 0; move1 < mrow1; move1++) {
        interSource[move1] = source_Object[move1];      //
      printf("\n..%s\t%d..\n",interSource[move1],move1);
        ret[move1].source_Objects = source_Object[move1];
    }

      for (move1 = 0; move1 < mcol1; move1++) {
        interTarget[move1] = target_object[move1];      //
      printf("\n..%s\t%d..\n",interTarget[move1],move1);
        ret[move1].target_Objects = target_object[move1];
    }


    // DD mamupulation
    M1 = RelMDD_RenameRelation(dd, no_rows, no_cols);
    Cudd_Ref(M1);
    M2 = RelMDD_RenameRelation(dd, no_rows, no_cols);
    iniforInters = RelMDD_ListSequence(RelMDD_RenameRelation(dd, no_rows,
      no_cols), no_rows, no_cols);
    tmp = Cudd_addApply(dd, Cudd_addIntersection, M1, M2);
    ret->node= tmp;

    int i, j,c =0;
     for(i=0;i<mrow1;i++){

     for(j=0;j<mcol1;j++){

         ret->relation[c]= holdMeet[i][j];// return the relation here
         //printf("%f",holdMeet[i][j]);
         c++;
     }
     //printf("\n");
   }
    return ret;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.2.31 RelMDDTopPtr RelMDDTopRelation (int *row,* int *col,* char ** *source,* char ** *target*)

function: RelMDDBottomPtr RelMDDTopRelation(int row, int col, char **source, char **target) Initializes the variables for the for creating the top relation. This is an external functions. It calls other functions and uses them to create the top relation based on the object provided. It returns the list of relations, the object and the other field members defined in the structure **RelMDDTop** (p. 16RelMDDTop Struct Referencesection.3.16): row is the size of row of the Matrix col is the size of column of the Matrix followed by list of source and target objects respectively

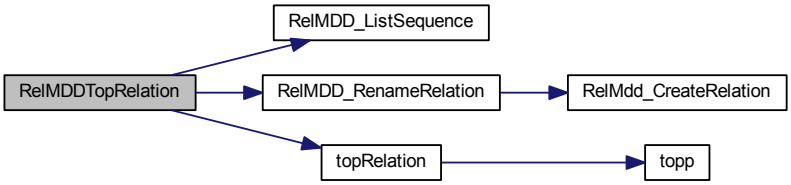
Definition at line 1383 of file RelMDD.c.

References dd, holdTop, inifor_Top, RelMDDTop::node, RelMDDTop::relation, RelMDD_ListSequence(), RelMDD-_RenameRelation(), RelMDDTop::source_Objects, RelMDDTop::target_Objects, Top_col, Top_row, Top_track, top-Relation(), topSource, and topTarget.

```
{
    int move;
    holdTop= (double **) calloc(row*col*2, sizeof(double*));
    for(move=0;move<row*col*2;move++){
    holdTop[move] =(double *) calloc(row*col*2, sizeof(double*));
    }
    RelMDDTopPtr ret;
    ret =(RelMDDTopPtr) malloc(sizeof(RelMDDTopPtr) * row*col);
    ret->relation = (double *) malloc(sizeof(double) * row*col*2);
    ret->source_Objects =(char *) malloc(sizeof(char) * row*col*2);
    ret->target_Objects =(char *) malloc(sizeof(char) * row*col*2);
    ret->node = (DdNode *) malloc(sizeof(DdNode) * row*col);
    Top_row = row;
    Top_col = col;
    Top_track = ( row*col) - 1;
    topSource = (char **) malloc(sizeof(char) *  row*col);
    topTarget = (char **) malloc(sizeof(char) * row*col);
    for (move = 0; move <  row; move++) {

        topSource[move] = source[move];
        ret[move].source_Objects=source[move];
        //printf("p%s\t%dp\n", topSource[move], move);

    }

    for (move = 0; move <  col; move++) {

        topTarget[move] = target[move];
        ret[move].target_Objects=target[move];
        //printf("p%s\t%dp\n", topTarget[move], move);

    }
    int i, j,l=0;

    inifor_Top =
        RelMDD_ListSequence(RelMDD_RenameRelation(dd,Top_row , Top_col ),Top_row
       , Top_col);
    for (i = 0; i < row*col; i++) {
        topRelation(inifor_Top[i]);
    }
     ret->node=RelMDD_RenameRelation(dd,Top_row , Top_col );
    for (j = 0; j < row; j++) {
        for (i = 0; i < col; i++) {
          ret->relation[l]=holdTop[i][j];
          l++;
        }
```

```
    }
    return ret;
}
```

Here is the call graph for this function:



4.2.2.32 RelMDDUnionPtr RelMDDUnionOperation (int *no_rows*, int *no_cols*, char ** *source_Object*, char ** *target_object*, double * *matrix1*, double * *matrix2*)

function:RelMDDUnionPtr RelMDDUnionOperation(int no_rows, int no_cols, char **ssrelation, char **ttrelation, double *matrix1, double *matrix2) Initializes the variables for the join operations. This is an external functions. It calls other functions and uses them to compute the join of two given relation or matrices. It returns the list of relations, the source and target objects and the other fields defined in the structure **RelMDDUnion** (p. 17RelMDD-Union Struct Referencesection.3.17): no_rows is the size of row of the Matrix no_cols is the size of col of the Matrix followed by the source and target object then finally the first matrix and second matrix respectively

Definition at line 744 of file RelMDD.c.

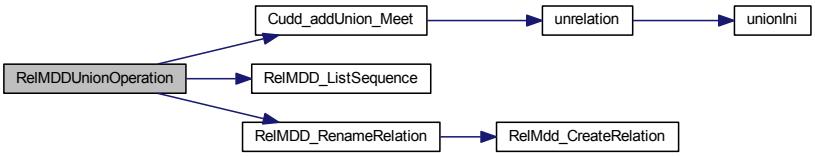
References Cudd_addUnion_Meet(), dd, holdJoin, iniforUnion, RelMDDUnion::node, RelMDDUnion::relation, RelMDD_ListSequence(), RelMDD_RenameRelation(), RelMDDUnion::source_Objects, RelMDDUnion::target_-Objects, track_union, ucol, unionMat1, UnionMat2, unionSource, unionTarget, and urol.

Referenced by main().

```
{
    int unrow = no_rows;
    int uncol = no_cols;
  //  holdJoin = (double *) malloc(sizeof(double) * unrow*uncol*2);
    int move;
    holdJoin= (double **) calloc(unrow * uncol*2, sizeof(double*));
    for(move=0;move< unrow * uncol*2;move++){
    holdJoin[move] =(double *) calloc( unrow * uncol*2, sizeof(double*));
    }
    RelMDDUnionPtr ret;

    ret =
        (RelMDDUnionPtr) malloc(sizeof(RelMDDUnionPtr) * unrow *uncol *2);
    ret->relation = (double *) malloc(sizeof(double) * unrow*uncol*2);
    ret->source_Objects =
        (char *) malloc(sizeof(char) * unrow*uncol);
    ret->target_Objects =
        (char *) malloc(sizeof(char) * unrow*uncol);
    ret->node = (DdNode *) malloc(sizeof(DdNode) *unrow*uncol*2);
    int move1;
    track_union = (unrow*uncol) - 1;
    unionMat1 = matrix1;
    UnionMat2 = matrix2;
    urol = no_rows;
    ucol = no_cols;
   // track_union = (no_rows * no_cols) - 1;
    unionSource = (char **) malloc(sizeof(char) * unrow*uncol);
    unionTarget = (char **) malloc(sizeof(char) * unrow*uncol);
    for (move1 = 0; move1 < unrow; move1++) {
        unionSource[move1] = source_Object[move1];      //
      printf("%s\t%d\n",unionSource[move1],move1);
        ret[move1].source_Objects = source_Object[move1];       // printf
       ("%s\t%d\n", ret[move1].source_Objects,move1);
```

```
    }
    for (move1 = 0; move1 < uncol; move1++) {
        unionTarget[move1] = target_object[move1];      //
      printf("%s\t%d\n",unionTarget[move1],move1);
        ret[move1].target_Objects = target_object[move1];       // printf
       ("%s\t%d\n", ret[move].target_Objects,move);

    }

    DdNode *M1;
    DdNode *M2;
    DdNode *tmp;
    M1 = RelMDD_RenameRelation(dd, unrow, uncol);
    Cudd_Ref(M1);
    //Cudd_RecursiveDeref(manager,f);
    M2 = RelMDD_RenameRelation(dd,  unrow, uncol);
    iniforUnion =
        RelMDD_ListSequence(RelMDD_RenameRelation(dd,  unrow, uncol),
                            unrow, uncol);
    tmp = Cudd_addApply(dd, Cudd_addUnion_Meet, M1, M2);
    ret->node = tmp;

     int i, j,c =0;
     for(i=0;i<unrow;i++){

     for(j=0;j<uncol;j++){

         ret->relation[c]= holdJoin[i][j];// return the relation here
        printf("%f",ret->relation[c]);
         c++;
     }
        }

     if (ret == NULL) {
        printf("\n out of memory\n");
        return NULL;

    }
    return ret;
}/*End RelMDDUnionOperation */
```

Here is the call graph for this function:



Here is the caller graph for this function:

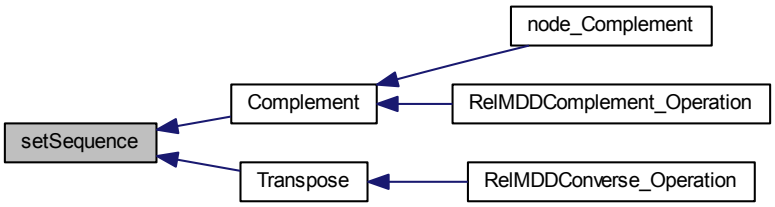
4.2.2.33 double setSequence (double *value*)

Definition at line 535 of file RelMDD.c.

Referenced by Complement(), and Transpose().

```
{

    return value;

}
```

Here is the caller graph for this function:



4.2.2.34 compoInterPtr test ( )

Definition at line 1340 of file RelMDDXmlParser.c.

```
                    {

//xmlDocPtr doc;
  //doc = xmlParseFile("w3.xml");

  if (xmlDocument== NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL ;
  root = xmlDocGetRootElement(xmlDocument);



   compoInterPtr ret;


   // memset(ret, 0, sizeof(identityPtr));
 ret=  relxmlReadCompostion_Inter(  root,xmlDocument);
  int k,kk;
printf("\n.................. in main.................................\n ");

  for(kk=0;kk<4;kk++){

printf("  %s || \t%s \n", *ret[kk].source, *ret[kk].target );
for(k=0;k<20;k++)
printf("  %f\n", ret[kk].content[k] );
}
 if( !root ||
      !root->name ||
      xmlStrcmp(root->name,(xmlChar *)"RelationBasis") )
  {
     xmlFreeDoc(xmlDocument);
     return NULL;
  }

return ret;
 }
```

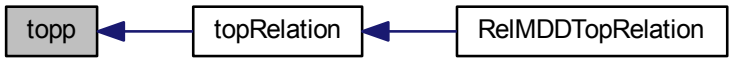
4.2.2.35 topPtr topp ( )

Definition at line 1235 of file RelMDDXmlParser.c.

Referenced by topRelation().

```
{

    //xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");
    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    topPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadTop(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.....................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;

}
```

Here is the caller graph for this function:



4.2.2.36 CUDD_VALUE_TYPE topRelation (CUDD_VALUE_TYPE *F*)

function: CUDD_VALUE_TYPE **topRelation(CUDD_VALUE_TYPE F)** (p. **??**) creates the the top relation relation: returns the a value from the xml file internal file

Definition at line 1442 of file RelMDD.c.

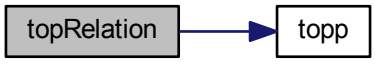
References holdTop, inifor_Top, top::len, top::rel, Top_col, Top_row, Top_track, topp(), topSource, and topTarget.

Referenced by RelMDDTopRelation().

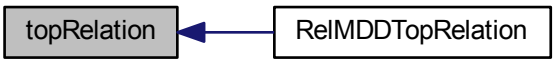
```
{

    int c;
    char *source_Object1[Top_row];
    char *target_Object2[Top_col];

    for (c = 0; c < Top_row; c++) {

        source_Object1[c] = topSource[c];

    }
    for (c = 0; c < Top_col; c++) {

        target_Object2[c] = topTarget[c];

    }

    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;

    CUDD_VALUE_TYPE value=0;
    int ste[listob1][listob2];
    topPtr het;
    het = (topPtr) topp();
```

```
    int size_bot = het->len;
    int  j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            ste[i][k] = yy;
            yy++;
            //printf("%d",ste[i][k]);
        }
    }

    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            //  if(ste[i][k]==Top_Top_trackt[Top_Top_track]){
            if (ste[i][k] == inifor_Top[Top_track]) {
                //printf("Top_Top_track%d%d\n", Top_track,
                     //  inifor_Top[Top_track]);
                //printf("\n%d\t%d\n", i, k);
                rn = i;
                cn = k;          //
            }

        }
    }

    /*    if(rn==cn) {
       for(j = 0; j<size_struct; j++){

       if( !(strcmp(source_Object1[cn], *ret[j].object  ))){

       printf("Top \t%s \t %f\n ",*ret[j].object, ret->rel[j]);
       toppp[rn][rn]=ret->rel[j];
       }

       }}
     */
//                              else{
    for (j = 0; j < size_bot; j++) {
        if (!(strcmp(source_Object1[rn], *het[j].source))
            && !(strcmp(target_Object2[cn], *het[j].target))) {
           // printf(".........%s\t%s\n", *het[j].source, *het[j].target);
            // rel_list1[l] = (double)cur->rel[j];
            //perror("gome");
           // printf(".........%f\t\n", het->rel[j]);
            holdTop[cn][rn] = het->rel[j];      //check the assignment for cn
       and rn
            // l++;

        }
    }
//}
    Top_track--;

    return value;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.2.37 DdNode* Transpose (DdManager * *dd*, DdNode * *f*)

function: *Transpose(DdManager * dd, DdNode * f) It does the ADD manipulation of Coverse operation. It is called by Cudd_addMonadicApply(); to compute the the transpose of a relation. It is an Internal fucntion.

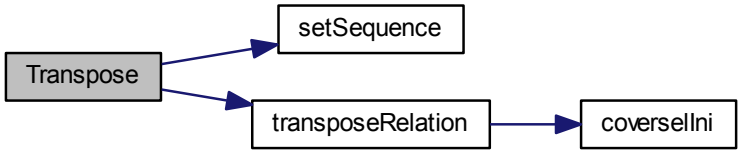
Definition at line 307 of file RelMDD.c.

References bck, setSequence(), and transposeRelation().

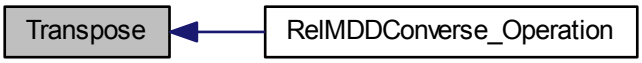
Referenced by RelMDDConverse_Operation().

```
{
    CUDD_VALUE_TYPE value=0;
    DdNode *res;
    if (f == bck) {
        return (bck);
    }

    else {
        if (cuddIsConstant(f) || f == 0) {
           transposeRelation(cuddV(f));
           //printf("<<%f>>", cuddV(f));
            value = setSequence(value);
            res = cuddUniqueConst(dd, value);
            return (res);
        }
    }
    return (NULL);

}/*Tranpose*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.2.38 CUDD_VALUE_TYPE transposeRelation (CUDD_VALUE_TYPE *F*)

function: **transposeRelation(CUDD_VALUE_TYPE F)** (p. **??**) This function does the converse operations by going into the file and selecting the converse of a given element matching it with the source and target object it is called by DdNode ∗Transpose(DdManager ∗ dd, DdNode ∗ f) to perform manipulation of DD. It is an Internal fucntion.

Definition at line 212 of file RelMDD.c.

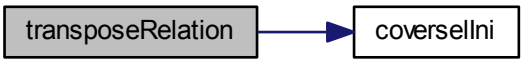
References converse::content, coverselIni(), holdconverse, iniforTrans, converse::len, converse::len_Content, track_trans, tran_col, tran_row, tran_Source, tran_Target, and TranposeMat.

Referenced by Transpose().

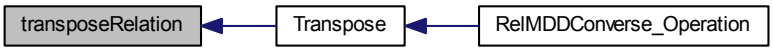
```
{
    int c;

    char *source_Object1[tran_col];//source object
    char *target_Object2[tran_row];// target object
    for (c = 0; c < tran_col; c++) {

        source_Object1[c] = tran_Target[c]; // initialization of source
        //printf("\n++%s++\n ", source_Object1[c]);

    }
    for (c = 0; c < tran_row ; c++) {

        target_Object2[c] = tran_Source[c]; // initialization of target
      // printf("\n++%s++\n ",target_Object2[c] );
    }

//char *source_Object1[]={"A","B","A","A"};
//char *target_Object2[]={"A","A","B","B"};

    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;
    CUDD_VALUE_TYPE value=0;
    int ste[listob2][listob1];
    conversePtr ret;
    ret = (conversePtr) coverselIni();
    int size_struct = ret->len;
    int  l, j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
   for (i = 0; i < listob2; i++) {

        for (k = 0; k < listob1; k++) {

            ste[i][k] = yy;
            yy++;
        // printf("\nN%d%d%dN\n",i,k,ste[i][k]);
        }
    }

    for (i = 0; i < listob2; i++) {

        for (k = 0; k < listob1; k++) {

            //  if(ste[i][k]==trackt[track]){
            if (ste[i][k] == iniforTrans[track_trans]) {
                //printf("track%d\n", track_trans);
                // printf("\n%d\t%d\n", i,k);
```

```
                rn = i;
                cn = k;
                //printf("\n%d\t%d\t%f\n", k,i, F);
                //printf("\n++%s\t%s++\n ", source_Object1[cn],
       target_Object2[rn]);
             }

         }
    }

    for (j = 0; j < size_struct; j++) {

         if (!(strcmp(source_Object1[cn], *ret[j].source))
             && !(strcmp(target_Object2[rn], *ret[j].target))) {
             //printf("\n++%s\t%s++\n ", *ret[j].source, *ret[j].target);
             for (l = 0; l < ret->len_Content[j]; l++) {
                 int ma = (int) (F - 1);
                 if (l % 2 == 0) {
                     if (ret[j].content[l] == TranposeMat[ma]) {
                     // printf("<<%f\t%f>>",F,TranposeMat[ma]);
                         holdconverse[cn][rn]= ret[j].content[l+1];
                         // printf("<<%d\t%f\t>>",ma,holdconverse[cn][rn]);
                         value = F;
                         track_trans--;
                     }
                 }
                 //}

                 //   printf("\n" );
             }

         }
    }

//track_trans--;
    return value;
}/*End of transposeRelation(CUDD_VALUE_TYPE F)*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.2.39 unionsPtr unionlni ( )

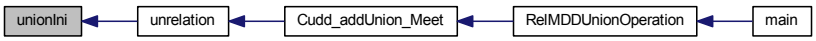
Definition at line 1147 of file RelMDDXmlParser.c.

Referenced by unrelation().

```
{
```

```
   // xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL;
    root = xmlDocGetRootElement(xmlDocument);

    unionsPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadUnion(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}
    if (!root || !root->name || xmlStrcmp(root->name, (xmlChar *)"RelationBasis
      ")) {
        xmlFreeDoc(xmlDocument);
        return NULL;
    }

    return ret;
}
```

Here is the caller graph for this function:



4.2.2.40 CUDD_VALUE_TYPE unrelation (CUDD_VALUE_TYPE *F*, CUDD_VALUE_TYPE *G*)

function: CUDD_VALUE_TYPE **unrelation(CUDD_VALUE_TYPE F, CUDD_VALUE_TYPE G)** (p. **??**) This function does the join operations by going into the file and selecting the join of a given element matching it with the source and target object it is called by DdNode ∗Cudd_addUnion_Meet(DdManager ∗ dd, DdNode ∗∗ f, DdNode ∗∗ g) to perform manipulations of DD. it returns a CUDD_VALUE_TYPE of the relation or index It is an Internal fucntion.

Definition at line 827 of file RelMDD.c.

References unions::content, holdJoin, iniforUnion, unions::len, unions::len_Content, track_union, ucol, unionIni(), unionMat1, UnionMat2, unionSource, unionTarget, and urol.

Referenced by Cudd_addUnion_Meet().

```
{
    int c;
    char *source_Object1[urol];
    char *target_Object2[ucol];
    for (c = 0; c < urol; c++) {

        source_Object1[c] = unionSource[c];

    }
    for (c = 0; c < ucol; c++) {

        target_Object2[c] = unionTarget[c];

    }

    int listob1 = (sizeof(source_Object1) / sizeof(char)) / 4;
    int listob2 = (sizeof(target_Object2) / sizeof(char)) / 4;
    int ste[listob1][listob2];
    CUDD_VALUE_TYPE value=0;
```

```
    unionsPtr ret;
    ret = (unionsPtr)unionIni();
    int size_struct = ret->len;
    int  l, j;
    int rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            ste[i][k] = yy;
            yy++;
            //printf("%d",ste[i][k]);
        }
    }

    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            //  if(ste[i][k]==trackt[track]){
            if (ste[i][k] == iniforUnion[track_union]) {
                //printf("track%d\n", track_union);
                //  printf("\n%d\t%d\n", i,k);
                rn = i;
                cn = k;          //

            }

        }
    }

    for (j = 0; j < size_struct; j++) {

        if (!(strcmp(source_Object1[rn], *ret[j].source))
            && !(strcmp(target_Object2[cn], *ret[j].target))) {
           // printf("\n%s\t%s\n ", *ret[j].source, *ret[j].target);
            // printf("\n%f\t%f\t%f\n" ,F,G, value);
            for (l = 0; l < ret->len_Content[j]; l++) {
                int ma = (int) (F - 1);
                int ka = (int) (G - 1);
                if (l % 3 == 0) {
                    if ((ret[j].content[l] == unionMat1[ma])
                        && (ret[j].content[l + 1] == UnionMat2[ka])) {
                        holdJoin[rn][cn] = ret[j].content[l + 2];
                        value = F;
                        track_union--;

                    }

                }

            }

        }
    }
    return value;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.2.3 Variable Documentation

4.2.3.1 DdNode* bck

Definition at line 24 of file RelMDD.c.

Referenced by Complement(), Cudd_addIntersection(), Cudd_addUnion_Meet(), RelMDD_int(), RelMDD_Int(), and Transpose().

4.2.3.2 unsigned int cacheSize

Definition at line 6 of file RelMDD.h.

4.2.3.3 DdManager* dd

Definition at line 23 of file RelMDD.c.

Referenced by multiplication(), node_Complement(), RelMDD_Int(), RelMDD_int(), RelMDD_ListSequence(), RelMDD_Quit(), RelMDDBottomRelation(), RelMDDComplement_Operation(), RelMDDConverse_Operation(), RelMDDIdentityRelation(), RelMDDintersectOperation(), RelMDDTopRelation(), and RelMDDUnionOperation().

4.2.3.4 unsigned int nslots

Definition at line 7 of file RelMDD.h.

4.2.3.5 unsigned int nvars

Definition at line 7 of file RelMDD.h.

Referenced by Cudd_addMatrixComposition().

4.2.3.6 DdNode* tempNode

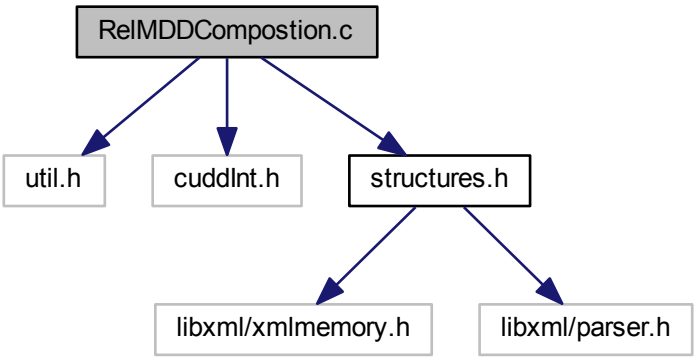
Definition at line 36 of file RelMDD.c.

Referenced by multiplication(), and RelMDDCompositionOperation().

4.3 RelMDDCompostion.c File Reference

```
#include "util.h"
#include "cuddInt.h"
#include "structures.h"
```

Include dependency graph for RelMDDCompostion.c:



Functions

- CUDD_VALUE_TYPE **union_OperationComposition** (CUDD_VALUE_TYPE F, CUDD_VALUE_TYPE G)
- CUDD_VALUE_TYPE **interserction_OperationComposition** (CUDD_VALUE_TYPE F, CUDD_VALUE_TYPE G)
- static DdNode * **addMMRecompostion** (DdManager *__dd__, DdNode *A, DdNode *B, int topP, int *vars)
- DdNode * **Cudd_addUnionPlus** (DdManager *__dd__, DdNode **f, DdNode **g)
- void **inter** (CUDD_VALUE_TYPE F, CUDD_VALUE_TYPE G)
- CUDD_VALUE_TYPE **valF1** ()
- DdNode * **Cudd_addMatrixComposition** (DdManager *__dd__, DdNode *A, DdNode *B, DdNode **z, int nz)

Variables

- static char rcsid[] **DD_UNUSED** = "$Id: cuddMatMult.c,v 1.17 2004/08/13 18:04:50 fabio Exp $"
- double * **mat1**
- double * **mat2**
- char ** **source_comp**
- char ** **target_Comp**
- double * **element_comp**
- int * **row_trackComp**
- int * **col_trackComp**
- static int **t** = 1
- int **row_comp**
- int **col_comp**

4.3.1 Function Documentation

4.3.1.1 static DdNode * addMMRecompostion (DdManager * *dd*, DdNode * *A*, DdNode * *B*, int *topP*, int * *vars*) [static]

Function

Synopsis [Performs the recursive step of Cudd_addMatrixMultiply.]

Description [Performs the recursive step of Cudd_addMatrixComposition. Returns a pointer to the result if successful; NULL otherwise.]

SideEffects [None]

Definition at line 137 of file RelMDDCompostion.c.

References Cudd_addUnionPlus(), t, and union_OperationComposition().

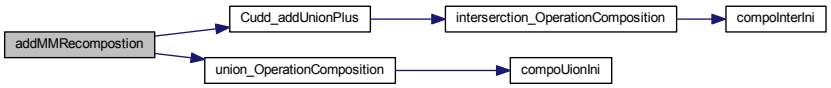
Referenced by Cudd_addMatrixComposition().

```
{
    DdNode *zero, *At,          /* positive cofactor of first operand */
    *Ae,                        /* negative cofactor of first operand */
    *Bt,                        /* positive cofactor of second operand */
    *Be,                        /* negative cofactor of second operand */
    *t,                         /* positive cofactor of result */
    *e,                         /* negative cofactor of result */
    *scaled,                    /* scaled result */
    *add_scale,                 /* ADD representing the scaling factor */
    *res;
    int i;                      /* loop index */
    double scale;               /* scaling factor */
    int index;                  /* index of the top variable */
    CUDD_VALUE_TYPE value;
    unsigned int topA, topB, topV;
    DD_CTFP cacheOp;

    statLine(dd);
    zero = DD_ZERO(dd);

    if (A == zero || B == zero) {
        return (zero);
    }

    if (cuddIsConstant(A) && cuddIsConstant(B)) {
        /* Compute the scaling factor. It is 2^k, where k is the
         ** number of summation variables below the current variable.
         ** Indeed, these constants represent blocks of 2^k identical
         ** constant values in both A and B.
         */
//value = cuddV(A) * cuddV(B); printf("\n%f\t%f\t%f\n", value, cuddV(A)
       ,cuddV(B));
        //inter(cuddV(A) ,cuddV(B));
        value = union_OperationComposition(cuddV(A), cuddV(B));
        //value =vv;
        //vv++;

        for (i = 0; i < dd->size; i++) {
            if (vars[i]) {
                if (dd->perm[i] > topP) {
                    value *= (CUDD_VALUE_TYPE) 2;
                }
            }
        }
        res = cuddUniqueConst(dd, value);
        return (res);
    }

    /* Standardize to increase cache efficiency. Clearly, A*B != B*A
     ** in matrix multiplication. However, which matrix is which is
     ** determined by the variables appearing in the ADDs and not by
     ** which one is passed as first argument.
     */
    if (A > B) {
        DdNode *tmp = A;
        A = B;
        B = tmp;
    }

    topA = cuddI(dd, A->index);
    topB = cuddI(dd, B->index);
    topV = ddMin(topA, topB);

    cacheOp = (DD_CTFP) addMMRecompostion;
    res = cuddCacheLookup2(dd, cacheOp, A, B);
    if (res != NULL) {
        /* If the result is 0, there is no need to normalize.
         ** Otherwise we count the number of z variables between
         ** the current depth and the top of the ADDs. These are
         ** the missing variables that determine the size of the
         ** constant blocks.
         */
        if (res == zero)
            return (res);
```

```
    scale = 1.0;
    for (i = 0; i < dd->size; i++) {
        if (vars[i]) {
            if (dd->perm[i] > topP && (unsigned) dd->perm[i] < topV) {
                scale *= 2;
            }
        }
    }
    if (scale > 1.0) {
        cuddRef(res);
        add_scale = cuddUniqueConst(dd, (CUDD_VALUE_TYPE) scale);
        if (add_scale == NULL) {
            Cudd_RecursiveDeref(dd, res);
            return (NULL);
        }
        cuddRef(add_scale);
        scaled = cuddAddApplyRecur(dd, Cudd_addTimes, res, add_scale);
        if (scaled == NULL) {
            Cudd_RecursiveDeref(dd, add_scale);
            Cudd_RecursiveDeref(dd, res);
            return (NULL);
        }
        cuddRef(scaled);
        Cudd_RecursiveDeref(dd, add_scale);
        Cudd_RecursiveDeref(dd, res);
        res = scaled;
        cuddDeref(res);
    }
    return (res);
}

/* compute the cofactors */
if (topV == topA) {
    At = cuddT(A);
    Ae = cuddE(A);
} else {
    At = Ae = A;
}
if (topV == topB) {
    Bt = cuddT(B);
    Be = cuddE(B);
} else {
    Bt = Be = B;
}

t = addMMRecompostion(dd, At, Bt, (int) topV, vars);
if (t == NULL)
    return (NULL);
cuddRef(t);
e = addMMRecompostion(dd, Ae, Be, (int) topV, vars);
if (e == NULL) {
    Cudd_RecursiveDeref(dd, t);
    return (NULL);
}
cuddRef(e);

index = dd->invperm[topV];
if (vars[index] == 0) {
    /* We have split on either the rows of A or the columns
     ** of B. We just need to connect the two subresults,
     ** which correspond to two submatrices of the result.
     */
    res = (t == e) ? t : cuddUniqueInter(dd, index, t, e);
    if (res == NULL) {
        Cudd_RecursiveDeref(dd, t);
        Cudd_RecursiveDeref(dd, e);
        return (NULL);
    }
    cuddRef(res);
    cuddDeref(t);
    cuddDeref(e);
} else {
    /* we have simultaneously split on the columns of A and
     ** the rows of B. The two subresults must be added.
     */
    res = cuddAddApplyRecur(dd, Cudd_addUnionPlus, t, e);
    if (res == NULL) {
        Cudd_RecursiveDeref(dd, t);
        Cudd_RecursiveDeref(dd, e);
        return (NULL);
    }
    cuddRef(res);
    Cudd_RecursiveDeref(dd, t);
    Cudd_RecursiveDeref(dd, e);
}

cuddCacheInsert2(dd, cacheOp, A, B, res);
```

```
    /* We have computed (and stored in the computed table) a minimal
     ** result; that is, a result that assumes no summation variables
     ** between the current depth of the recursion and its top
     ** variable. We now take into account the z variables by properly
     ** scaling the result.
     */
    if (res != zero) {
        scale = 1.0;
        for (i = 0; i < dd->size; i++) {
            if (vars[i]) {
                if (dd->perm[i] > topP && (unsigned) dd->perm[i] < topV) {
                    scale *= 2;
                }
            }
        }
        if (scale > 1.0) {
            add_scale = cuddUniqueConst(dd, (CUDD_VALUE_TYPE) scale);
            if (add_scale == NULL) {
                Cudd_RecursiveDeref(dd, res);
                return (NULL);
            }
            cuddRef(add_scale);
            scaled = cuddAddApplyRecur(dd, Cudd_addTimes, res, add_scale);
            if (scaled == NULL) {
                Cudd_RecursiveDeref(dd, res);
                Cudd_RecursiveDeref(dd, add_scale);
                return (NULL);
            }
            cuddRef(scaled);
            Cudd_RecursiveDeref(dd, add_scale);
            Cudd_RecursiveDeref(dd, res);
            res = scaled;
        }
    }
    cuddDeref(res);
    return (res);

}                               /* end of addMMRecur */
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.3.1.2 DdNode* Cudd_addMatrixComposition (DdManager * *dd*, DdNode * *A*, DdNode * *B*, DdNode ** *z*, int *nz*)

AutomaticEnd Function

Synopsis [Calculates the product of two matrices represented as ADDs.]

Description [Calculates the product of two matrices, A and B, represented as ADDs. This procedure implements the quasiring multiplication algorithm. A is assumed to depend on variables x (rows) and z (columns). B is assumed to depend on variables z (rows) and y (columns). The product of A and B then depends on x (rows) and y (columns). Only the z variables have to be explicitly identified; they are the "summation" variables. Returns a pointer to the result if successful; NULL otherwise.]

SideEffects [None]

SeeAlso [Cudd_addTimesPlus Cudd_addTriangle Cudd_bddAndAbstract]

This is Funtion is a direct modification of Cudd_addMatrixMultiply. Basically, replacing addition operation with union operation and multiplication with intersection. Copyright goes to Fabio not me please and just using to get my MSc

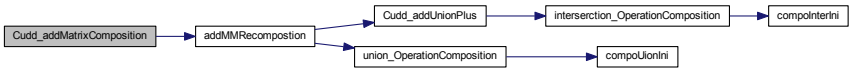
Definition at line 89 of file RelMDDCompostion.c.

References addMMRecompostion(), and nvars.

Referenced by multiplication().

```
{
    int i, nvars, *vars;
    DdNode *res;

    /* Array vars says what variables are "summation" variables. */
    nvars = dd->size;
    vars = ALLOC(int, nvars);
    if (vars == NULL) {
        dd->errorCode = CUDD_MEMORY_OUT;
        return (NULL);
    }
    for (i = 0; i < nvars; i++) {
        vars[i] = 0;
    }
    for (i = 0; i < nz; i++) {
        vars[z[i]->index] = 1;
    }

    do {
        dd->reordered = 0;
        res = addMMRecompostion(dd, A, B, -1, vars);
    } while (dd->reordered == 1);
    FREE(vars);
    return (res);

}                               /* end of Cudd_addMatrixMultiply */
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.3.1.3 DdNode * Cudd_addUnionPlus (DdManager * *dd*, DdNode ** *f*, DdNode ** *g*)

Function**********

The union operation from the xml is done here

Definition at line 354 of file RelMDDCompostion.c.

References interserction_OperationComposition().

Referenced by addMMRecompostion().

```
{
    DdNode *res;
    DdNode *F, *G;
    CUDD_VALUE_TYPE value;

    F = *f;
    G = *g;
    if (F == DD_ZERO(dd))
        return (G);
    if (G == DD_ZERO(dd))
        return (F);
    if (cuddIsConstant(F) && cuddIsConstant(G)) {
        //
       printf("\n.....................................................................................\n");
        //cInterserction(cuddV(F), cuddV(G));
        //value = cuddV(F)+cuddV(G);

        value = interserction_OperationComposition(cuddV(F), cuddV(G));
        //printf("\n+++ %f%f\t%f+++\n", value, cuddV(F), cuddV(G));
        res = cuddUniqueConst(dd, value);
        return (res);
    }
    // printf("\n");
    if (F > G) {                /* swap f and g */
        *f = G;
        *g = F;
    }
    return (NULL);

}
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.3.1.4 void inter (CUDD_VALUE_TYPE *F*, CUDD_VALUE_TYPE *G*)

4.3.1.5 CUDD_VALUE_TYPE interserction_OperationComposition (CUDD_VALUE_TYPE *F*, CUDD_VALUE_TYPE *G*)

Function**********

Mapping the relations to thier objects

Definition at line 490 of file RelMDDCompostion.c.

References col_trackComp, compoInterIni(), compoInter::content, element_comp, compoInter::len, compoInter-::len_Content, row_trackComp, source_comp, t, and target_Comp.

Referenced by Cudd_addUnionPlus().

```
{

    CUDD_VALUE_TYPE value=0;
    compoInterPtr ret;
    ret = (compoInterPtr)compoInterIni();
    int size_struct = ret->len;
```

```
    int j, l, rn, cn;
    rn = row_trackComp[(int) (F)];
    cn = col_trackComp[(int) (F)];

    for (j = 0; j < size_struct; j++) {
        if (!(strcmp(source_comp[rn], *ret[j].source))
            && !(strcmp(target_Comp[cn], *ret[j].target))) {
            //printf("\n%s\t%s\n ", *ret[j].source, *ret[j].target);

            for (l = 0; l < ret->len_Content[j]; l++) {
                int ma = (int) (F);
                int ka = (int) (G);
                if (l % 3 == 0) {
                    if ((ret[j].content[l] ==element_comp[ma])
                        && (ret[j].content[l + 1] == element_comp[ka])) {

                        //printf("\n+++%f\t%f\t%f+++\n", element_comp[ma],
                             //  element_comp[ka], ret[j].content[l + 2]);
                        //    arrya[t]= ret[j].content[l+2];
                        element_comp[t] = ret[j].content[l + 2];
                        //element_comp[t]= ret[j].content[l + 2];
                        // printf("\n%f\n",interarr[ma]);
                        //row[t] = rn;
                        row_trackComp[t]=rn;
                        //col[t] = cn;
                        col_trackComp[t]=cn;
                        value = t;
                        //printf("\nt in un-%d \t%d\t %d t-\n", t,
       row_trackComp[t],
                               // col_trackComp[t]);
                        t++;
                        // t++;
                    }

                }

            }
        }
    }

    return value;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.3.1.6 CUDD_VALUE_TYPE union_OperationComposition (CUDD_VALUE_TYPE *F*, CUDD_VALUE_TYPE *G*)

AutomaticStart

Function

Mapping the relations to thier objects

Definition at line 399 of file RelMDDCompostion.c.

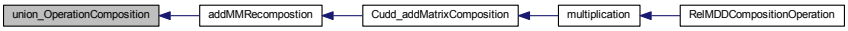
References col_comp, col_trackComp, compoUionIni(), compoUnion::content, element_comp, compoUnion::len, compoUnion::len_Content, mat1, mat2, row_comp, row_trackComp, source_comp, t, and target_Comp.

Referenced by addMMRecompostion().

```
{

    int listob1 = row_comp;
    int listob2 = col_comp;
    CUDD_VALUE_TYPE state[listob1][listob2];    // malloc here
    CUDD_VALUE_TYPE value=0;
    compoUnionPtr ret;
    ret = (compoUnionPtr) compoUionIni();
    int size_struct = ret->len;
    int j, l, rn=0, cn=0;
    static int i, k;
    int yy = 1;
    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            state[i][k] = (CUDD_VALUE_TYPE) yy;
            yy++;
            // printf("%d",state[i][k]);
        }
    }

    for (i = 0; i < listob1; i++) {

        for (k = 0; k < listob2; k++) {

            if (state[i][k] == F) {

                //printf("\n1- %f\t%f\t%f\n", F, G, state[i][k]);
                //  printf("\n%d\t%d\n", i,k);
                rn = i;
            }

            if (state[i][k] == (G)) {   // to G-first of the second matrix

                //printf("\n%f\t%f\t%f\n" ,F,G, state[i][k]);
                //  printf("\n,,%d\t%d,,\n", i,k);
                cn = k;         //
            }
        }
    }

    for (j = 0; j < size_struct; j++) {

        if (!(strcmp(source_comp[rn], *ret[j].source))
            && !(strcmp(target_Comp[cn], *ret[j].target))) {
           // printf("\n2-%s\t%s\n ", *ret[j].source, *ret[j].target);

            for (l = 0; l < ret->len_Content[j]; l++) {
                int ma = (int) (F - 1);
                int ka = (int) (G - 1); // to G-first of the second matrix -1
                if (l % 3 == 0) {
                    if ((ret[j].content[l] == mat1[ma])
                        && (ret[j].content[l + 1] == mat2[ka])) {
                        //printf("\nREL%f\t%f\t%fREL\n", mat1[ma], mat2[ka],
       ret[j].content[l + 2]);    //}} }}}
                        //row[t] = rn;
                        //col[t] = cn;
                        row_trackComp[t]=rn;
                        col_trackComp[t]=cn;
                        //arrya[t] = ret[j].content[l + 2];
                        element_comp[t]= ret[j].content[l + 2];
                        //printf("\n,,%f,,\n",arrya[t]);
                        //value= ret[j].content[l+2];
                        value = (double) (t);
                        //printf("\nt-%d \t%d\t %d t-\n", t, row_trackComp[t],
       col_trackComp[t]);
                        t++;
                    }

                }

        }}
    }

    return value;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.3.1.7 CUDD_VALUE_TYPE valF1 ( )

4.3.2 Variable Documentation

4.3.2.1 int col_comp

Definition at line 58 of file RelMDDCompostion.c.

Referenced by RelMDDCompositionOperation(), and union_OperationComposition().

4.3.2.2 int* col_trackComp

Definition at line 54 of file RelMDDCompostion.c.

Referenced by interserction_OperationComposition(), multiplication(), RelMDDCompositionOperation(), and union-_OperationComposition().

4.3.2.3 char rcsid [ ] DD_UNUSED = "$Id: cuddMatMult.c,v 1.17 2004/08/13 18:04:50 fabio Exp $" [static]

CFile**********

Definition at line 26 of file RelMDDCompostion.c.

4.3.2.4 double* element_comp

Definition at line 52 of file RelMDDCompostion.c.

Referenced by interserction_OperationComposition(), multiplication(), RelMDDCompositionOperation(), and union-_OperationComposition().

4.3.2.5 double* mat1

Definition at line 48 of file RelMDDCompostion.c.

Referenced by RelMDDCompositionOperation(), and union_OperationComposition().

4.3.2.6 double* mat2

Definition at line 49 of file RelMDDCompostion.c.

Referenced by RelMDDCompositionOperation(), and union_OperationComposition().

4.3.2.7 int row_comp

Definition at line 57 of file RelMDDCompostion.c.

Referenced by RelMDDCompositionOperation(), and union_OperationComposition().

4.3.2.8 int* row_trackComp

Definition at line 53 of file RelMDDCompostion.c.

Referenced by interserction_OperationComposition(), multiplication(), RelMDDCompositionOperation(), and union_OperationComposition().

4.3.2.9 char** source_comp

Definition at line 50 of file RelMDDCompostion.c.

Referenced by interserction_OperationComposition(), RelMDDCompositionOperation(), and union_OperationComposition().

4.3.2.10 int t = 1 [static]

Definition at line 56 of file RelMDDCompostion.c.

Referenced by addMMRecompostion(), interserction_OperationComposition(), RelMDD_ListSequence(), and union_OperationComposition().

4.3.2.11 char** target_Comp

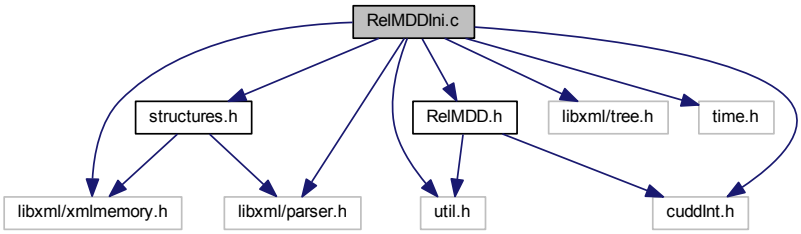
Definition at line 51 of file RelMDDCompostion.c.

Referenced by interserction_OperationComposition(), RelMDDCompositionOperation(), and union_OperationComposition().

4.4 RelMDDIni.c File Reference

```
#include "util.h"
#include "cuddInt.h"
#include "structures.h"
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>
#include <libxml/tree.h>
#include <time.h>
#include "RelMDD.h"
```

Include dependency graph for RelMDDIni.c:



Functions

- void **RelMDD_int** ()

4.4.1 Function Documentation

4.4.1.1 void RelMDD_int ( )

Definition at line 9 of file RelMDDIni.c.

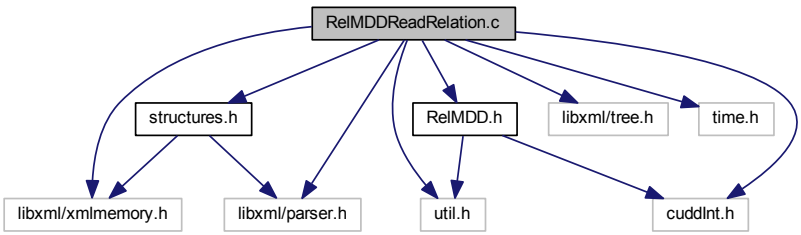
References bck, and dd.

```
                {
 dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
 Cudd_AutodynDisable(dd);
 bck=  Cudd_ReadPlusInfinity( dd );
 Cudd_SetBackground( dd,bck );
}
```

4.5 RelMDDReadRelation.c File Reference

```
#include "util.h"
#include "cuddInt.h"
#include "structures.h"
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>
#include <libxml/tree.h>
#include <time.h>
#include "RelMDD.h"
```

Include dependency graph for RelMDDReadRelation.c:



Functions

- **fileInforPtr RelMdd_ReadFile** (char ∗filename)

4.5.1 Function Documentation

4.5.1.1 fileInforPtr RelMdd_ReadFile (char ∗ *filename*)

CFile

RelMDD has a special format to read in matrices from files. This format adhere to the format of Harwell-Boeing benchmark suite. This format specifies how matrices should be stored and read from a file. In our case the item on the file specifies the number of rows and column of the matrix. The second and third items on the file specify the list of source and target objects respectively. This is followed by the matrix in which the row and column indices of each element of the matrix is also specified. Refer to the appendix for sample file.

Definition at line 22 of file RelMDDReadRelation.c.

References fileInfor::c, fileInfor::r, fileInfor::rel, fileInfor::rowid, fileInfor::rowida, fileInfor::sourcelist, and fileInfor::targetlist.

Referenced by main().

```
{
    FILE *ifp;

    int u, v, err, i;


    double val;

    ifp = fopen(filename, "r");
    if (ifp == NULL) {
        fprintf(stderr, "Can't open input file in.list!\n");
        exit(1);
    }

    err = fscanf(ifp, "%d %d", &u, &v);
    //printf("\n%d \t %d\n", u,  v);
    fileInforPtr ret;
    ret = (fileInforPtr) malloc(sizeof(fileInfor) * u * v);

    ret->r = u;
    ret->c = v;
    // printf("\n%d \t %d\n", ret->r,   ret->c);
    if (err == EOF) {
        fprintf(stderr, "Can't open input file in.list!\n");
        exit(1);
    } else if (err != 2) {
        fprintf(stderr, "Can't open input file in.list!\n");
        exit(1);
    }
```

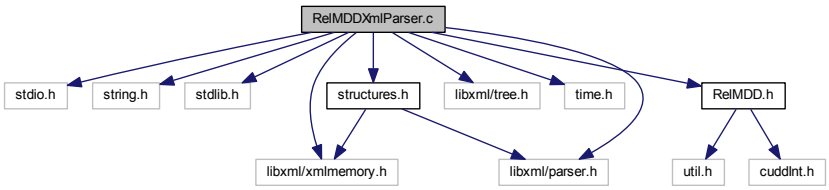
```
    char *rowlist = NULL;
    rowlist = (char *) malloc(u * 3);

    err = fscanf(ifp, "%s", rowlist);

    if (err == EOF) {
        return (0);
    } else if (err != 1) {

        return (0);
    }

    char *collist = NULL;
    collist = (char *) malloc(v * 3);
    err = fscanf(ifp, "%s", collist);

    if (err == EOF) {
        return (0);
    } else if (err != 1) {

        return (0);
    }

    ret->sourcelist = (char *) malloc(sizeof(char));

    char *pch;
    int l = 0;

    pch = strtok(rowlist, "%[^]% [],.-");
    while (pch != NULL) {
        //  printf ("%s\n",pch);
        ret[l].sourcelist = pch;
        l++;
        pch = strtok(NULL, "[] ,%[^]%n.-");
    }


    ret->targetlist = (char *) malloc(sizeof(char));

    char *pchs;
    int ll = 0;

    pchs = strtok(collist, " [],.-");
    while (pchs != NULL) {
        ret[ll].targetlist = pchs;
        ll++;
        pchs = strtok(NULL, "[] ,.-");
    }

    ret->rel = (double *) malloc(sizeof(double) * u * v);

    ret->rowid = (int *) malloc(sizeof(int) * u * v);

    ret->rowida = (int *) malloc(sizeof(int) * u * v);

    i = 0;
    while (!feof(ifp)) {

        err = fscanf(ifp, "%d %d %lf", &u, &v, &val);
        // printf("\n%d \t %d \t%f\n ", u,  v, val);
        //printf("%d",err);
        if (err == EOF) {
            break;
        }
        //else if () {
        //   break;
        else if (err != 3) {

            break;
        }
        //else if (u >= m || v >= n || u < 0 || v < 0) {
        //   break;

        //}
        ret->rowid[i] = u;      //printf("%d\t ", ret->rowid[i]);
        ret->rowida[i] = v;     // printf("%d\t ", ret->rowida[i]);
        //ret->colid[i]=u; printf("%d\t ", ret->colid[i]);
        ret->rel[i] = val;      //printf("%f\n ", ret->rel[i]);
        i++;
    }

    fclose(ifp);

    return ret;
}
```

Here is the caller graph for this function:

4.6 RelMDDXmlParser.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>
#include <libxml/tree.h>
#include <time.h>
#include "structures.h"
#include "RelMDD.h"
```

Include dependency graph for RelMDDXmlParser.c:



Macros

- #define **MAXTOKENS** 250

Functions

- char ∗ **trim** (char ∗s)
- **complementPtr compIIni** ()
- **intersectionPtr interIni** ()
- **unionsPtr unionIni** ()
- xmlDocPtr **getdoc** ()
- double ∗ **readString** (unsigned char ∗s)
- **compoUnionPtr compoUionIni** ()
- **conversePtr coverseIIni** ()
- **compoInterPtr compoInterIni** ()
- **compoInterPtr test** ()
- xmlChar ∗ **relXmlGetComments** (xmlDocPtr doc, xmlNodePtr cur)
- xmlChar ∗ **relXmlGetListObject** (xmlDocPtr doc, xmlNodePtr cur)

- void **relXmlReturnObjects** (xmlDocPtr doc, xmlNodePtr root)
- char ∗∗ **split** (char ∗string)
- **relationPtr relxmlReadrelations** (xmlNode ∗root, xmlDocPtr doc)
- char ∗∗ **relXmlGetNumberRelation** (xmlChar ∗pt, int i)
- **identityPtr relxmlReadIdentity** (xmlNode ∗root, xmlDocPtr doc)
- char ∗∗ **relXmlGetSource** (xmlChar ∗pt, int i)
- **bottomPtr relxmlReadBottom** (xmlNode ∗root, xmlDocPtr doc)
- **topPtr relxmlReadTop** (xmlNode ∗root, xmlDocPtr doc)
- **unionsPtr relxmlReadUnion** (xmlNode ∗root, xmlDocPtr doc)
- char ∗∗ **relXmlGetTarget** (xmlChar ∗pt, int i)
- **intersectionPtr relxmlReadIntersection** (xmlNode ∗root, xmlDocPtr doc)
- **compoUnionPtr relxmlReadCompostion_Union** (xmlNode ∗root, xmlDocPtr doc)
- **compoInterPtr relxmlReadCompostion_Inter** (xmlNode ∗root, xmlDocPtr doc)
- **conversePtr relxmlReadTransposition** (xmlNode ∗root, xmlDocPtr doc)
- **complementPtr relxmlReadComplement** (xmlNode ∗root, xmlDocPtr doc)
- xmlChar ∗ **targetObjects** (xmlChar ∗pt, int i)
- **identityPtr idet** ()
- **bottomPtr bott** ()
- **topPtr topp** ()

Variables

- static int **len_list**
- xmlDocPtr **xmlDocument**

4.6.1 Macro Definition Documentation

4.6.1.1 #define MAXTOKENS 250

CFile This file contains all the functions for maniputions of the xml file content internally. In this file we imported several functions from Libxml2(fucntions are copy right of .... Copyright (C) 1998-2003 Daniel Veillard.All Rights Reserved.)

.....

This functions accepts and xml file and extract the content, making it availble for the RelMDD system Internally. The user need not call any of this functions unless u want to modify it .

Definition at line 17 of file RelMDDXmlParser.c.

4.6.2 Function Documentation

4.6.2.1 bottomPtr bott ( )

Definition at line 1207 of file RelMDDXmlParser.c.

References relxmlReadBottom(), and xmlDocument.

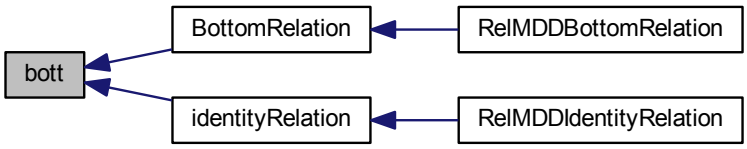
Referenced by BottomRelation(), and identityRelation().

```
{

   // xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);
```

```
    bottomPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadBottom(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.6.2.2 complementPtr compIni ( )

Definition at line 1086 of file RelMDDXmlParser.c.

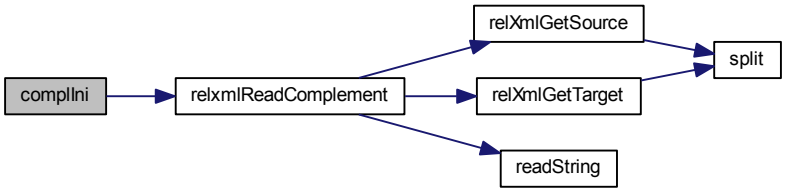
References relxmlReadComplement(), and xmlDocument.

Referenced by complementRelation().

```
{

    //xmlDocPtr doc;
    //doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    complementPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadComplement(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){
```

```
//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.6.2.3 compoInterPtr compoInterIni ( )

Definition at line 1303 of file RelMDDXmlParser.c.

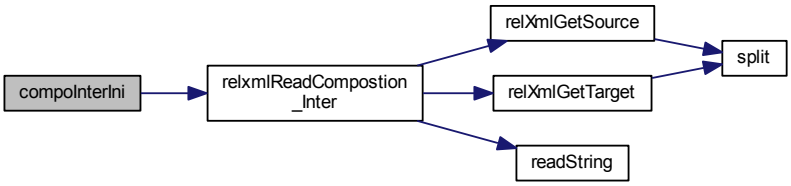
References relxmlReadCompostion_Inter(), and xmlDocument.

Referenced by interserction_OperationComposition().

```
                              {

//xmlDocPtr doc;
 // doc = xmlParseFile("w3.xml");

  if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL ;
  root = xmlDocGetRootElement(xmlDocument);


   compoInterPtr ret;


   // memset(ret, 0, sizeof(identityPtr));
 ret=  relxmlReadCompostion_Inter(  root,xmlDocument);
  //int k,kk;
// printf("\n.................. in main.................................\n ");

 //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
```

```
//}
 if( !root ||
      !root->name ||
      xmlStrcmp(root->name, (xmlChar *)"RelationBasis") )
  {
     xmlFreeDoc(xmlDocument);
     return NULL;
  }

return ret;
 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.6.2.4 compoUnionPtr compoUionIni ( )

Definition at line 1264 of file RelMDDXmlParser.c.

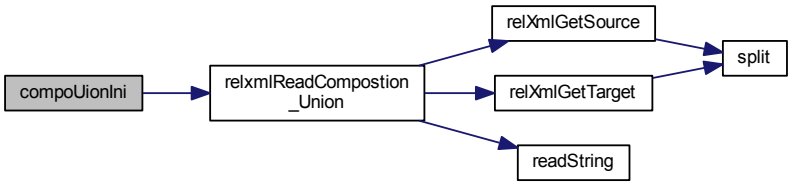
References relxmlReadCompostion_Union(), and xmlDocument.

Referenced by union_OperationComposition().

```
                              {

//xmlDocPtr doc;
  //doc = xmlParseFile("w3.xml");

  if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL ;
  root = xmlDocGetRootElement(xmlDocument);



   compoUnionPtr ret;


   // memset(ret, 0, sizeof(identityPtr));
 ret=  relxmlReadCompostion_Union(  root,xmlDocument);
  //int k,kk;
// printf("\n.................. in main.................................\n ");

 //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}
 if( !root ||
      !root->name ||
      xmlStrcmp(root->name,(xmlChar *)"RelationBasis") )
```

```
  {
     xmlFreeDoc(xmlDocument);
     return NULL;
  }

return ret;
 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.6.2.5 conversePtr coversellni ( )

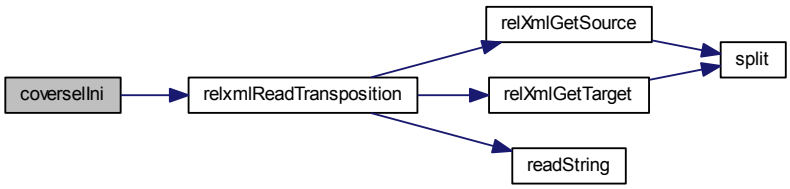
Definition at line 1058 of file RelMDDXmlParser.c.

References relxmlReadTransposition(), and xmlDocument.

Referenced by transposeRelation().

```
{

    //xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument== NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    conversePtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadTransposition(root,xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.6.2.6 xmlDocPtr getdoc ( )

4.6.2.7 identityPtr idet ( )

xmlDocPtr doc;

Definition at line 1179 of file RelMDDXmlParser.c.

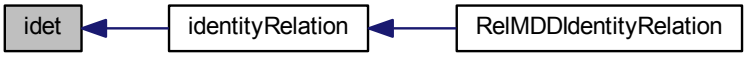
References relxmlReadIdentity(), and xmlDocument.

Referenced by identityRelation().

```
{

    //doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    identityPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadIdentity(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.....................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.6.2.8 intersectionPtr interlni ( )

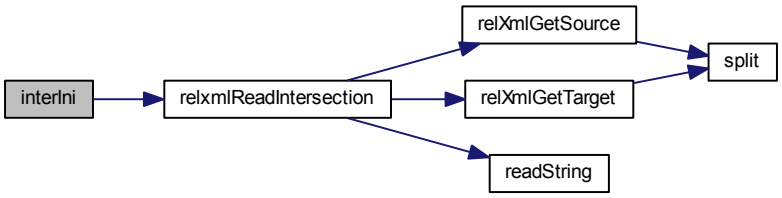
Definition at line 1115 of file RelMDDXmlParser.c.

References relxmlReadIntersection(), and xmlDocument.

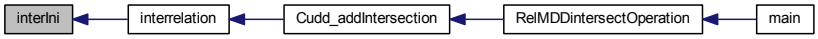
Referenced by interrelation().

```
{

  //  xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL;
    root = xmlDocGetRootElement(xmlDocument);

    intersectionPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadIntersection(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.....................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}
    if (!root || !root->name || xmlStrcmp(root->name, (xmlChar *)"RelationBasis
      ")) {
        xmlFreeDoc(xmlDocument);
        return NULL;
    }

    return ret;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.9 double * readString (unsigned char * *s*)

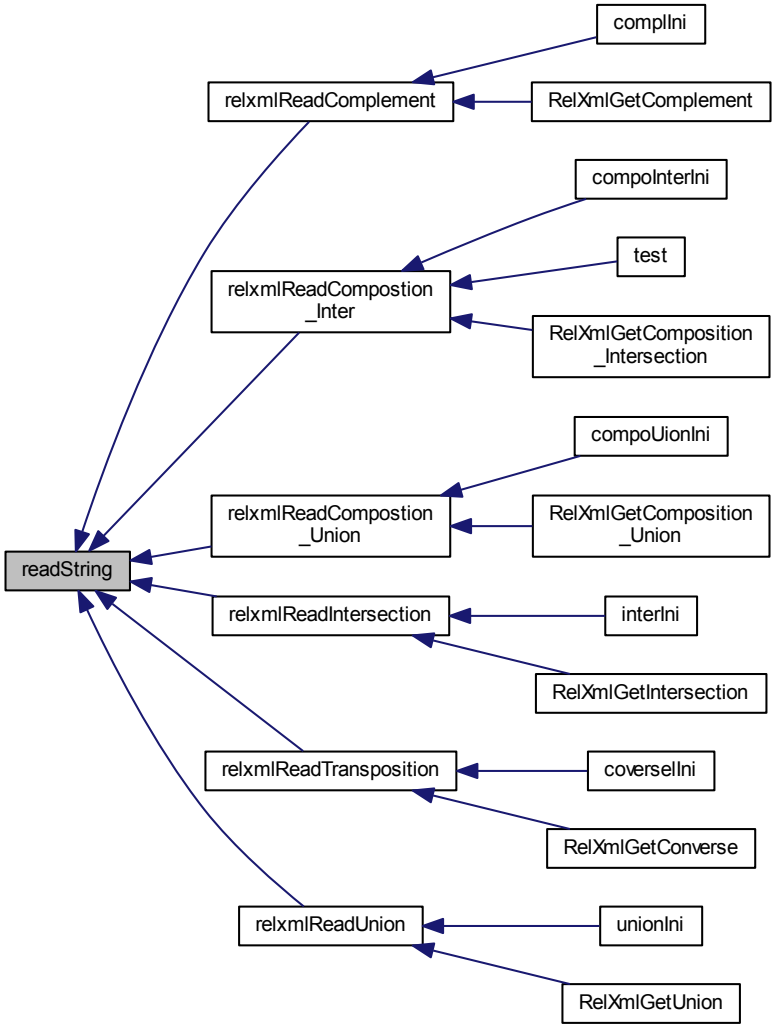
Definition at line 1033 of file RelMDDXmlParser.c.

References len_list.

Referenced by relxmlReadComplement(), relxmlReadCompostion_Inter(), relxmlReadCompostion_Union(), relxmlReadIntersection(), relxmlReadTransposition(), and relxmlReadUnion().

```
{
    float num;
    int nc, i = 0;
    double *list = (double *) malloc((strlen((char *)s) + 1) * sizeof(double));
//sscanf(s, "%[^]%n", &num1, &nd);

    while (sscanf((char *)s, "%f%n", &num, &nc) == 1
           || sscanf((char *)s, "%*[,;]%f%n", &num, &nc) == 1) {

        s += nc;
        list[i] = (double) num;

        i++;
    }
    //printf("the leng in fuction%d",i);
    //for(j=0;j<i;j++)
    //printf("\n%f\n", list[j]);
    //printf("new list%d",i);
    len_list = i;
    //printf("....%d\n.....................................", i);
    return list;
}
```

Here is the caller graph for this function:



4.6.2.10 xmlChar* relXmlGetComments (xmlDocPtr *doc*, xmlNodePtr *cur*)

Function Function: **relXmlGetComments(xmlDocPtr doc, xmlNodePtr cur)** (p. **??**) This function reads the comments from the comments tag and return it ..

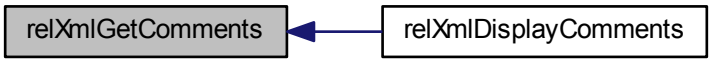
Definition at line 50 of file RelMDDXmlParser.c.

Referenced by relXmlDisplayComments().

```
{

    xmlChar *comments =NULL;

    cur = cur->xmlChildrenNode;
    while (cur != NULL) {
        if ((!xmlStrcmp(cur->name, (const xmlChar *) "comment"))) {

            comments = xmlNodeListGetString(doc, cur->xmlChildrenNode, 1);
            printf("The comments on this basis file:\n%s ", comments);
```

```
        }
        cur = cur->next;

    }

    return (comments);
}
```

Here is the caller graph for this function:



4.6.2.11 xmlChar* relXmlGetListObject (xmlDocPtr *doc,* xmlNodePtr *cur*)

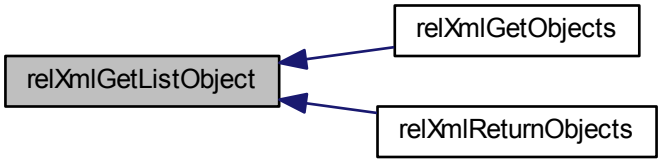
Function Function: **relXmlGetListObject(xmlDocPtr doc, xmlNodePtr cur)** (p. **??**) This fucntion returns the string objects from the xmlfile A,B,C.. unformated

Definition at line 79 of file RelMDDXmlParser.c.

Referenced by relXmlGetObjects(), and relXmlReturnObjects().

```
{

    xmlChar *object =NULL;

    cur = cur->xmlChildrenNode;
    while (cur != NULL) {
        if ((!xmlStrcmp(cur->name, (const xmlChar *) "objects"))) {

            object = xmlNodeListGetString(doc, cur->xmlChildrenNode, 1);

        }
        cur = cur->next;

    }

    return (object);
}
```

Here is the caller graph for this function:



4.6.2.12 char** relXmlGetNumberRelation (xmlChar * *pt*, int *i*)

Function Function: relXmlGetNumberRelation(xmlDocPtr doc, xmlNodePtr cur) returns list of number of relatinos from the relations tag , you must free the return value after use:

Definition at line 255 of file RelMDDXmlParser.c.

References split().

```
{   //char *string
    char **number;
    number = split((char *)pt);

    for (i = 0; number[i] != NULL; i++)

        printf("number %s\n", number[i]);

    return number;
}
```

Here is the call graph for this function:

4.6.2.13 char** relXmlGetSource (xmlChar * *pt*, int *i*)

Function Function: **relXmlGetSource(xmlChar * pt, int i)** (p. **??**) get the object attribute of

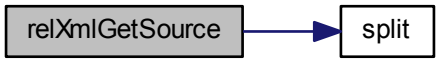
Definition at line 336 of file RelMDDXmlParser.c.

References split().

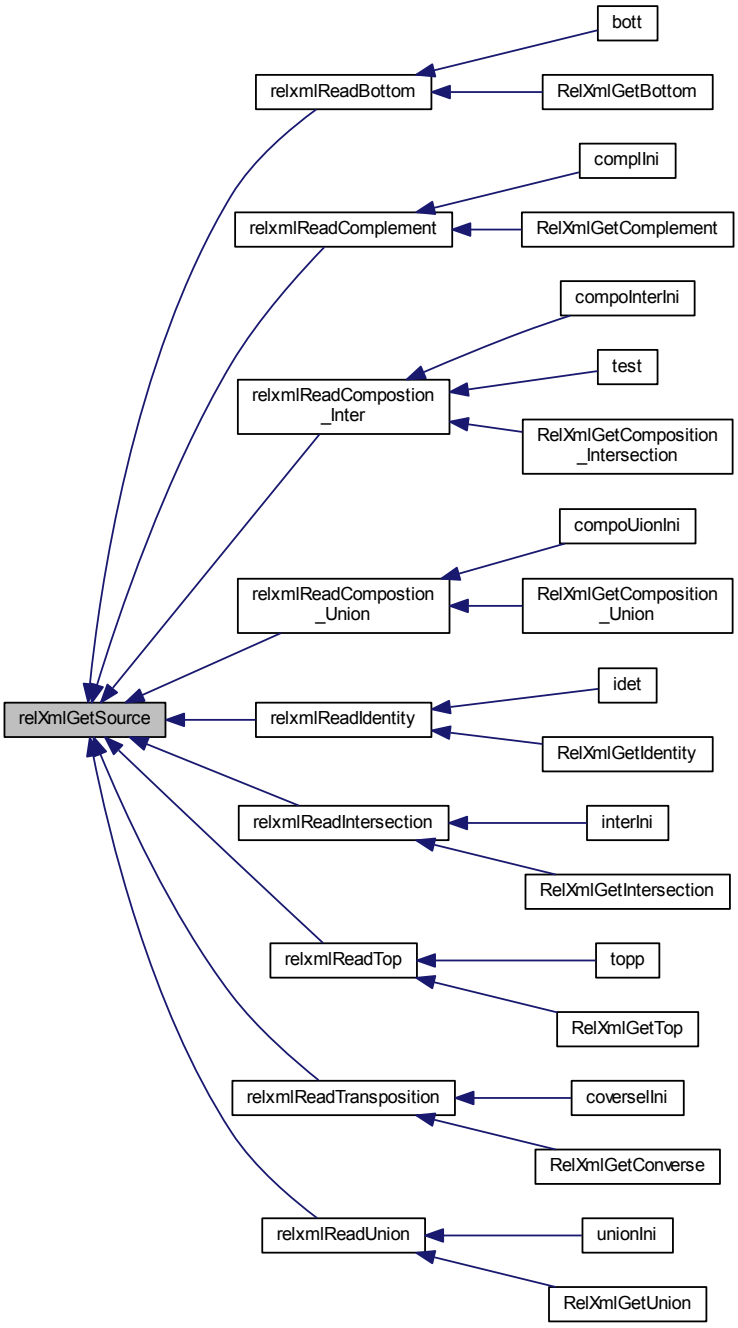
Referenced by relxmlReadBottom(), relxmlReadComplement(), relxmlReadCompostion_Inter(), relxmlReadCompostion_Union(), relxmlReadIdentity(), relxmlReadIntersection(), relxmlReadTop(), relxmlReadTransposition(), and relxmlReadUnion().

```
{
    char **tokens;
    tokens = split((char *)pt);
    return tokens;
}/*End of relxmlGetSource*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.14 char** relXmlGetTarget (xmlChar * *pt,* int *i*)

Definition at line 588 of file RelMDDXmlParser.c.

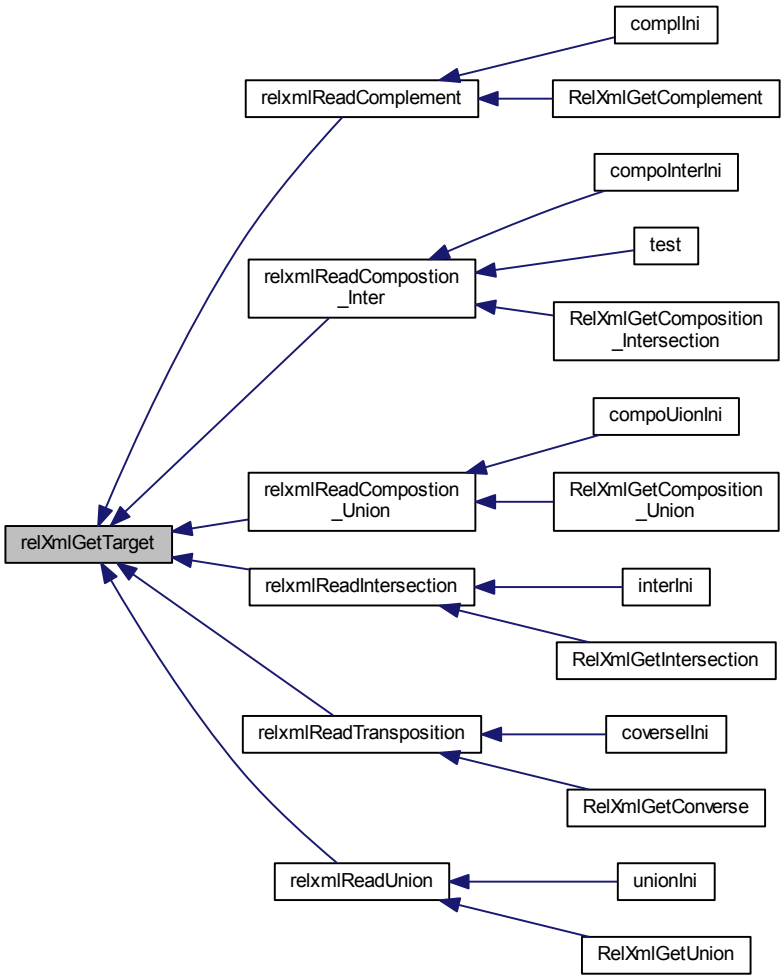
References split().

Referenced by relxmlReadComplement(), relxmlReadCompostion_Inter(), relxmlReadCompostion_Union(), relxmlReadIntersection(), relxmlReadTransposition(), and relxmlReadUnion().

```
{
    char **tokens;
    tokens = split((char *)pt);
    return tokens;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.6.2.15 bottomPtr relxmlReadBottom (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: **relxmlReadBottom(xmlNode * root, xmlDocPtr doc)** (p. **??**) read the bottom relation or tranverse the bottom relations

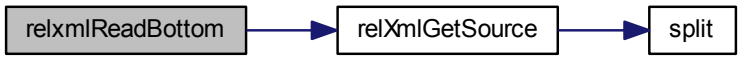
Definition at line 350 of file RelMDDXmlParser.c.

References bottom::len, bottom::rel, relXmlGetSource(), bottom::source, and bottom::target.

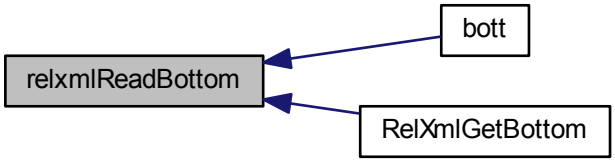
Referenced by bott(), and RelXmlGetBottom().

```
{

    xmlNode *cur_node, *child_node;
    xmlChar *sourceAtrr, *targetAtrr, *numberAtrr;
    int i;
    bottomPtr ret;
    ret = (bottomPtr) malloc(sizeof(bottom) * 100);//increase this if needed
    ret->rel = (double *) malloc(sizeof(double) * 100);// increase this if
       needed
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "bottom")) {

                    //get the object attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                        // ret->source[i] =sourceAtrr;
                    }
                    // get the relations attribute of the identity.
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        //relXmlGetBottomTarget(targetAtrr,i);
                        //ret->target[i] =targetAtrr;
                        ret[i].target = relXmlGetSource(targetAtrr, i);
                    }

                    numberAtrr =
                        xmlGetProp(child_node,
                                   (const xmlChar *) "relation");
                    if (numberAtrr) {
                        ret->rel[i] = atof((char *)numberAtrr);
                        ret->len = i + 1;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(numberAtrr);
                    i++;

                }

            }

        }

    }

   /* int j;
    for (j = 0; j < ret->len; j++)

        printf("bottom \t%s\t%s\t %f\n", *ret[j].source, *ret[j].target,
               ret->rel[j]);*/

    return ret;
}/*End of relxmlReadBottom */
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.16 complementPtr relxmlReadComplement (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: **relxmlReadComplement(xmlNode * root, xmlDocPtr doc)** (p. **??**) This functions returns the Complement data from the xml file ....

Definition at line 941 of file RelMDDXmlParser.c.

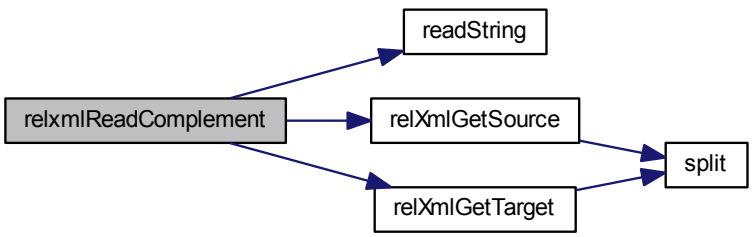
References complement::content, complement::len, complement::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), complement::source, and complement::target.

Referenced by complIni(), and RelXmlGetComplement().

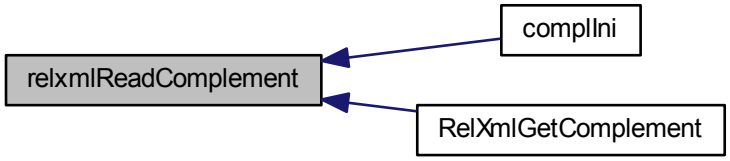
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    complement *ret = NULL;
    ret = (complement *) malloc(sizeof(complement) * 100);
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
    }
    ret->content = (double *) malloc(sizeof(double) * 100);
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
```

```
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "complement")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
       ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len = i + 1;

                    }
                    content_union = xmlNodeGetContent(child_node);

                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
      printf("cfvffd....%d\n.........................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }
    /* int k,kk;

       for(kk=0;kk<ret->len;kk++){
       printf("\nstarting union here so please be read thank you God\n ");

       printf("  %s || \t%s \n\t..%d..", *ret[kk].source, *ret[kk].target,
       ret->len_Content[kk]);
       for(k=0;k<ret->len_Content[kk];k++)
       printf("  %f\n", ret[kk].content[k]);} */
    xmlCleanupParser();

    return ret;
}/*End of complement*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.17 compoInterPtr relxmlReadCompostion_Inter (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: **relxmlReadCompostion_Inter(xmlNode * root, xmlDocPtr doc)** (p. **??**) This functions returns the Compostion_Intersection data from the xml file ....

Definition at line 773 of file RelMDDXmlParser.c.

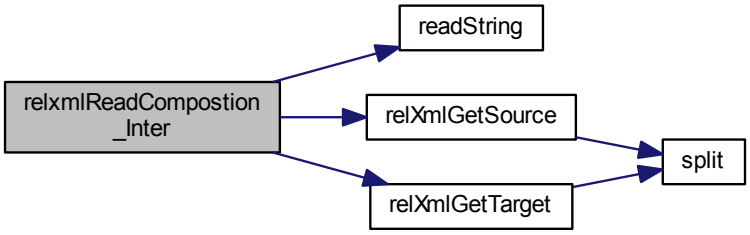
References compoInter::content, compoInter::len, compoInter::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), compoInter::source, and compoInter::target.

Referenced by compoInterIni(), RelXmlGetComposition_Intersection(), and test().

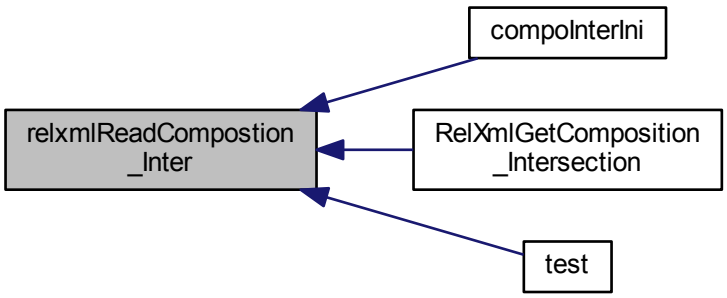
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    compoInter *ret = NULL;
    ret = (compoInter *) malloc(sizeof(compoInter) * 100);
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
    }
    ret->content = (double *) malloc(sizeof(double) * 100);
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name, (const xmlChar *)
                               "composition_Intersection")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
    ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len = i + 1;
                    }
                    content_union = xmlNodeGetContent(child_node);
```

```
                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
      printf("cfvffd....%d\n.....................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }

    xmlCleanupParser();

    return ret;
}/*End of relxmlReadCompostion_Inter*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.18 compoUnionPtr relxmlReadCompostion_Union (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: relxmlReadCompostion_Union(xmlDocPtr doc, xmlNodePtr cur) This functions returns the Compostion_Union data from the xml file ....

Definition at line 692 of file RelMDDXmlParser.c.

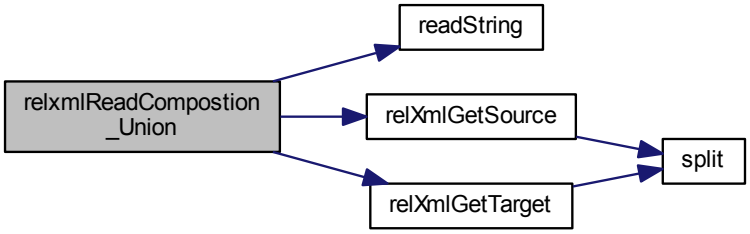
References compoUnion::content, compoUnion::len, compoUnion::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), compoUnion::source, and compoUnion::target.

Referenced by compoUionIni(), and RelXmlGetComposition_Union().

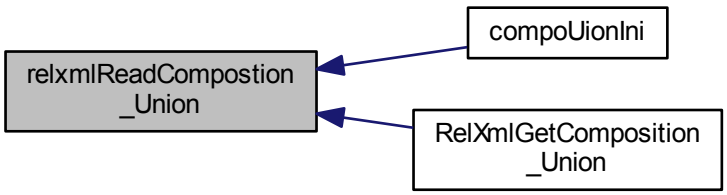
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    compoUnion *ret = NULL;
    ret = (compoUnion *) malloc(sizeof(compoUnion) * 100);
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
    }
    ret->content = (double *) malloc(sizeof(double) * 100);
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "composition_Union")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
       ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len=i+1;
                    }
                    content_union = xmlNodeGetContent(child_node);

                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
      printf("cfvffd....%d\n.....................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }
    xmlCleanupParser();
```

```
    return ret;
}/*End of relxmlReadCompostion_Union*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.19 identityPtr relxmlReadIdentity (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: relXmlReadIdentity(xmlDocPtr doc, xmlNodePtr cur) returns the identity list from the identity tag: to get the identity relation

Definition at line 275 of file RelMDDXmlParser.c.

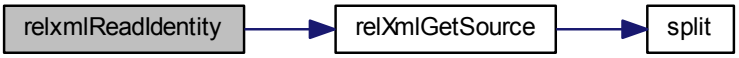
References identity::len, identity::object, identity::rel, and relXmlGetSource().

Referenced by idet(), and RelXmlGetIdentity().

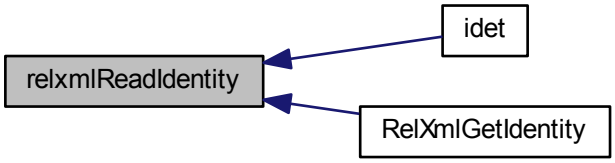
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *sourceAtrr, *targetAtrr;
    int i;
    identityPtr ret;
    ret = (identityPtr) malloc(sizeof(identity) * 100); // increase this if you
       are dealing with large data set
    ret->rel = (double *) malloc(sizeof(double) * 100);
    ret->object = (char **) malloc(sizeof(char));
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {
```

```
            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "identity")) {

                    //get the object attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "object");
                    if (sourceAtrr) {

                        ret[i].object = relXmlGetSource(sourceAtrr, i);
                        ret->len = i + 1;
                    }
                    // get the relations attribute of the identity.
                    targetAtrr =
                        xmlGetProp(child_node,
                                   (const xmlChar *) "relation");
                    if (targetAtrr) {
                        ret->rel[i] = atof((char *)targetAtrr);

                    }
                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    i++;
                }

            }

        }
    }
    //int j;
    // for(j = 0; j<ret->len; j++)

//printf("ok \t%s\n",*ret[j].object);
    return ret;
}/*End of  relxmlReadIdentity */
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.20 intersectionPtr relxmlReadIntersection (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: **relxmlReadIntersection(xmlNode * root, xmlDocPtr doc)** (p. **??**) This functions returns the intersection data from the xml file ....

Definition at line 602 of file RelMDDXmlParser.c.

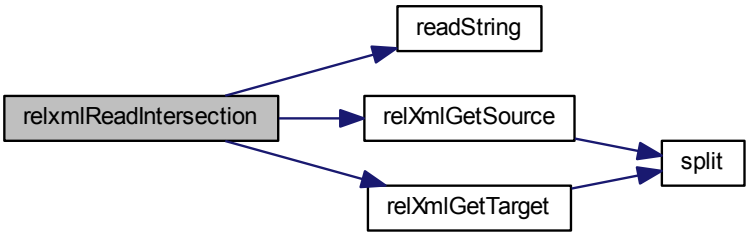
References intersection::content, intersection::len, intersection::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), intersection::source, and intersection::target.

Referenced by interIni(), and RelXmlGetIntersection().

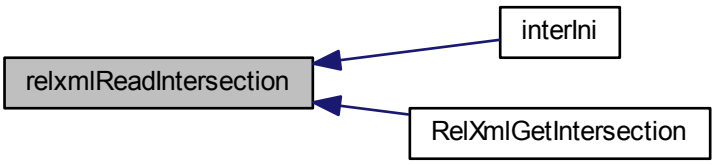
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    intersection *ret = NULL;
    ret = (intersection *) malloc(sizeof(intersection) * 100);
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
    }
    ret->content = (double *) malloc(sizeof(double) * 100);
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "intersection")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
       ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len = i + 1;
                    }
                    content_union = xmlNodeGetContent(child_node);

                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
      printf("cfvffd....%d\n.....................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;

                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }
    /* int k,kk;
```

```
        printf("\n.................intersection
        ...................................\n ");

        for(kk=0;kk<ret->len;kk++){

        printf("  %s || \t%s \n", *ret[kk].source, *ret[kk].target );
        for(k=0;k<ret->len_Content[kk];k++)
        printf("  %f\n", ret[kk].content[k] );} */
    xmlCleanupParser();

    return ret;
}/*End of relxmlReadIntersection*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.21 relationPtr relxmlReadrelations (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: **relxmlReadrelations(xmlNode * root, xmlDocPtr doc)** (p. **??**) This fucntion extract the content of the relation tag

Definition at line 190 of file RelMDDXmlParser.c.

References relation::number, relation::source, and relation::target.

Referenced by RelXmlGetRelation().

```
{

    xmlNode *cur_node, *child_node;
    xmlChar *sourceAtrr, *targetAtrr, *numberAtrr;
    int i;
    relationPtr ret;
    ret = (relationPtr) malloc(sizeof(relationPtr));
```

```
    ret->number = (int *) malloc(sizeof(int) * 100);// modify this if the
       number of objects are more than 100
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "relations")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {

                        ret->source[i] = (char *)sourceAtrr;
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
       ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {

                        ret->target[i] = (char *)sourceAtrr;
                    }
                    numberAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "number");
                    if (numberAtrr) {

                        ret->number[i] = atoi((char*)numberAtrr);

                    }
                    i++;

                }

            }

        }
    }
//xmlCleanupParser();
    return ret;
}
```

Here is the caller graph for this function:

4.6.2.22 topPtr relxmlReadTop (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: topPtr **relxmlReadTop(xmlNode * root, xmlDocPtr doc)** (p. **??**) This fucntion reads the top relation or tranverse the top relations

Definition at line 428 of file RelMDDXmlParser.c.

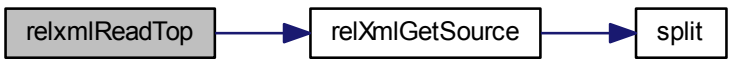
References top::len, top::rel, relXmlGetSource(), top::source, and top::target.

Referenced by RelXmlGetTop(), and topp().

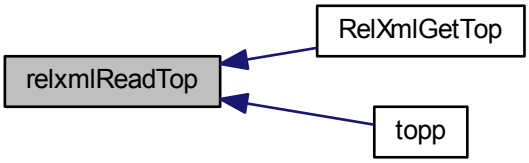
```
{
```

```
    xmlNode *cur_node, *child_node;
    xmlChar *sourceAtrr, *targetAtrr, *numberAtrr;
    int i;
    topPtr ret;
    ret = (topPtr) malloc(sizeof(top) * 100); //increase
    ret->rel = (double *) malloc(sizeof(double) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name, (const xmlChar *) "top"))
                {

                    //get the object attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {

                        //ret->source[i] =sourceAtrr;
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the relations attribute of the identity.
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetSource(targetAtrr, i);
                        //ret->target[i] =targetAtrr;
                    }
                    numberAtrr =
                        xmlGetProp(child_node,
                                   (const xmlChar *) "relation");
                    if (numberAtrr) {

                        ret->rel[i] = atof((char *)numberAtrr);
                        ret->len = i + 1;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(numberAtrr);

                    i++;

                }

            }

        }
    }

    // for(j = 0; j<ret->len; j++)

//printf("bottom \t%s\t%s\t %f\n",*ret[j].source,*ret[j].target,ret->rel[j]);

    return ret;
}/*End of relxmlReadTop*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.23 conversePtr relxmlReadTransposition (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: **relxmlReadTransposition(xmlNode * root, xmlDocPtr doc)** (p. **??**) This functions returns the transposition data from the xml file ....

Definition at line 857 of file RelMDDXmlParser.c.

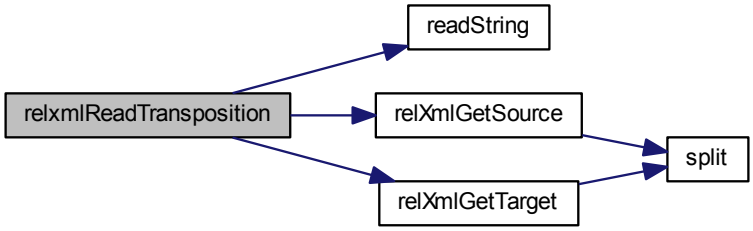
References converse::content, converse::len, converse::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), converse::source, and converse::target.

Referenced by coverselIni(), and RelXmlGetConverse().

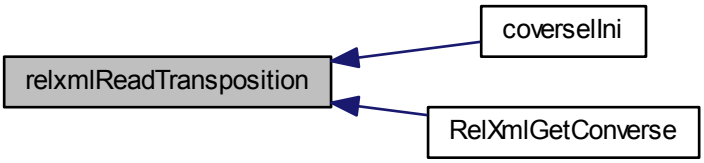
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    converse *ret = NULL;
    ret = (converse *) malloc(sizeof(converse) * 100);
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
    }

    ret->content = (double *) malloc(sizeof(double) * 100);
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "transposition")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
   ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len = i + 1;

                    }
```

```
                    content_union = xmlNodeGetContent(child_node);

                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
      printf("cfvffd....%d\n.....................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }

    xmlCleanupParser();

    return ret;
}/*End of transposition*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.24 unionsPtr relxmlReadUnion (xmlNode * *root*, xmlDocPtr *doc*)

Function Function:unionsPtr **relxmlReadUnion(xmlNode * root, xmlDocPtr doc)** (p. **??**) This functions returns the union data from the xml file ....

Definition at line 503 of file RelMDDXmlParser.c.

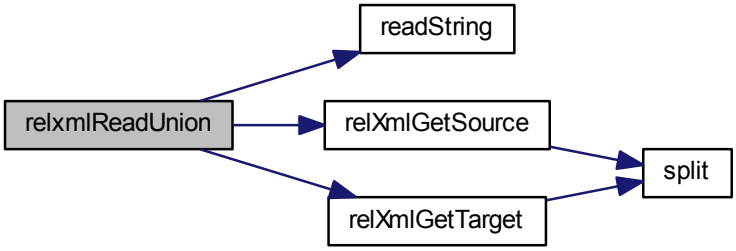
References unions::content, unions::len, unions::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), unions::source, and unions::target.

Referenced by RelXmlGetUnion(), and unionIni().

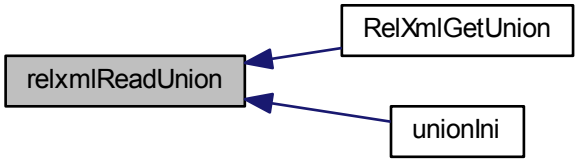
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    unions *ret = NULL;
    ret = (unions *) malloc(sizeof(unions) * 100);
    ret->content = (double *) malloc(sizeof(double) * 100);
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    // basis *ret = (basis*) calloc(40,sizeof(basis));
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
    }
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "union")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
       ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len = i + 1;
                    }
                    content_union = xmlNodeGetContent(child_node);

                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
      printf("cfvffd....%d\n.....................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }
```

```
    /*int k,kk;

       for(kk=0;kk<ret->len;kk++){
       printf("\nstarting union here so please be read thank you God\n ");

       printf("  %s || \t%s \n\t..%d..", *ret[kk].source, *ret[kk].target,
       ret->len_Content[kk]);
       for(k=0;k<ret->len_Content[kk];k++)
       printf("  %f\n", ret[kk].content[k]);} */

    xmlCleanupParser();

    return ret;
}/*End relxmlReadUnion */
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.25 void relXmlReturnObjects (xmlDocPtr *doc*, xmlNodePtr *root*)

Function Function:**relXmlReturnObjects(xmlDocPtr doc, xmlNodePtr root)** (p. **??**) this functions displays the list of objects from the object tag.

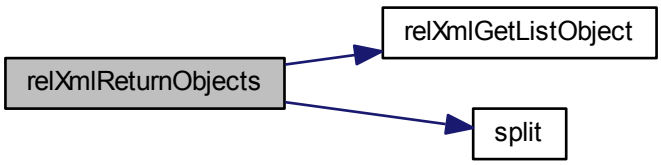
Definition at line 105 of file RelMDDXmlParser.c.

References relXmlGetListObject(), and split().

```
{
    root = root->xmlChildrenNode;
    char *string;
    char **tokens = NULL;
    while (root != NULL) {
```

```
        if ((!xmlStrcmp(root->name, (const xmlChar *) "relation"))) {
            string = (char *)relXmlGetListObject(doc, root);

            printf("list of objects from the object in basis file %s\n",
                   string);
            tokens = (char**)split(string);
        }

        root = root->next;
    }

    int i = 0;

    for (i = 0; tokens[i] != NULL; i++)

        printf("%02dobjects broken into token: %s\n", i, tokens[i]);

    for (i = 0; tokens[i] != NULL; i++)
        free(tokens[i]);

    free(tokens);

}
```

Here is the call graph for this function:



4.6.2.26 char** split (char * *string*)

Function Function: **split(char *string)** (p. **??**) split string into tokens, return token array it serve as tokenizer

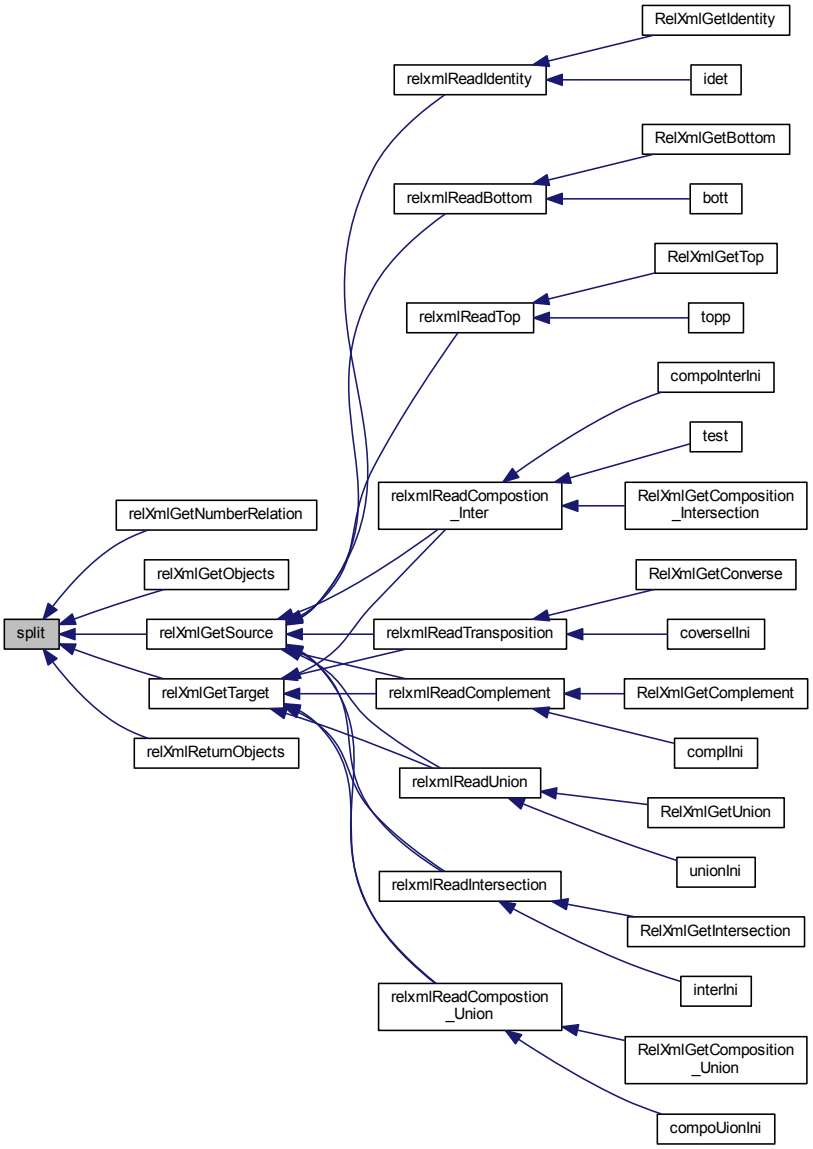
Definition at line 143 of file RelMDDXmlParser.c.

Referenced by relXmlGetNumberRelation(), relXmlGetObjects(), relXmlGetSource(), relXmlGetTarget(), and relXmlReturnObjects().

```
{

    char *delim = ".,:;'\"+-_(){}[]<>*&^%$#@!?~/|\\= \t\r\n";
    char **tokens = NULL;
    char *working = NULL;
    char *token = NULL;
    int idx = 0;
    int len = strlen(string) + 1;

    tokens = malloc(sizeof(char *) * len);
    if (tokens == NULL)
        return NULL;
    working = malloc(sizeof(char) * strlen(string) + 1);
    if (working == NULL)
        return NULL;

    /* to make sure, copy string to a safe place */
    strcpy(working, string);
    for (idx = 0; idx < len; idx++)
        tokens[idx] = NULL;

    token = strtok(working, delim);
    idx = 0;
```

```
    /* always keep the last entry NULL termindated */
    while ((idx < (len - 1)) && (token != NULL)) {
        tokens[idx] = malloc(sizeof(char) * strlen(token) + 1);
        if (tokens[idx] != NULL) {
            strcpy(tokens[idx], token);
            idx++;
            token = strtok(NULL, delim);
        }
    }

    free(working);
    return tokens;
}
```

Here is the caller graph for this function:



4.6.2.27 xmlChar* targetObjects (xmlChar * *pt*, int *i*)

Definition at line 1024 of file RelMDDXmlParser.c.

```
{

    xmlChar *ptt[4];
    ptt[i] = pt;
//printf(" target objects%s%d \n", ptt[i],i);
    return *ptt;
}
```

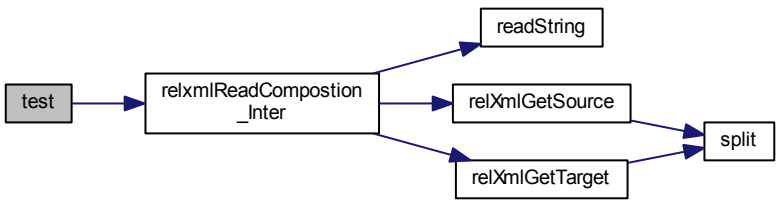
4.6.2.28 compoInterPtr test ( )

Definition at line 1340 of file RelMDDXmlParser.c.

References relxmlReadCompostion_Inter(), and xmlDocument.

```
                    {

//xmlDocPtr doc;
  //doc = xmlParseFile("w3.xml");

  if (xmlDocument== NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL ;
  root = xmlDocGetRootElement(xmlDocument);



   compoInterPtr ret;


   // memset(ret, 0, sizeof(identityPtr));
 ret=  relxmlReadCompostion_Inter(  root,xmlDocument);
  int k,kk;
printf("\n.................. in main.....................................\n ");

  for(kk=0;kk<4;kk++){

printf("  %s || \t%s \n", *ret[kk].source, *ret[kk].target );
for(k=0;k<20;k++)
printf("  %f\n", ret[kk].content[k] );
}
 if( !root ||
      !root->name ||
      xmlStrcmp(root->name,(xmlChar *)"RelationBasis") )
  {
     xmlFreeDoc(xmlDocument);
     return NULL;
  }

return ret;
 }
```

Here is the call graph for this function:



4.6.2.29 topPtr topp ( )

Definition at line 1235 of file RelMDDXmlParser.c.

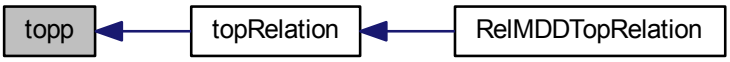
References relxmlReadTop(), and xmlDocument.

Referenced by topRelation().

```
{

    //xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");
    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    topPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadTop(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;

}
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.6.2.30 char* trim (char * *s*)

4.6.2.31 unionsPtr unionIni ( )

Definition at line 1147 of file RelMDDXmlParser.c.

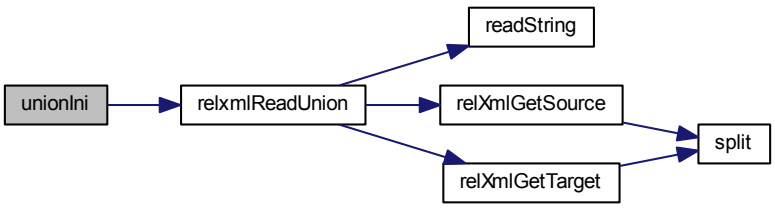
References relxmlReadUnion(), and xmlDocument.

Referenced by unrelation().

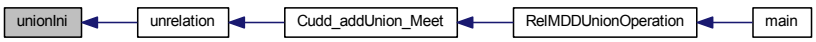
```
{
```

```
   // xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL;
    root = xmlDocGetRootElement(xmlDocument);

    unionsPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadUnion(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}
    if (!root || !root->name || xmlStrcmp(root->name, (xmlChar *)"RelationBasis
      ")) {
        xmlFreeDoc(xmlDocument);
        return NULL;
    }

    return ret;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3 Variable Documentation

4.6.3.1 int len_list [static]

Prototypes decalations of functions in this file.

Definition at line 29 of file RelMDDXmlParser.c.

Referenced by readString(), relxmlReadComplement(), relxmlReadCompostion_Inter(), relxmlReadCompostion_-Union(), relxmlReadIntersection(), relxmlReadTransposition(), and relxmlReadUnion().

4.6.3.2 xmlDocPtr xmlDocument

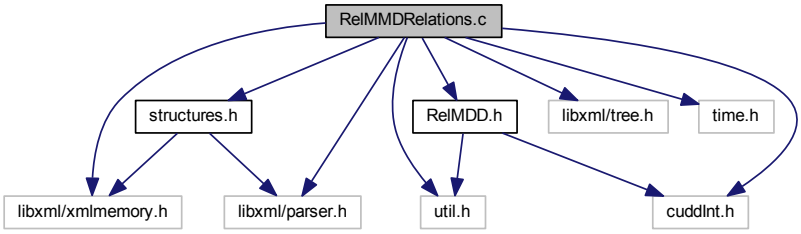
Definition at line 42 of file RelMDDXmlParser.c.

Referenced by bott(), complIni(), compoInterIni(), compoUionIni(), coverselIni(), idet(), interIni(), RelMDDParseXmlFile(), relXmlDisplayComments(), RelXmlGetBottom(), RelXmlGetComplement(), RelXmlGetComposition_Intersection(), RelXmlGetComposition_Union(), RelXmlGetConverse(), RelXmlGetIdentity(), RelXmlGetIntersection(), relXmlGetObjects(), RelXmlGetRelation(), RelXmlGetTop(), RelXmlGetUnion(), test(), topp(), and unionIni().

4.7 RelMMDRelations.c File Reference

```
#include "util.h"
#include "cuddInt.h"
#include "structures.h"
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>
#include <libxml/tree.h>
#include <time.h>
#include "RelMDD.h"
```

Include dependency graph for RelMMDRelations.c:



Functions

- int **RelMdd_CreateRelation** (DdManager ∗**dd**, DdNode ∗∗E, DdNode ∗∗∗x, DdNode ∗∗∗y, DdNode ∗∗∗xn, DdNode ∗∗∗yn_, int ∗nx, int ∗ny, int ∗m, int ∗n, int bx, int sx, int by, int sy, int row_nn, int col_nn, int ∗row_n, int ∗col_n, double ∗id_relation)

4.7.1 Function Documentation

4.7.1.1 int RelMdd_CreateRelation (DdManager ∗ *dd,* DdNode ∗∗ *E,* DdNode ∗∗∗ *x,* DdNode ∗∗∗ *y,* DdNode ∗∗∗ *xn,* DdNode ∗∗∗ *yn_,* int ∗ *nx,* int ∗ *ny,* int ∗ *m,* int ∗ *n,* int *bx,* int *sx,* int *by,* int *sy,* int *row_nn,* int *col_nn,* int ∗ *row_n,* int ∗ *col_n,* double ∗ *id_relation*)

CFile

Definition at line 27 of file RelMMDRelations.c.

Referenced by RelMDD_CreateCompostionVariables(), RelMDD_RenameRelation(), and RelMDD_SwapVariables().

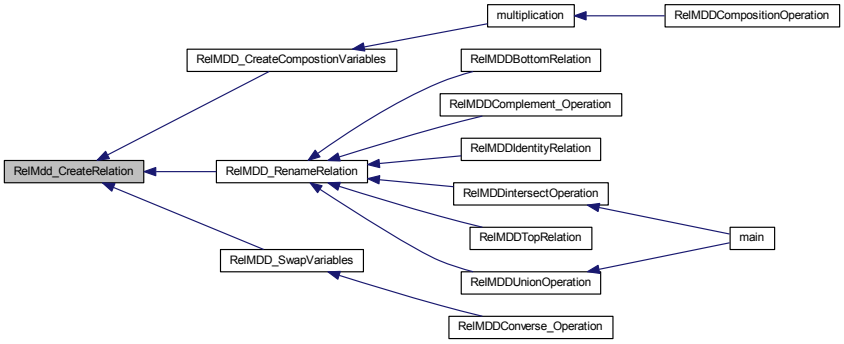
```
{
    // dd = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
```

```
DdNode *one, *zero;
DdNode *w, *neW;
DdNode *minterm1;
int u, v, i, nv;
int lnx, lny;
//CUDD_VALUE_TYPE val;
// CUDD_VALUE_TYPE *val1;
//val1 =(double *) malloc(sizeof(double));
// val1 = RelMDD_Identity();
//  save;

// CUDD_VALUE_TYPE val1 []={1,0,0, 0,0,2,0, 0,0,0,3, 0,0,0,4};
//CUDD_VALUE_TYPE *val1=RelMDD_Identity();
CUDD_VALUE_TYPE *val1 = id_relation;
int kkk=0;

// int uu [] ={0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4};
int *uu;
uu = row_n;
// int vv [] ={0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4};
int *vv;
vv = col_n;
int pp = 0;
int ppp = 0;
DdNode **lx, **ly, **lxn, **lyn;

one = DD_ONE(dd);
zero = DD_ZERO(dd);

/* err = fscanf(fp, "%d %d", &u, &v);
   if (err == EOF) {
   return(0);
   } else if (err != 2) {
   return(0);
   } */
u = row_nn;
v = col_nn;
*m = u;
/* Compute the number of x variables. */
lx = *x;
lxn = *xn;
u--;                            /* row and column numbers start from 0 */
for (lnx = 0; u > 0; lnx++) {
    u >>= 1;
}
/* Here we rely on the fact that REALLOC of a null pointer is
 ** translates to an ALLOC.
 */
if (lnx > *nx) {
    *x = lx = REALLOC(DdNode *, *x, lnx);
    if (lx == NULL) {
        dd->errorCode = CUDD_MEMORY_OUT;
        return (0);
    }
    *xn = lxn = REALLOC(DdNode *, *xn, lnx);
    if (lxn == NULL) {
        dd->errorCode = CUDD_MEMORY_OUT;
        return (0);
    }
}

*n = v;
/* Compute the number of y variables. */
ly = *y;
lyn = *yn_;
v--;                            /* row and column numbers start from 0 */
for (lny = 0; v > 0; lny++) {
    v >>= 1;
}
/* Here we rely on the fact that REALLOC of a null pointer is
 ** translates to an ALLOC.
 */
if (lny > *ny) {
    *y = ly = REALLOC(DdNode *, *y, lny);
    if (ly == NULL) {
        dd->errorCode = CUDD_MEMORY_OUT;
        return (0);
    }
    *yn_ = lyn = REALLOC(DdNode *, *yn_, lny);
    if (lyn == NULL) {
        dd->errorCode = CUDD_MEMORY_OUT;
        return (0);
    }
}

/* Create all new variables. */
for (i = *nx, nv = bx + (*nx) * sx; i < lnx; i++, nv += sx) {
```

```
        do {
            dd->reordered = 0;
            lx[i] = cuddUniqueInter(dd, nv, one, zero);
        } while (dd->reordered == 1);
        if (lx[i] == NULL)
            return (0);
        cuddRef(lx[i]);
        do {
            dd->reordered = 0;
            lxn[i] = cuddUniqueInter(dd, nv, zero, one);
        } while (dd->reordered == 1);
        if (lxn[i] == NULL)
            return (0);
        cuddRef(lxn[i]);
    }
    for (i = *ny, nv = by + (*ny) * sy; i < lny; i++, nv += sy) {
        do {
            dd->reordered = 0;
            ly[i] = cuddUniqueInter(dd, nv, one, zero);
        } while (dd->reordered == 1);
        if (ly[i] == NULL)
            return (0);
        cuddRef(ly[i]);
        do {
            dd->reordered = 0;
            lyn[i] = cuddUniqueInter(dd, nv, zero, one);
        } while (dd->reordered == 1);
        if (lyn[i] == NULL)
            return (0);
        cuddRef(lyn[i]);
    }
    *nx = lnx;
    *ny = lny;

    *E = dd->background;        /* this call will never cause reordering */
    cuddRef(*E);

    while (val1) {
        /* = fscanf(fp, "%d %d %lf", &u, &v, &val);
           if (err == EOF) {
           break;
           } else if (err != 3) {
           return(0);
           } */
        if (uu[pp] >= *m || vv[ppp] >= *n || uu[pp] < 0 || vv[ppp] < 0) {
            return (0);
        }

        minterm1 = one;
        cuddRef(minterm1);

        /* Build minterm1 corresponding to this arc */
        for (i = lnx - 1; i >= 0; i--) {
            if (uu[pp] & 1) {
                w = Cudd_addApply(dd, Cudd_addTimes, minterm1, lx[i]);
            } else {
                w = Cudd_addApply(dd, Cudd_addTimes, minterm1, lxn[i]);
            }
            if (w == NULL) {
                Cudd_RecursiveDeref(dd, minterm1);
                return (0);
            }
            cuddRef(w);
            Cudd_RecursiveDeref(dd, minterm1);
            minterm1 = w;
            uu[pp] >>= 1;
        }
        for (i = lny - 1; i >= 0; i--) {
            if (vv[ppp] & 1) {
                w = Cudd_addApply(dd, Cudd_addTimes, minterm1, ly[i]);
            } else {
                w = Cudd_addApply(dd, Cudd_addTimes, minterm1, lyn[i]);
            }
            if (w == NULL) {
                Cudd_RecursiveDeref(dd, minterm1);
                return (0);
            }
            cuddRef(w);
            Cudd_RecursiveDeref(dd, minterm1);
            minterm1 = w;
            vv[ppp] >>= 1;
        }
        /* Create new constant node if necessary.
         ** This call will never cause reordering.
         */
        neW = cuddUniqueConst(dd, val1[kkk]);
        if (neW == NULL) {
```

```
            Cudd_RecursiveDeref(dd, minterm1);
            return (0);
        }
        cuddRef(neW);

        w = Cudd_addIte(dd, minterm1, neW, *E);
        if (w == NULL) {
            Cudd_RecursiveDeref(dd, minterm1);
            Cudd_RecursiveDeref(dd, neW);
            return (0);
        }
        cuddRef(w);
        Cudd_RecursiveDeref(dd, minterm1);
        Cudd_RecursiveDeref(dd, neW);
        Cudd_RecursiveDeref(dd, *E);
        *E = w;
        kkk++;
        pp++;
        ppp++;
    }
    return (1);

}
```

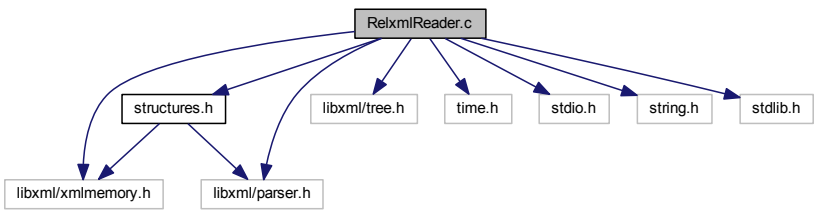
Here is the caller graph for this function:



4.8 RelxmlReader.c File Reference

```
#include "structures.h"
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>
#include <libxml/tree.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

Include dependency graph for RelxmlReader.c:



Functions

- int **RelMDDValidateXmlFile** (const char ∗xmlPath)
- int **RelMDDParseXmlFile** (const char ∗xmlFilename)
- **unionsPtr RelXmlGetUnion** ()
- **intersectionPtr RelXmlGetIntersection** ()
- **complementPtr RelXmlGetComplement** ()
- **compoUnionPtr RelXmlGetComposition_Union** ()
- **compoInterPtr RelXmlGetComposition_Intersection** ()
- **conversePtr RelXmlGetConverse** ()
- **topPtr RelXmlGetTop** ()
- **bottomPtr RelXmlGetBottom** ()
- **identityPtr RelXmlGetIdentity** ()
- **relationPtr RelXmlGetRelation** ()
- void **relXmlDisplayComments** ()
- char ∗∗ **relXmlGetObjects** ()

Variables

- xmlDocPtr **xmlDocument**

4.8.1 Function Documentation

4.8.1.1 int RelMDDParseXmlFile (const char ∗ *xmlFilename*)

Function Function: **RelMDDParseXmlFile(const char ∗xmlFilename)** (p. **??**) This functions accepts a file and makes the content availble for all other functions. it is used for initialization .

Definition at line 56 of file RelxmlReader.c.

References xmlDocument.

Referenced by main().

```
{
    xmlDocPtr doc;
    doc = xmlParseFile(xmlFilename);
    if (doc == NULL) {
        return (0);

    }
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
        xmlFreeDoc(doc);
        return (0);
    }
    //xmlNode *root = NULL;
    //root = xmlDocGetRootElement(doc);
    xmlDocument = doc;
    return (1);

}/* RelMDDParseXmlFile*/
```

Here is the caller graph for this function:

4.8.1.2 int RelMDDValidateXmlFile (const char * *xmlPath*)

Function Function: **RelMDDValidateXmlFile(const char *xmlPath)** (p. **??**) this function calls the validations function to valid an xml file. It returns a non zero value is the file was validated successfully. otherwise returns zeor.

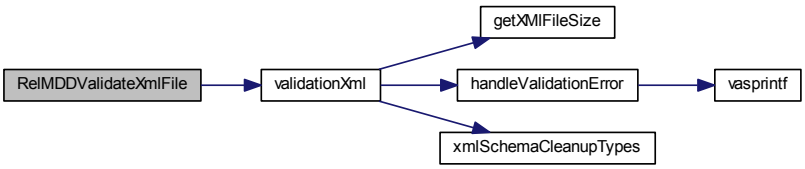
Definition at line 34 of file RelxmlReader.c.

References validationXml().

Referenced by main().

```
{
    LIBXML_TEST_VERSION
        int isvalid = validationXml("xmlschema.xsd", xmlPath);
    if (isvalid == 0) {
        printf("Validation successful so continue to load the file: ");
        return isvalid;
    } else {
        printf
            ("Validation is not successful please check you xml document
           against the schema ");
        return isvalid;
    }
}/* RelMDDValidateXmlFile*/
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.8.1.3 void relXmlDisplayComments ( )

Function Function: relXmlDisplayComment*s() print/displays to the comments from the comments tag.

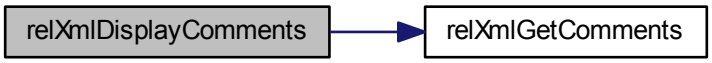
Definition at line 360 of file RelxmlReader.c.

References relXmlGetComments(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
```

```
    }
    root = xmlDocGetRootElement(doc);
    root = root->xmlChildrenNode;
    while (root != NULL) {
        if ((!xmlStrcmp(root->name, (const xmlChar *) "relation"))) {
            relXmlGetComments(doc, root);

        }

        root = root->next;
    }

}/* relXmlDisplayComments*/
```

Here is the call graph for this function:



4.8.1.4 bottomPtr RelXmlGetBottom ( )

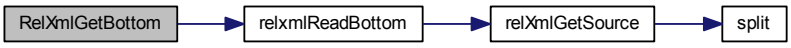
Function Function:**RelXmlGetBottom()** (p. **??**) Returns the contents of the bottom tag

Definition at line 277 of file RelxmlReader.c.

References relxmlReadBottom(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    bottomPtr ret = NULL;
    ret = (bottomPtr) malloc(sizeof(bottom));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(top));
    ret = (bottomPtr) relxmlReadBottom(root, doc);
    return ret;
}
```

Here is the call graph for this function:



4.8.1.5 complementPtr RelXmlGetComplement ( )

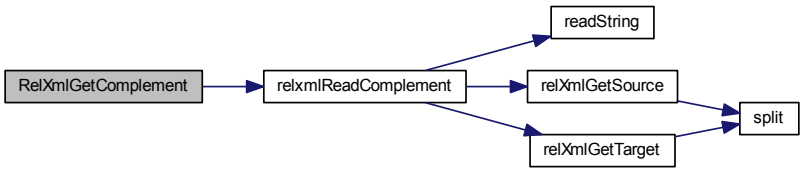
Function Function: **RelXmlGetComplement()** (p. **??**) Returns the contents of the complement tag

Definition at line 145 of file RelxmlReader.c.

References relxmlReadComplement(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    complementPtr ret = NULL;
    ret = (complementPtr) malloc(sizeof(complement));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(complement));
    ret = (complementPtr) relxmlReadComplement(root, doc);
    return ret;
}/* RelXmlGetComplement*/
```

Here is the call graph for this function:



4.8.1.6 compoInterPtr RelXmlGetComposition_Intersection ( )

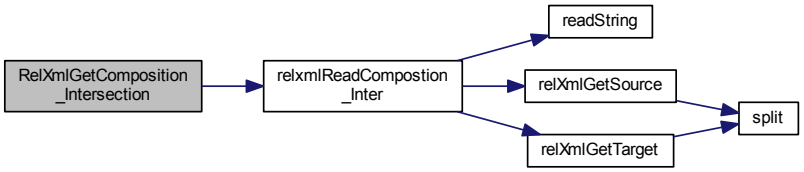
Function Function: **RelXmlGetComposition_Intersection()** (p. **??**) Returns the contents of the Composition_-Intersection tag

Definition at line 197 of file RelxmlReader.c.

References relxmlReadCompostion_Inter(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    compoInterPtr ret = NULL;
    ret = (compoInterPtr) malloc(sizeof(compoInter));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(compoInter));
    ret = (compoInterPtr) relxmlReadCompostion_Inter(root, doc);
    return ret;
}/*RelXmlGetComposition_Intersection*/
```

Here is the call graph for this function:



4.8.1.7 compoUnionPtr RelXmlGetComposition_Union ( )

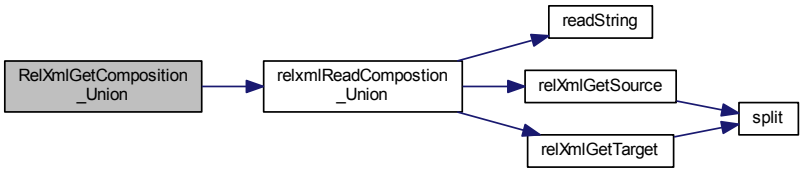
Function Function: **RelXmlGetComposition_Union()** (p. **??**) Returns the contents of the Composition_Union tag

Definition at line 171 of file RelxmlReader.c.

References relxmlReadCompostion_Union(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    compoUnionPtr ret = NULL;
    ret = (compoUnionPtr) malloc(sizeof(compoUnion));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(compoUnion));
    ret = (compoUnionPtr) relxmlReadCompostion_Union(root, doc);
    return ret;
}/*RelXmlGetComposition_Union*/
```

Here is the call graph for this function:



4.8.1.8 conversePtr RelXmlGetConverse ( )

Function Function: **RelXmlGetConverse()** (p. **??**) Returns the contents of the converse tag

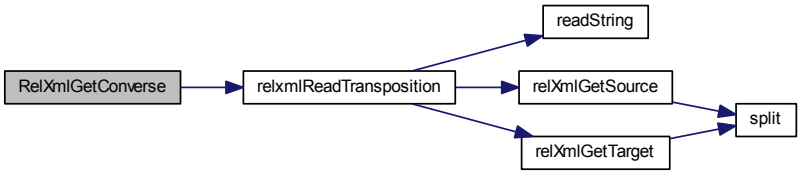
Definition at line 222 of file RelxmlReader.c.

References relxmlReadTransposition(), and xmlDocument.

```
{
```

```
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    conversePtr ret = NULL;
    ret = (conversePtr) malloc(sizeof(converse));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(converse));
    ret = (conversePtr) relxmlReadTransposition(root, doc);
    return ret;
}/* RelXmlGetConverse*/
```

Here is the call graph for this function:



4.8.1.9 identityPtr RelXmlGetIdentity ( )

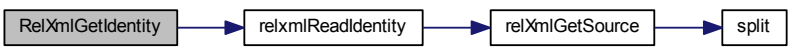
Function Function: RelXmlGetIdentity Returns the contents of the identity tag

Definition at line 305 of file RelxmlReader.c.

References relxmlReadIdentity(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    identityPtr ret = NULL;
    ret = (identityPtr) malloc(sizeof(identity));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(identity));
    ret = (identityPtr) relxmlReadIdentity(root, doc);
    return ret;
}/* RelXmlGetIdentity*/
```

Here is the call graph for this function:



4.8.1.10 intersectionPtr RelXmlGetIntersection ( )

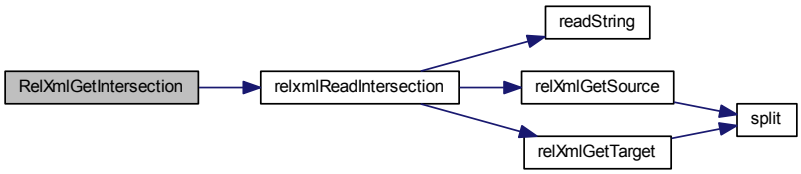
Function Function: **RelXmlGetIntersection()** (p. **??**) Returns the contents of the Intersection tag

Definition at line 119 of file RelxmlReader.c.

References relxmlReadIntersection(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    intersectionPtr ret = NULL;
    ret = (intersectionPtr) malloc(sizeof(intersection));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(intersection));
    ret = (intersectionPtr) relxmlReadIntersection(root, doc);
    return ret;
}/* RelXmlGetIntersection*/
```

Here is the call graph for this function:



4.8.1.11 char ** relXmlGetObjects ( )

Function Function: **relXmlGetObjects()** (p. **??**) returns the list of objects

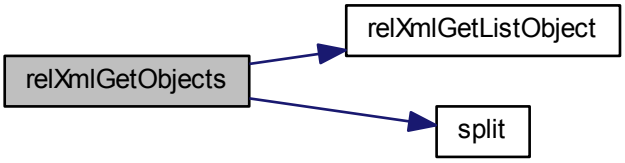
Definition at line 388 of file RelxmlReader.c.

References relXmlGetListObject(), split(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    root = root->xmlChildrenNode;
    xmlChar *string;
    char **tokens = NULL;
    while (root != NULL) {
        if ((!xmlStrcmp(root->name, (const xmlChar *) "relation"))) {
            string = (xmlChar *) relXmlGetListObject(doc, root);

            printf("list of objects from the object in basis file %s\n",
                   string);
            tokens = (char **) split((char *)string);
        }

        root = root->next;
    }
```

```
    return tokens;
}/*relXmlGetObjects*/
```

Here is the call graph for this function:



4.8.1.12 relationPtr RelXmlGetRelation ( )

Function Function: **RelXmlGetRelation()** (p. **??**) returns the contents from the relation tags

Definition at line 332 of file RelxmlReader.c.

References relxmlReadrelations(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    relationPtr ret = NULL;
    ret = (relationPtr) malloc(sizeof(relation));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(relation));
    ret = (relationPtr) relxmlReadrelations(root, doc);
    return ret;
}/*RelXmlGetRelation*/
```

Here is the call graph for this function:

4.8.1.13 topPtr RelXmlGetTop ( )

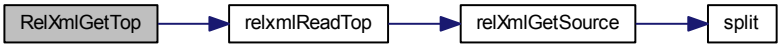
Function Function: **RelXmlGetTop()** (p. **??**) Returns the contents of the top tag

Definition at line 249 of file RelxmlReader.c.

References relxmlReadTop(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    topPtr ret = NULL;
    ret = (topPtr) malloc(sizeof(top));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(top));
    ret = (topPtr) relxmlReadTop(root, doc);
    return ret;
}
```

Here is the call graph for this function:



4.8.1.14 unionsPtr RelXmlGetUnion ( )

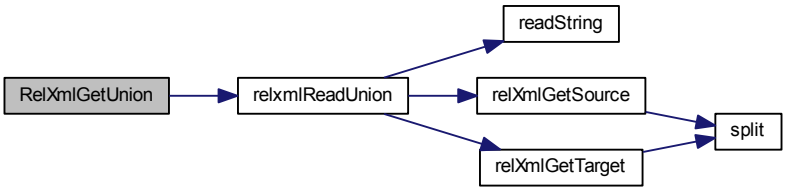
Function Function: **RelXmlGetUnion()** (p. **??**) Returns the contents of the union tag

Definition at line 83 of file RelxmlReader.c.

References relxmlReadUnion(), and xmlDocument.

```
{

    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    unionsPtr ret = NULL;
    ret = (unionsPtr) malloc(sizeof(unions));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(unions));
    ret = (unionsPtr) relxmlReadUnion(root, doc);

    /* for(kk=0;kk<ret->len;kk++){
       printf("\nstarting union here so please be read thank you God\n ");

       printf("  %s || \t%s \n\t..%d..", *ret[kk].source, *ret[kk].target,
       ret->len_Content[kk]);
       for(k=0;k<ret->len_Content[kk];k++)
       printf("  %f\n", ret[kk].content[k]);}

     */
    return ret;
}/* RelXmlGetUnion*/
```

Here is the call graph for this function:



4.8.2 Variable Documentation

4.8.2.1 xmlDocPtr xmlDocument

CFile This file contains all the functions for maniputions of the xml file content. In this file we imported several functions from Libxml2(fucntions are copy right of .... Copyright (C) 1998-2003 Daniel Veillard.All Rights Reserved.)

.....

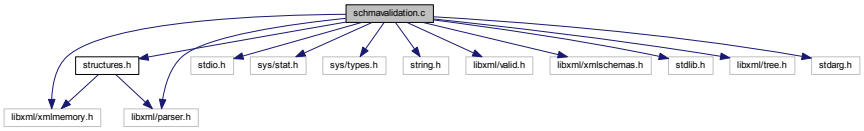
This functions accepts and xml file and extract the content, making it availble for the RelMDD system and is and contains all external functions for use by the user.

Definition at line 23 of file RelxmlReader.c.

4.9 schmavalidation.c File Reference

```
#include "structures.h"
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <string.h>
#include <libxml/parser.h>
#include <libxml/valid.h>
#include <libxml/xmlschemas.h>
#include <stdlib.h>
#include <libxml/xmlmemory.h>
#include <libxml/tree.h>
#include <stdarg.h>
```

Include dependency graph for schmavalidation.c:



Functions

- int **vasprintf** (char ∗∗ptr, const char ∗format, va_list arg)
- u_int32_t **getXMlFileSize** (const char ∗file_name)

- void **xmlSchemaCleanupTypes** (void)
- void **handleValidationError** (void ∗ctx, const char ∗format,...)
- int **validationXml** (const char ∗xsdPath, const char ∗xmlPath)

4.9.1 Function Documentation

4.9.1.1 u_int32_t getXMlFileSize (const char ∗ *file_name*)

Definition at line 30 of file schmavalidation.c.

Referenced by validationXml().

```
                                                {
    struct stat buf;
    if ( stat(file_name, &buf) != 0 ) return(0);
    return (unsigned int)buf.st_size;
}
```

Here is the caller graph for this function:

4.9.1.2 void handleValidationError (void ∗ *ctx*, const char ∗ *format*, ...)

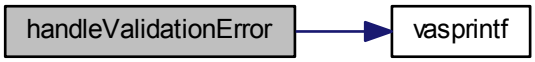
Definition at line 36 of file schmavalidation.c.

References vasprintf().

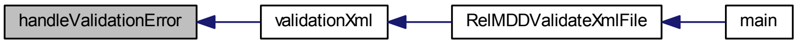
Referenced by validationXml().

```
                                                                {
    char *errMsg;
    va_list args;
    va_start(args, format);
    vasprintf(&errMsg, format, args);
    va_end(args);
    fprintf(stderr, "Validation Error: %s", errMsg);
    free(errMsg);
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.1.3 int validationXml (const char ∗ *xsdPath*, const char ∗ *xmlPath*)

Definition at line 46 of file schmavalidation.c.

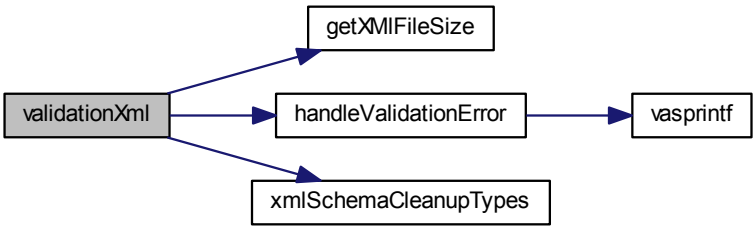
References getXMlFileSize(), handleValidationError(), and xmlSchemaCleanupTypes().

Referenced by RelMDDValidateXmlFile().

```
                                                                {
    printf("''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''");
    printf("\n");

    printf("XSD File: %s\n", xsdPath);
    printf("XML File: %s\n", xmlPath);

    int xmlLength = getXMlFileSize(xmlPath);
    char *xmlSource = (char *)malloc(sizeof(char) * xmlLength);

    FILE *p = fopen(xmlPath, "r");
    char c;
    unsigned int i = 0;
    while ((c = fgetc(p)) != EOF) {
        xmlSource[i++] = c;
    }
    printf("\n");

   // printf("XML Source:\n\n%s\n", xmlSource); \\ you can use this to print
       out the content of the xml file
    fclose(p);

    printf("\n");

    int result = 42;
    xmlSchemaParserCtxtPtr parserCtxt = NULL;
    xmlSchemaPtr schema = NULL;
    xmlSchemaValidCtxtPtr validCtxt = NULL;

    xmlDocPtr xmlDocumentPointer = xmlParseMemory(xmlSource, xmlLength);
    parserCtxt = xmlSchemaNewParserCtxt(xsdPath);

    if (parserCtxt == NULL) {
        fprintf(stderr, "Could not create XSD schema parsing context.\n");
        goto leave;
    }

    schema = xmlSchemaParse(parserCtxt);
    //xmlSchemaDump(stdout, schema);
    if (schema == NULL) {
        fprintf(stderr, "Could not parse XSD schema.\n");
        goto leave;
    }

    validCtxt = xmlSchemaNewValidCtxt(schema);

    if (!validCtxt) {
        fprintf(stderr, "Could not create XSD schema validation context.\n");
        goto leave;
    }

    xmlSetStructuredErrorFunc(NULL, NULL);
    xmlSetGenericErrorFunc(NULL, handleValidationError);
    xmlThrDefSetStructuredErrorFunc(NULL, NULL);
    xmlThrDefSetGenericErrorFunc(NULL, handleValidationError);

    result = xmlSchemaValidateDoc(validCtxt, xmlDocumentPointer);

leave:
```

```
    if (parserCtxt) {
        xmlSchemaFreeParserCtxt(parserCtxt);
    }

    if (schema) {
        xmlSchemaFree(schema);
    }

    if (validCtxt) {
        xmlSchemaFreeValidCtxt(validCtxt);
    }
    xmlSchemaCleanupTypes();
    xmlCleanupParser();
    xmlMemoryDump();
    printf("\n");
    printf("Validation successful: %s (result: %d)\n", (result == 0) ? "YES" :
      "NO", result);

    return result;
}
```

Here is the call graph for this function:



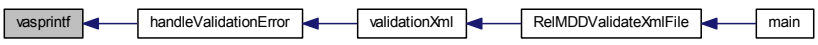
Here is the caller graph for this function:

4.9.1.4 int vasprintf (char ** *ptr*, const char * *format*, va_list *arg*)

CFile

Referenced by handleValidationError().

Here is the caller graph for this function:



4.9.1.5 void xmlSchemaCleanupTypes (void)

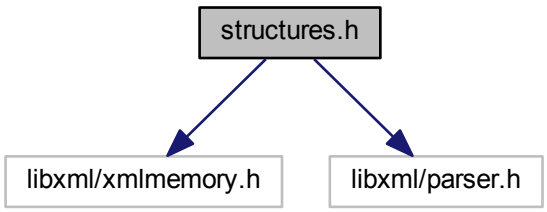
Referenced by validationXml().

Here is the caller graph for this function:

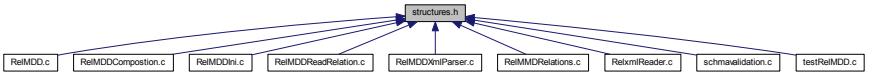
4.10 structures.h File Reference

```
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>
```

Include dependency graph for structures.h:



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



Data Structures

- struct **relation**
- struct **identity**
- struct **bottom**
- struct **top**
- struct **unions**
- struct **intersection**
- struct **compoUnion**
- struct **compoInter**
- struct **converse**
- struct **complement**

Macros

- #define **MAXA** 500

Typedefs

- typedef struct **relation relation**
- typedef struct **relation** ∗ **relationPtr**
- typedef struct **identity identity**
- typedef struct **identity** ∗ **identityPtr**
- typedef struct **bottom bottom**
- typedef struct **bottom** ∗ **bottomPtr**
- typedef struct **top top**
- typedef struct **top** ∗ **topPtr**
- typedef struct **unions unions**
- typedef struct **unions** ∗ **unionsPtr**
- typedef struct **intersection intersection**
- typedef struct **intersection** ∗ **intersectionPtr**
- typedef struct **compoUnion compoUnion**
- typedef struct **compoUnion** ∗ **compoUnionPtr**
- typedef struct **compoInter compoInter**
- typedef struct **compoInter** ∗ **compoInterPtr**
- typedef struct **converse converse**
- typedef struct **converse** ∗ **conversePtr**
- typedef struct **complement complement**
- typedef struct **complement** ∗ **complementPtr**

Functions

- int **RelMDDValidateXmlFile** (const char ∗xmlPath)
- int **RelMDDParseXmlFile** (const char ∗xmlFilename)
- **unionsPtr RelXmlGetUnion** ()
- **intersectionPtr RelXmlGetIntersection** ()
- **complementPtr RelXmlGetComplement** ()
- **compoUnionPtr RelXmlGetComposition_Union** ()
- **compoInterPtr RelXmlGetComposition_Intersection** ()
- **conversePtr RelXmlGetConverse** ()
- **topPtr RelXmlGetTop** ()
- **bottomPtr RelXmlGetBottom** ()
- **identityPtr RelXmlGetIdentity** ()
- **relationPtr RelXmlGetRelation** ()
- void **relXmlDisplayComments** ()
- char ∗∗ **relXmlGetObjects** ()
- **compoUnionPtr compoUionIni** ()
- **conversePtr coverseIni** ()
- **compoInterPtr compoInterIni** ()
- **complementPtr compIni** ()
- **intersectionPtr interIni** ()
- **unionsPtr unionIni** ()
- **compoInterPtr test** ()
- **identityPtr idet** ()
- **bottomPtr bott** ()
- **topPtr topp** ()
- int **validationXml** (const char ∗xsdPath, const char ∗xmlPath)

- **intersectionPtr relxmlReadIntersection** (xmlNode *root, xmlDocPtr doc)
- **compoUnionPtr relxmlReadCompostion_Union** (xmlNode *root, xmlDocPtr doc)
- **compoInterPtr relxmlReadCompostion_Inter** (xmlNode *root, xmlDocPtr doc)
- **conversePtr relxmlReadTransposition** (xmlNode *root, xmlDocPtr doc)
- **complementPtr relxmlReadComplement** (xmlNode *root, xmlDocPtr doc)
- **unionsPtr relxmlReadUnion** (xmlNode *root, xmlDocPtr doc)
- xmlChar * **relXmlGetComments** (xmlDocPtr doc, xmlNodePtr cur)
- xmlChar * **relXmlGetListObject** (xmlDocPtr doc, xmlNodePtr cur)
- **relationPtr relxmlReadrelations** (xmlNode *root, xmlDocPtr doc)
- char ** **relXmlGetSourceRelation** (xmlChar *pt, int i)
- char ** **relXmlGetTargetRelation** (xmlChar *pt, int i)
- char ** **relXmlGetNumberRelation** (xmlChar *pt, int i)
- **identityPtr relxmlReadIdentity** (xmlNode *root, xmlDocPtr doc)
- char ** **relXmlGetObjectIdentity** (xmlChar *pt, int i)
- double * **relXmlGetRelationIdentity** (xmlChar *pt, int i)
- **topPtr relxmlReadTop** (xmlNode *root, xmlDocPtr doc)
- **bottomPtr relxmlReadBottom** (xmlNode *root, xmlDocPtr doc)
- xmlChar * **relXmlGetTopSource** (xmlChar *pt, int i)
- xmlChar * **relXmlGetTopTarget** (xmlChar *pt, int i)
- xmlChar * **relXmlGetTopRelation** (xmlChar *pt, int i)
- char ** **relXmlGetSource** (xmlChar *pt, int i)
- char ** **relXmlGetTarget** (xmlChar *pt, int i)
- char ** **split** (char *string)

Variables

- xmlDocPtr **xmlDocument**
- const char * **xmlfile**
- double * **mat1**
- double * **mat2**
- char ** **source_comp**
- char ** **target_Comp**
- int **row_comp**
- int **col_comp**
- double * **element_comp**
- int * **row_trackComp**
- int * **col_trackComp**

4.10.1 Macro Definition Documentation

4.10.1.1 #define MAXA 500

Definition at line 4 of file structures.h.

4.10.2 Typedef Documentation

4.10.2.1 typedef struct bottom bottom

4.10.2.2 typedef struct bottom * bottomPtr

4.10.2.3 typedef struct complement complement

4.10.2.4 typedef struct complement * complementPtr

4.10.2.5 typedef struct compoInter compoInter

4.10.2.6 typedef struct compoInter * compoInterPtr

4.10.2.7 typedef struct compoUnion compoUnion

4.10.2.8 typedef struct compoUnion * compoUnionPtr

4.10.2.9 typedef struct converse converse

4.10.2.10 typedef struct converse * conversePtr

4.10.2.11 typedef struct identity identity

4.10.2.12 typedef struct identity * identityPtr

4.10.2.13 typedef struct intersection intersection

4.10.2.14 typedef struct intersection * intersectionPtr

4.10.2.15 typedef struct relation relation

4.10.2.16 typedef struct relation * relationPtr

4.10.2.17 typedef struct top top

4.10.2.18 typedef struct top * topPtr

4.10.2.19 typedef struct unions unions

4.10.2.20 typedef struct unions * unionsPtr

4.10.3 Function Documentation

4.10.3.1 bottomPtr bott ( )

Definition at line 1207 of file RelMDDXmlParser.c.

References relxmlReadBottom(), and xmlDocument.

```
{

   // xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    bottomPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadBottom(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the call graph for this function:

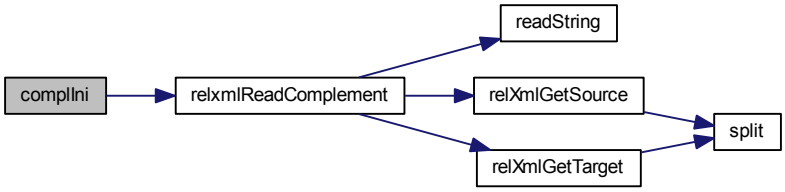
4.10.3.2 complementPtr complIni ( )

Definition at line 1086 of file RelMDDXmlParser.c.

References relxmlReadComplement(), and xmlDocument.

```
{

    //xmlDocPtr doc;
    //doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    complementPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadComplement(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.....................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the call graph for this function:



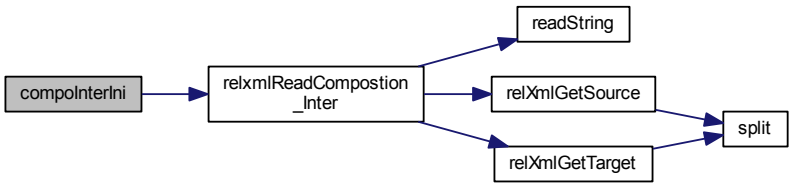
4.10.3.3 compoInterPtr compoInterIni ( )

Definition at line 1303 of file RelMDDXmlParser.c.

References relxmlReadCompostion_Inter(), and xmlDocument.

```
                                {

//xmlDocPtr doc;
 // doc = xmlParseFile("w3.xml");

  if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL ;
  root = xmlDocGetRootElement(xmlDocument);



   compoInterPtr ret;


   // memset(ret, 0, sizeof(identityPtr));
 ret=  relxmlReadCompostion_Inter(  root,xmlDocument);
  //int k,kk;
// printf("\n.................. in main.................................\n ");

 //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}
 if( !root ||
      !root->name ||
      xmlStrcmp(root->name, (xmlChar *)"RelationBasis") )
  {
     xmlFreeDoc(xmlDocument);
     return NULL;
  }

return ret;
 }
```

Here is the call graph for this function:



4.10.3.4 compoUnionPtr compoUnionIni ( )

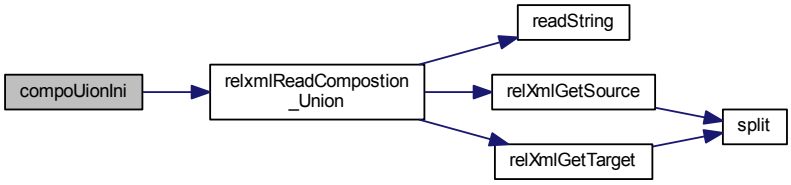
Definition at line 1264 of file RelMDDXmlParser.c.

References relxmlReadCompostion_Union(), and xmlDocument.

```
                                {

//xmlDocPtr doc;
  //doc = xmlParseFile("w3.xml");

  if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL ;
  root = xmlDocGetRootElement(xmlDocument);



   compoUnionPtr ret;


   // memset(ret, 0, sizeof(identityPtr));
 ret=  relxmlReadCompostion_Union(  root,xmlDocument);
```

```
  //int k,kk;
// printf("\n.................. in main.....................................\n ");

 //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}
 if( !root ||
      !root->name ||
      xmlStrcmp(root->name,(xmlChar *)"RelationBasis") )
  {
     xmlFreeDoc(xmlDocument);
     return NULL;
  }

return ret;
 }
```

Here is the call graph for this function:



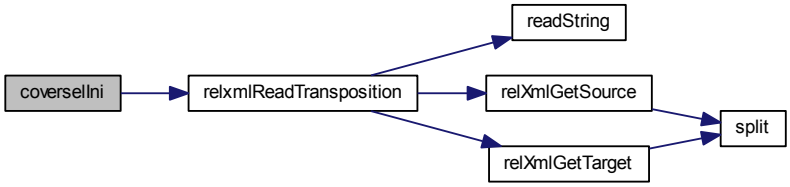
4.10.3.5 conversePtr coversellni ( )

Definition at line 1058 of file RelMDDXmlParser.c.

References relxmlReadTransposition(), and xmlDocument.

```
{

    //xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument== NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    conversePtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadTransposition(root,xmlDocument);
    //int k,kk;
// printf("\n.................. in main.....................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the call graph for this function:



4.10.3.6 identityPtr idet ( )

xmlDocPtr doc;

Definition at line 1179 of file RelMDDXmlParser.c.

References relxmlReadIdentity(), and xmlDocument.

```
{

    //doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
    root = xmlDocGetRootElement(xmlDocument);

    identityPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadIdentity(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.....................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;
}
```

Here is the call graph for this function:

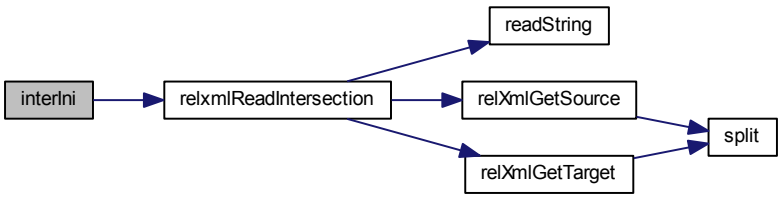
4.10.3.7 intersectionPtr interIni ( )

Definition at line 1115 of file RelMDDXmlParser.c.

References relxmlReadIntersection(), and xmlDocument.

```
{

  //  xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL;
    root = xmlDocGetRootElement(xmlDocument);

    intersectionPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadIntersection(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}
    if (!root || !root->name || xmlStrcmp(root->name, (xmlChar *)"RelationBasis
      ")) {
        xmlFreeDoc(xmlDocument);
        return NULL;
    }

    return ret;
}
```

Here is the call graph for this function:



4.10.3.8 int RelMDDParseXmlFile (const char * *xmlFilename*)

Function Function: **RelMDDParseXmlFile(const char *xmlFilename)** (p. **??**) This functions accepts a file and makes the content availble for all other functions. it is used for initialization .

Definition at line 56 of file RelxmlReader.c.

References xmlDocument.

Referenced by main().

```
{
    xmlDocPtr doc;
    doc = xmlParseFile(xmlFilename);
    if (doc == NULL) {
        return (0);

    }
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
        xmlFreeDoc(doc);
        return (0);
    }
    //xmlNode *root = NULL;
    //root = xmlDocGetRootElement(doc);
    xmlDocument = doc;
```

```
    return (1);
}/* RelMDDParseXmlFile*/
```

Here is the caller graph for this function:

4.10.3.9 int RelMDDValidateXmlFile (const char * *xmlPath*)

Function Function: **RelMDDValidateXmlFile(const char *xmlPath)** (p. **??**) this function calls the validations function to valid an xml file. It returns a non zero value is the file was validated successfully. otherwise returns zeor.

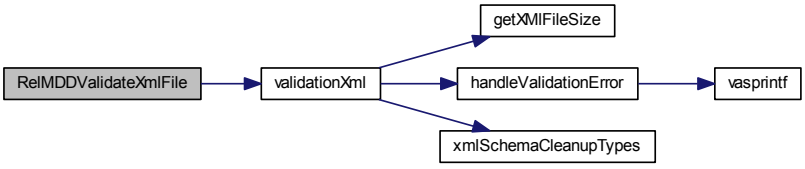
Definition at line 34 of file RelxmlReader.c.

References validationXml().

Referenced by main().

```
{
    LIBXML_TEST_VERSION
        int isvalid = validationXml("xmlschema.xsd", xmlPath);
    if (isvalid == 0) {
        printf("Validation successful so continue to load the file: ");
        return isvalid;
    } else {
        printf
            ("Validation is not successful please check you xml document
           against the schema ");
        return isvalid;
    }
}/* RelMDDValidateXmlFile*/
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.10.3.10 void relXmlDisplayComments ( )

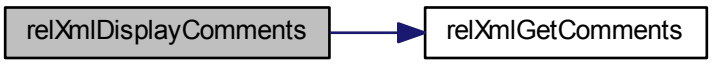
Function Function: relXmlDisplayComment∗s() print/displays to the comments from the comments tag.

Definition at line 360 of file RelxmlReader.c.

References relXmlGetComments(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    root = root->xmlChildrenNode;
    while (root != NULL) {
        if ((!xmlStrcmp(root->name, (const xmlChar *) "relation"))) {
            relXmlGetComments(doc, root);

        }

        root = root->next;
    }

}/* relXmlDisplayComments*/
```

Here is the call graph for this function:



4.10.3.11 bottomPtr RelXmlGetBottom ( )

Function Function:**RelXmlGetBottom()** (p. **??**) Returns the contents of the bottom tag

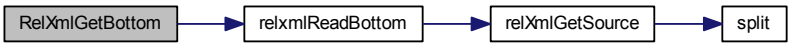
Definition at line 277 of file RelxmlReader.c.

References relxmlReadBottom(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
```

```
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    bottomPtr ret = NULL;
    ret = (bottomPtr) malloc(sizeof(bottom));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(top));
    ret = (bottomPtr) relxmlReadBottom(root, doc);
    return ret;
}
```

Here is the call graph for this function:



4.10.3.12 xmlChar* relXmlGetComments (xmlDocPtr *doc*, xmlNodePtr *cur*)

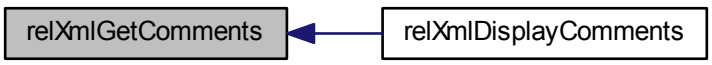
Function Function: **relXmlGetComments(xmlDocPtr doc, xmlNodePtr cur)** (p. **??**) This function reads the comments from the comments tag and return it ..

Definition at line 50 of file RelMDDXmlParser.c.

Referenced by relXmlDisplayComments().

```
{

    xmlChar *comments =NULL;

    cur = cur->xmlChildrenNode;
    while (cur != NULL) {
        if ((!xmlStrcmp(cur->name, (const xmlChar *) "comment"))) {

            comments = xmlNodeListGetString(doc, cur->xmlChildrenNode, 1);
            printf("The comments on this basis file:\n%s ", comments);

        }
        cur = cur->next;

    }

    return (comments);
}
```

Here is the caller graph for this function:



4.10.3.13 complementPtr RelXmlGetComplement ( )

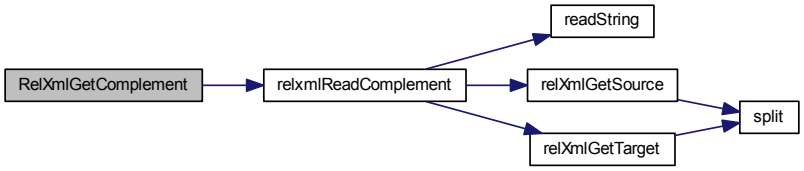
Function Function: **RelXmlGetComplement()** (p. **??**) Returns the contents of the complement tag

Definition at line 145 of file RelxmlReader.c.

References relxmlReadComplement(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    complementPtr ret = NULL;
    ret = (complementPtr) malloc(sizeof(complement));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(complement));
    ret = (complementPtr) relxmlReadComplement(root, doc);
    return ret;
}/* RelXmlGetComplement*/
```

Here is the call graph for this function:



4.10.3.14 compoInterPtr RelXmlGetComposition_Intersection ( )

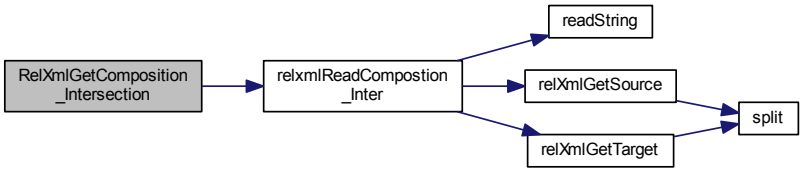
Function Function: **RelXmlGetComposition_Intersection()** (p. **??**) Returns the contents of the Composition_-Intersection tag

Definition at line 197 of file RelxmlReader.c.

References relxmlReadCompostion_Inter(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    compoInterPtr ret = NULL;
    ret = (compoInterPtr) malloc(sizeof(compoInter));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(compoInter));
    ret = (compoInterPtr) relxmlReadCompostion_Inter(root, doc);
    return ret;
}/*RelXmlGetComposition_Intersection*/
```

Here is the call graph for this function:



4.10.3.15 compoUnionPtr RelXmlGetComposition_Union ( )

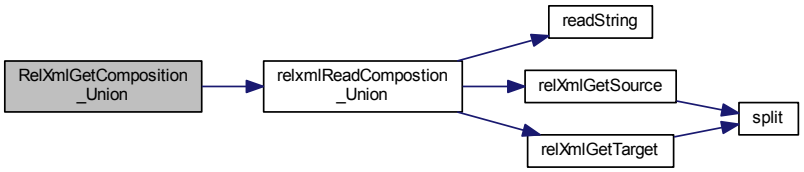
Function Function: **RelXmlGetComposition_Union()** (p. **??**) Returns the contents of the Composition_Union tag

Definition at line 171 of file RelxmlReader.c.

References relxmlReadCompostion_Union(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    compoUnionPtr ret = NULL;
    ret = (compoUnionPtr) malloc(sizeof(compoUnion));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(compoUnion));
    ret = (compoUnionPtr) relxmlReadCompostion_Union(root, doc);
    return ret;
}/*RelXmlGetComposition_Union*/
```

Here is the call graph for this function:



4.10.3.16 conversePtr RelXmlGetConverse ( )

Function Function: **RelXmlGetConverse()** (p. **??**) Returns the contents of the converse tag

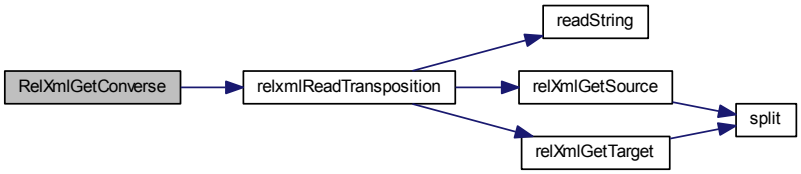
Definition at line 222 of file RelxmlReader.c.

References relxmlReadTransposition(), and xmlDocument.

```
{
```

```
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    conversePtr ret = NULL;
    ret = (conversePtr) malloc(sizeof(converse));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(converse));
    ret = (conversePtr) relxmlReadTransposition(root, doc);
    return ret;
}/* RelXmlGetConverse*/
```

Here is the call graph for this function:



4.10.3.17 identityPtr RelXmlGetIdentity ( )

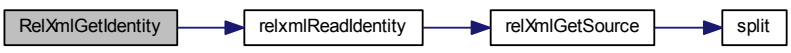
Function Function: RelXmlGetIdentity Returns the contents of the identity tag

Definition at line 305 of file RelxmlReader.c.

References relxmlReadIdentity(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    identityPtr ret = NULL;
    ret = (identityPtr) malloc(sizeof(identity));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(identity));
    ret = (identityPtr) relxmlReadIdentity(root, doc);
    return ret;
}/* RelXmlGetIdentity*/
```

Here is the call graph for this function:



4.10.3.18 intersectionPtr RelXmlGetIntersection ( )

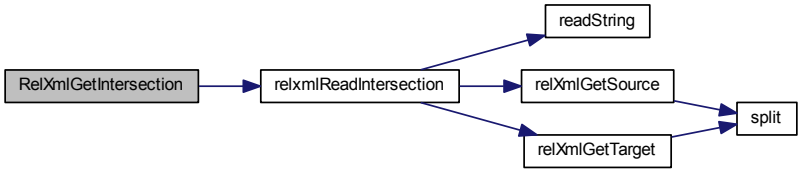
Function Function: **RelXmlGetIntersection()** (p. **??**) Returns the contents of the Intersection tag

Definition at line 119 of file RelxmlReader.c.

References relxmlReadIntersection(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    intersectionPtr ret = NULL;
    ret = (intersectionPtr) malloc(sizeof(intersection));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(intersection));
    ret = (intersectionPtr) relxmlReadIntersection(root, doc);
    return ret;
}/* RelXmlGetIntersection*/
```

Here is the call graph for this function:



4.10.3.19 xmlChar* relXmlGetListObject (xmlDocPtr *doc*, xmlNodePtr *cur*)

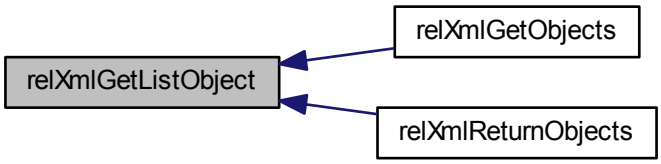
Function Function: **relXmlGetListObject(xmlDocPtr doc, xmlNodePtr cur)** (p. **??**) This fucntion returns the string objects from the xmlfile A,B,C.. unformated

Definition at line 79 of file RelMDDXmlParser.c.

Referenced by relXmlGetObjects(), and relXmlReturnObjects().

```
{

    xmlChar *object =NULL;

    cur = cur->xmlChildrenNode;
    while (cur != NULL) {
        if ((!xmlStrcmp(cur->name, (const xmlChar *) "objects"))) {

            object = xmlNodeListGetString(doc, cur->xmlChildrenNode, 1);

        }
        cur = cur->next;

    }

    return (object);
}
```

Here is the caller graph for this function:



4.10.3.20 char** relXmlGetNumberRelation (xmlChar * *pt*, int *i*)

Function Function: relXmlGetNumberRelation(xmlDocPtr doc, xmlNodePtr cur) returns list of number of relatinos from the relations tag , you must free the return value after use:

Definition at line 255 of file RelMDDXmlParser.c.

References split().

```
{   //char *string
    char **number;
    number = split((char *)pt);

    for (i = 0; number[i] != NULL; i++)

        printf("number %s\n", number[i]);

    return number;
}
```

Here is the call graph for this function:

4.10.3.21 char** relXmlGetObjectIdentity (xmlChar * *pt*, int *i*)

4.10.3.22 char** relXmlGetObjects ( )

Function Function: **relXmlGetObjects()** (p. **??**) returns the list of objects

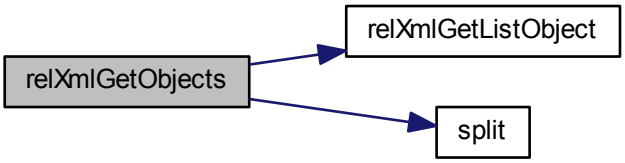
Definition at line 388 of file RelxmlReader.c.

References relXmlGetListObject(), split(), and xmlDocument.

```
{
    xmlDocPtr doc;
```

```
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    root = root->xmlChildrenNode;
    xmlChar *string;
    char **tokens = NULL;
    while (root != NULL) {
        if ((!xmlStrcmp(root->name, (const xmlChar *) "relation"))) {
            string = (xmlChar *) relXmlGetListObject(doc, root);

            printf("list of objects from the object in basis file %s\n",
                   string);
            tokens = (char **) split((char *)string);
        }

        root = root->next;
    }

    return tokens;

}/*relXmlGetObjects*/
```

Here is the call graph for this function:



4.10.3.23 relationPtr RelXmlGetRelation ( )

Function Function: **RelXmlGetRelation()** (p. **??**) returns the contents from the relation tags

Definition at line 332 of file RelxmlReader.c.

References relxmlReadrelations(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    relationPtr ret = NULL;
    ret = (relationPtr) malloc(sizeof(relation));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(relation));
    ret = (relationPtr) relxmlReadrelations(root, doc);
    return ret;
}/*RelXmlGetRelation*/
```

Here is the call graph for this function:

4.10.3.24 double* relXmlGetRelationIdentity (xmlChar * *pt,* int *i*)

4.10.3.25 char** relXmlGetSource (xmlChar * *pt,* int *i*)

Function Function: **relXmlGetSource(xmlChar * pt, int i)** (p. **??**) get the object attribute of

Definition at line 336 of file RelMDDXmlParser.c.

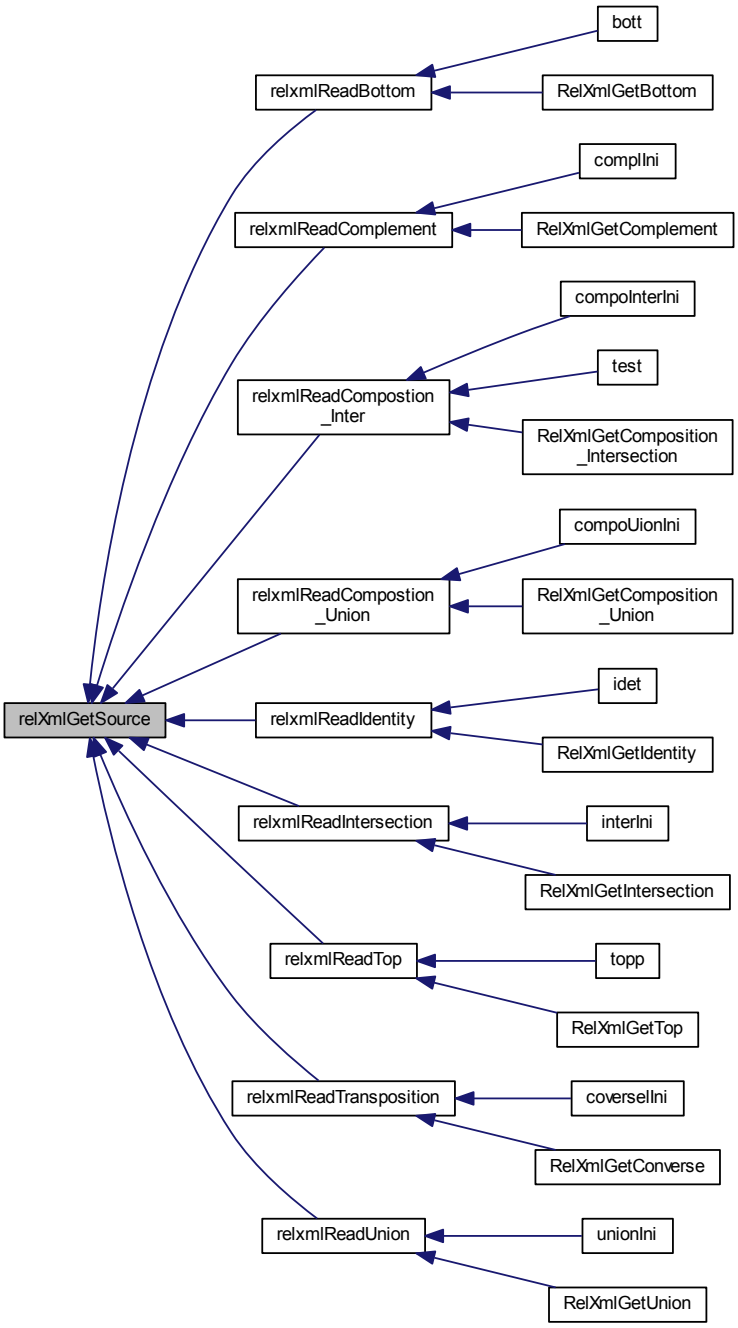
References split().

Referenced by relxmlReadBottom(), relxmlReadComplement(), relxmlReadCompostion_Inter(), relxmlReadCompostion_Union(), relxmlReadIdentity(), relxmlReadIntersection(), relxmlReadTop(), relxmlReadTransposition(), and relxmlReadUnion().

```
{
    char **tokens;
    tokens = split((char *)pt);
    return tokens;
}/*End of relxmlGetSource*/
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.10.3.26 char** relXmlGetSourceRelation (xmlChar * *pt,* int *i*)

4.10.3.27 char** relXmlGetTarget (xmlChar * *pt,* int *i*)

Definition at line 588 of file RelMDDXmlParser.c.

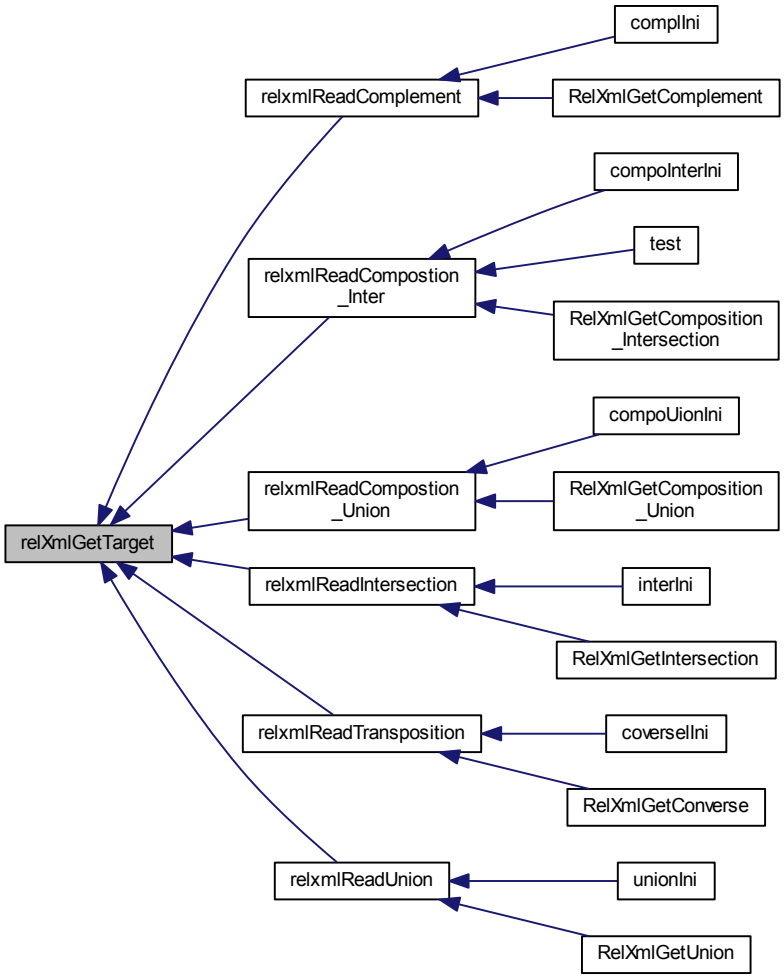
References split().

Referenced by relxmlReadComplement(), relxmlReadCompostion_Inter(), relxmlReadCompostion_Union(), relxmlReadIntersection(), relxmlReadTransposition(), and relxmlReadUnion().

```
{
    char **tokens;
    tokens = split((char *)pt);
    return tokens;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.10.3.28 char** relXmlGetTargetRelation (xmlChar * *pt*, int *i*)

4.10.3.29 topPtr RelXmlGetTop ( )

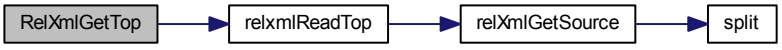
Function Function: **RelXmlGetTop()** (p. **??**) Returns the contents of the top tag

Definition at line 249 of file RelxmlReader.c.

References relxmlReadTop(), and xmlDocument.

```
{
    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    topPtr ret = NULL;
    ret = (topPtr) malloc(sizeof(top));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(top));
    ret = (topPtr) relxmlReadTop(root, doc);
    return ret;
}
```

Here is the call graph for this function:



4.10.3.30 xmlChar* relXmlGetTopRelation (xmlChar * *pt*, int *i*)

4.10.3.31 xmlChar* relXmlGetTopSource (xmlChar * *pt*, int *i*)

4.10.3.32 xmlChar* relXmlGetTopTarget (xmlChar * *pt*, int *i*)

4.10.3.33 unionsPtr RelXmlGetUnion ( )

Function Function: **RelXmlGetUnion()** (p. **??**) Returns the contents of the union tag

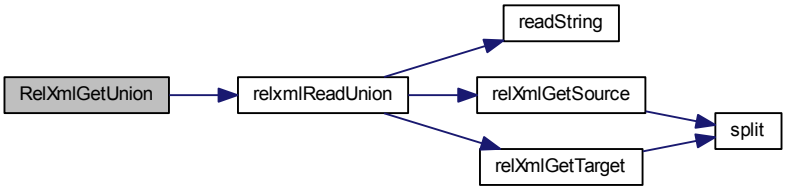
Definition at line 83 of file RelxmlReader.c.

References relxmlReadUnion(), and xmlDocument.

```
{

    xmlDocPtr doc;
    xmlNode *root = NULL;
    doc = xmlDocument;
    if (doc == NULL) {
        printf("error: could not parse file file.xml\n");
    }
    root = xmlDocGetRootElement(doc);
    unionsPtr ret = NULL;
    ret = (unionsPtr) malloc(sizeof(unions));
    if (ret == NULL) {
        fprintf(stderr, "out of memory\n");
        return (NULL);
    }
    memset(ret, 0, sizeof(unions));
```

```
    ret = (unionsPtr) relxmlReadUnion(root, doc);

    /* for(kk=0;kk<ret->len;kk++){
       printf("\nstarting union here so please be read thank you God\n ");

       printf("  %s || \t%s \n\t..%d..", *ret[kk].source, *ret[kk].target,
       ret->len_Content[kk]);
       for(k=0;k<ret->len_Content[kk];k++)
       printf("  %f\n", ret[kk].content[k]);}

     */
    return ret;
}/* RelXmlGetUnion*/
```

Here is the call graph for this function:



4.10.3.34 bottomPtr relxmlReadBottom (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: **relxmlReadBottom(xmlNode * root, xmlDocPtr doc)** (p. **??**) read the bottom relation or tranverse the bottom relations

Definition at line 350 of file RelMDDXmlParser.c.

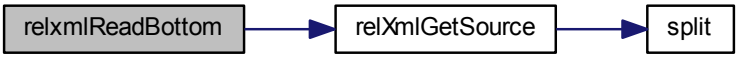
References bottom::len, bottom::rel, relXmlGetSource(), bottom::source, and bottom::target.

Referenced by bott(), and RelXmlGetBottom().

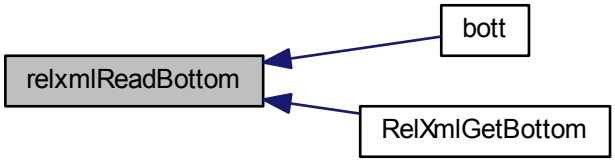
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *sourceAtrr, *targetAtrr, *numberAtrr;
    int i;
    bottomPtr ret;
    ret = (bottomPtr) malloc(sizeof(bottom) * 100);//increase this if needed
    ret->rel = (double *) malloc(sizeof(double) * 100);// increase this if
       needed
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "bottom")) {

                    //get the object attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                        // ret->source[i] =sourceAtrr;
                    }
                    // get the relations attribute of the identity.
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
```

```
                    if (targetAtrr) {
                        //relXmlGetBottomTarget(targetAtrr,i);
                        //ret->target[i] =targetAtrr;
                        ret[i].target = relXmlGetSource(targetAtrr, i);
                    }

                    numberAtrr =
                        xmlGetProp(child_node,
                                   (const xmlChar *) "relation");
                    if (numberAtrr) {
                        ret->rel[i] = atof((char *)numberAtrr);
                        ret->len = i + 1;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(numberAtrr);
                    i++;

                }

            }

        }

    }

   /* int j;
    for (j = 0; j < ret->len; j++)

        printf("bottom \t%s\t%s\t %f\n", *ret[j].source, *ret[j].target,
               ret->rel[j]);*/

    return ret;
}/*End of relxmlReadBottom */
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.35 complementPtr relxmlReadComplement (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: **relxmlReadComplement(xmlNode * root, xmlDocPtr doc)** (p. **??**) This functions returns the Complement data from the xml file ....

Definition at line 941 of file RelMDDXmlParser.c.

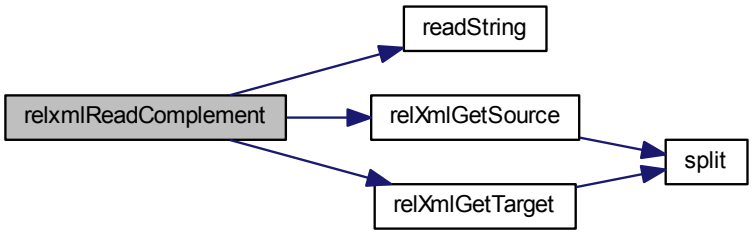
References complement::content, complement::len, complement::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), complement::source, and complement::target.

Referenced by complIni(), and RelXmlGetComplement().

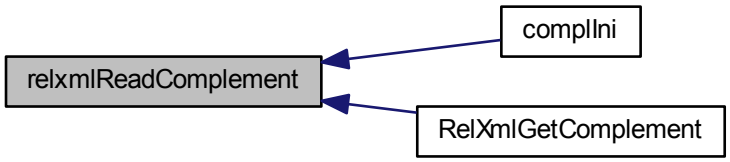
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    complement *ret = NULL;
    ret = (complement *) malloc(sizeof(complement) * 100);
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
    }
    ret->content = (double *) malloc(sizeof(double) * 100);
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "complement")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
       ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len = i + 1;

                    }
                    content_union = xmlNodeGetContent(child_node);

                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
      printf("cfvffd....%d\n.....................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }
    /* int k,kk;

       for(kk=0;kk<ret->len;kk++){
       printf("\nstarting union here so please be read thank you God\n ");

       printf("  %s || \t%s \n\t..%d..", *ret[kk].source, *ret[kk].target,
       ret->len_Content[kk]);
       for(k=0;k<ret->len_Content[kk];k++)
```

```
       printf("  %f\n", ret[kk].content[k]);} */
    xmlCleanupParser();

    return ret;
}/*End of complement*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.36 compoInterPtr relxmlReadCompostion_Inter (xmlNode * *root,* xmlDocPtr *doc*)

Function Function: **relxmlReadCompostion_Inter(xmlNode * root, xmlDocPtr doc)** (p. **??**) This functions returns the Compostion_Intersection data from the xml file ....

Definition at line 773 of file RelMDDXmlParser.c.

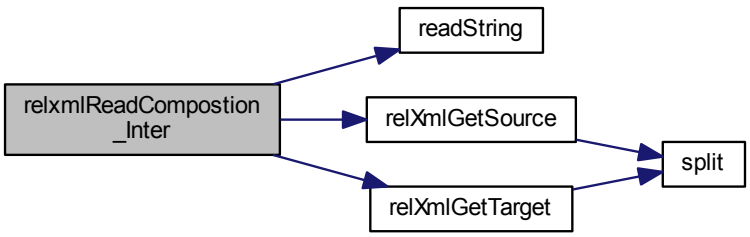
References compoInter::content, compoInter::len, compoInter::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), compoInter::source, and compoInter::target.

Referenced by compoInterIni(), RelXmlGetComposition_Intersection(), and test().

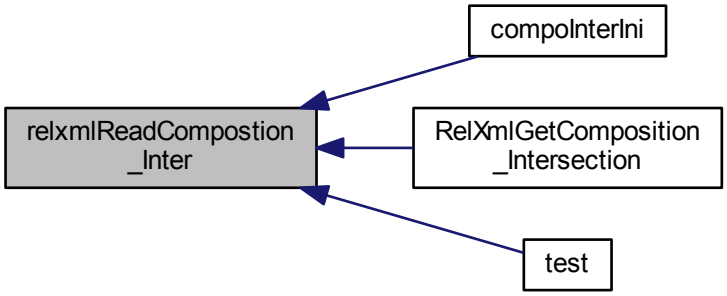
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    compoInter *ret = NULL;
    ret = (compoInter *) malloc(sizeof(compoInter) * 100);
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
    }
    ret->content = (double *) malloc(sizeof(double) * 100);
```

```
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name, (const xmlChar *)
                               "composition_Intersection")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
       ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len = i + 1;
                    }
                    content_union = xmlNodeGetContent(child_node);

                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
      printf("cfvffd....%d\n.............................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }

    xmlCleanupParser();

    return ret;
}/*End of relxmlReadCompostion_Inter*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.37 compoUnionPtr relxmlReadCompostion_Union (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: relxmlReadCompostion_Union(xmlDocPtr doc, xmlNodePtr cur) This functions returns the Compostion_Union data from the xml file ....

Definition at line 692 of file RelMDDXmlParser.c.

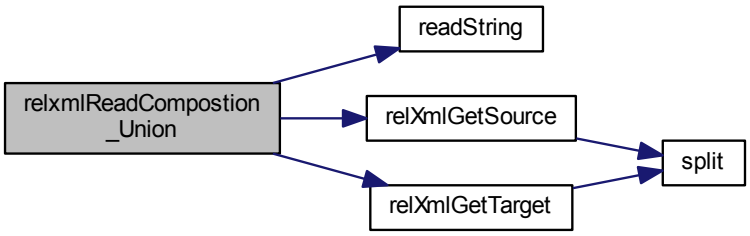
References compoUnion::content, compoUnion::len, compoUnion::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), compoUnion::source, and compoUnion::target.

Referenced by compoUionIni(), and RelXmlGetComposition_Union().

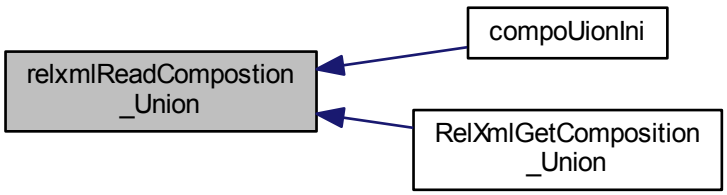
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    compoUnion *ret = NULL;
    ret = (compoUnion *) malloc(sizeof(compoUnion) * 100);
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
    }
    ret->content = (double *) malloc(sizeof(double) * 100);
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "composition_Union")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
       ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
```

```
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len=i+1;
                    }
                    content_union = xmlNodeGetContent(child_node);

                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
      printf("cfvffd....%d\n.....................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }
    xmlCleanupParser();

    return ret;
}/*End of relxmlReadCompostion_Union*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.38 identityPtr relxmlReadIdentity (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: relXmlReadIdentity(xmlDocPtr doc, xmlNodePtr cur) returns the identity list from the identity tag: to get the identity relation

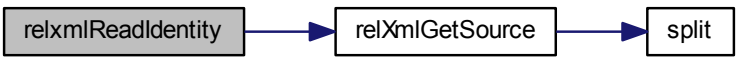
Definition at line 275 of file RelMDDXmlParser.c.

References identity::len, identity::object, identity::rel, and relXmlGetSource().

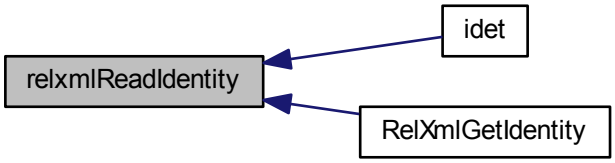
Referenced by idet(), and RelXmlGetIdentity().

```
{

    xmlNode *cur_node, *child_node;
    xmlChar *sourceAtrr, *targetAtrr;
    int i;
    identityPtr ret;
    ret = (identityPtr) malloc(sizeof(identity) * 100); // increase this if you
       are dealing with large data set
    ret->rel = (double *) malloc(sizeof(double) * 100);
    ret->object = (char **) malloc(sizeof(char));
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "identity")) {

                    //get the object attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "object");
                    if (sourceAtrr) {

                        ret[i].object = relXmlGetSource(sourceAtrr, i);
                        ret->len = i + 1;
                    }
                    // get the relations attribute of the identity.
                    targetAtrr =
                        xmlGetProp(child_node,
                                   (const xmlChar *) "relation");
                    if (targetAtrr) {
                        ret->rel[i] = atof((char *)targetAtrr);

                    }
                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    i++;
                }

            }

        }
    }
    //int j;
    // for(j = 0; j<ret->len; j++)

//printf("ok \t%s\n",*ret[j].object);
    return ret;
}/*End of  relxmlReadIdentity */
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.39 intersectionPtr relxmlReadIntersection (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: **relxmlReadIntersection(xmlNode * root, xmlDocPtr doc)** (p. **??**) This functions returns the intersection data from the xml file ....

Definition at line 602 of file RelMDDXmlParser.c.

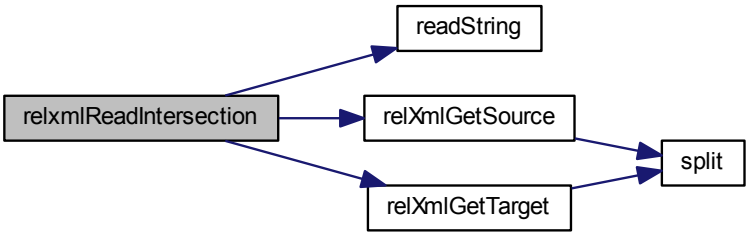
References intersection::content, intersection::len, intersection::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), intersection::source, and intersection::target.

Referenced by interIni(), and RelXmlGetIntersection().

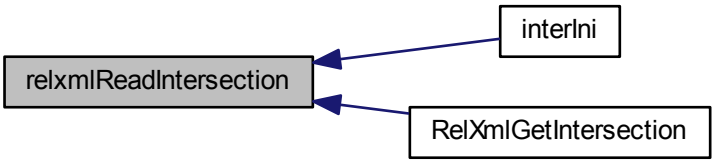
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    intersection *ret = NULL;
    ret = (intersection *) malloc(sizeof(intersection) * 100);
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
    }
    ret->content = (double *) malloc(sizeof(double) * 100);
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "intersection")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
       ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len = i + 1;
                    }
                    content_union = xmlNodeGetContent(child_node);
```

```
                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
      printf("cfvffd....%d\n.....................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;

                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }
    /* int k,kk;
       printf("\n.................intersection
       ..................................\n ");

       for(kk=0;kk<ret->len;kk++){

       printf("  %s || \t%s \n", *ret[kk].source, *ret[kk].target );
       for(k=0;k<ret->len_Content[kk];k++)
       printf("  %f\n", ret[kk].content[k] );} */
    xmlCleanupParser();

    return ret;
}/*End of relxmlReadIntersection*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.40 relationPtr relxmlReadrelations (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: **relxmlReadrelations(xmlNode * root, xmlDocPtr doc)** (p. **??**) This fucntion extract the content of the relation tag

Definition at line 190 of file RelMDDXmlParser.c.

References relation::number, relation::source, and relation::target.

Referenced by RelXmlGetRelation().

```
{

    xmlNode *cur_node, *child_node;
    xmlChar *sourceAtrr, *targetAtrr, *numberAtrr;
    int i;
    relationPtr ret;
    ret = (relationPtr) malloc(sizeof(relationPtr));
    ret->number = (int *) malloc(sizeof(int) * 100);// modify this if the
       number of objects are more than 100
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "relations")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {

                        ret->source[i] = (char *)sourceAtrr;
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
       ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {

                        ret->target[i] = (char *)sourceAtrr;
                    }
                    numberAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "number");
                    if (numberAtrr) {

                        ret->number[i] = atoi((char*)numberAtrr);

                    }
                    i++;

                }

            }

        }
    }
//xmlCleanupParser();
    return ret;
}
```

Here is the caller graph for this function:

4.10.3.41 topPtr relxmlReadTop (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: topPtr **relxmlReadTop(xmlNode * root, xmlDocPtr doc)** (p. **??**) This fucntion reads the top relation or tranverse the top relations

Definition at line 428 of file RelMDDXmlParser.c.

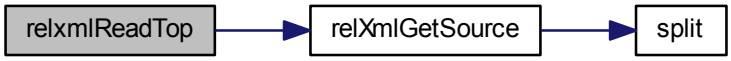
References top::len, top::rel, relXmlGetSource(), top::source, and top::target.

Referenced by RelXmlGetTop(), and topp().

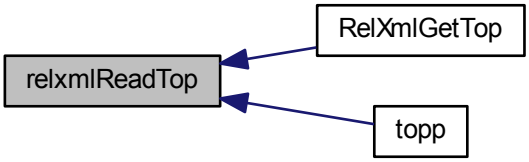
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *sourceAtrr, *targetAtrr, *numberAtrr;
    int i;
    topPtr ret;
    ret = (topPtr) malloc(sizeof(top) * 100); //increase
    ret->rel = (double *) malloc(sizeof(double) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name, (const xmlChar *) "top"))
                {

                    //get the object attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {

                        //ret->source[i] =sourceAtrr;
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the relations attribute of the identity.
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetSource(targetAtrr, i);
                        //ret->target[i] =targetAtrr;
                    }
                    numberAtrr =
                        xmlGetProp(child_node,
                                   (const xmlChar *) "relation");
                    if (numberAtrr) {

                        ret->rel[i] = atof((char *)numberAtrr);
                        ret->len = i + 1;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(numberAtrr);

                    i++;
```

```
                }

            }

        }
    }

    // for(j = 0; j<ret->len; j++)

//printf("bottom \t%s\t%s\t %f\n",*ret[j].source,*ret[j].target,ret->rel[j]);

    return ret;
}/*End of relxmlReadTop*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.42 conversePtr relxmlReadTransposition (xmlNode * *root*, xmlDocPtr *doc*)

Function Function: **relxmlReadTransposition(xmlNode * root, xmlDocPtr doc)** (p. **??**) This functions returns the transposition data from the xml file ....

Definition at line 857 of file RelMDDXmlParser.c.

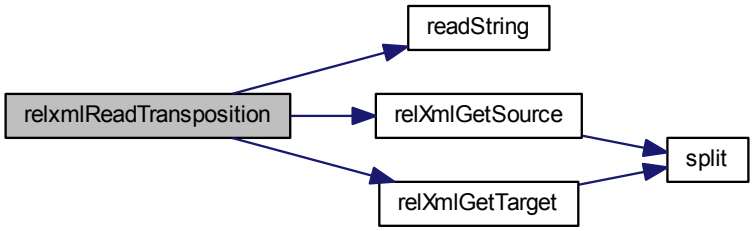
References converse::content, converse::len, converse::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), converse::source, and converse::target.

Referenced by coverselIni(), and RelXmlGetConverse().

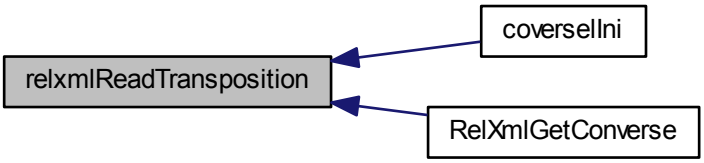
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    converse *ret = NULL;
    ret = (converse *) malloc(sizeof(converse) * 100);
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
```

```
    }

    ret->content = (double *) malloc(sizeof(double) * 100);
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {

        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "transposition")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
       ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len = i + 1;

                    }
                    content_union = xmlNodeGetContent(child_node);

                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
      printf("cfvffd....%d\n.....................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }

    xmlCleanupParser();

    return ret;
}/*End of transposition*/
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.43 unionsPtr relxmlReadUnion (xmlNode * *root*, xmlDocPtr *doc*)

Function Function:unionsPtr **relxmlReadUnion(xmlNode * root, xmlDocPtr doc)** (p. **??**) This functions returns the union data from the xml file ....

Definition at line 503 of file RelMDDXmlParser.c.

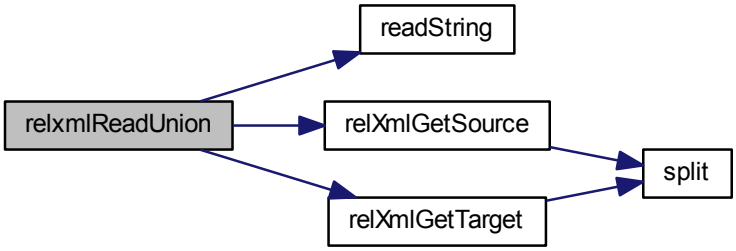
References unions::content, unions::len, unions::len_Content, len_list, readString(), relXmlGetSource(), relXmlGetTarget(), unions::source, and unions::target.

Referenced by RelXmlGetUnion(), and unionIni().

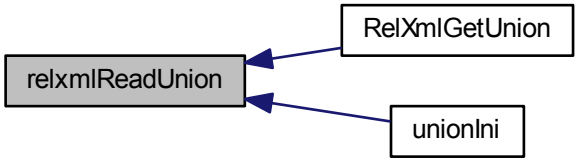
```
{

    xmlNode *cur_node, *child_node;
    xmlChar *content_union;
    xmlChar *sourceAtrr, *targetAtrr;
    unions *ret = NULL;
    ret = (unions *) malloc(sizeof(unions) * 100);
    ret->content = (double *) malloc(sizeof(double) * 100);
    ret->len_Content = (int *) malloc(sizeof(int) * 100);
    ret->source = (char **) malloc(sizeof(char));
    ret->target = (char **) malloc(sizeof(char));
    // basis *ret = (basis*) calloc(40,sizeof(basis));
    if (ret == NULL) {
        /* Memory could not be allocated, so print an error and exit. */
        fprintf(stderr, "Couldn't allocate memory\n");
        exit(EXIT_FAILURE);
    }
    int i;
    for (cur_node = root->children; cur_node != NULL;
         cur_node = cur_node->next) {
```

```
        if (cur_node->type == XML_ELEMENT_NODE &&
            !xmlStrcmp(cur_node->name, (const xmlChar *) "relation")) {

            for (child_node = cur_node->children, i = 0;
                 child_node != NULL; child_node = child_node->next) {

                if (cur_node->type == XML_ELEMENT_NODE &&
                    !xmlStrcmp(child_node->name,
                               (const xmlChar *) "union")) {

                    //get the source attribute
                    sourceAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "source");
                    if (sourceAtrr) {
                        ret[i].source = relXmlGetSource(sourceAtrr, i);
                    }
                    // get the target attributeprop = xmlGetNsProp(cur,
      ("name"), NULL);
                    targetAtrr =
                        xmlGetProp(child_node, (const xmlChar *) "target");
                    if (targetAtrr) {
                        ret[i].target = relXmlGetTarget(targetAtrr, i);
                        ret->len = i + 1;
                    }
                    content_union = xmlNodeGetContent(child_node);

                    if (content_union) {

                        ret[i].content =
                            readString(xmlNodeListGetString
                                       (doc, child_node->xmlChildrenNode,
                                        1));

                        ret->len_Content[i] = len_list;
                        //
     printf("cfvffd....%d\n.....................................dfdfd",  ret->len_Content[i] );
                        len_list = 0;
                        i++;
                    }

                    xmlFree(sourceAtrr);
                    xmlFree(targetAtrr);
                    xmlFree(content_union);

                }

            }

        }
    }

    /*int k,kk;

       for(kk=0;kk<ret->len;kk++){
       printf("\nstarting union here so please be read thank you God\n ");

       printf("  %s || \t%s \n\t..%d..", *ret[kk].source, *ret[kk].target,
       ret->len_Content[kk]);
       for(k=0;k<ret->len_Content[kk];k++)
       printf("  %f\n", ret[kk].content[k]);} */

    xmlCleanupParser();

    return ret;
}/*End relxmlReadUnion */
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.44 char** split (char * *string*)

Function Function: **split(char *string)** (p. **??**) split string into tokens, return token array it serve as tokenizer

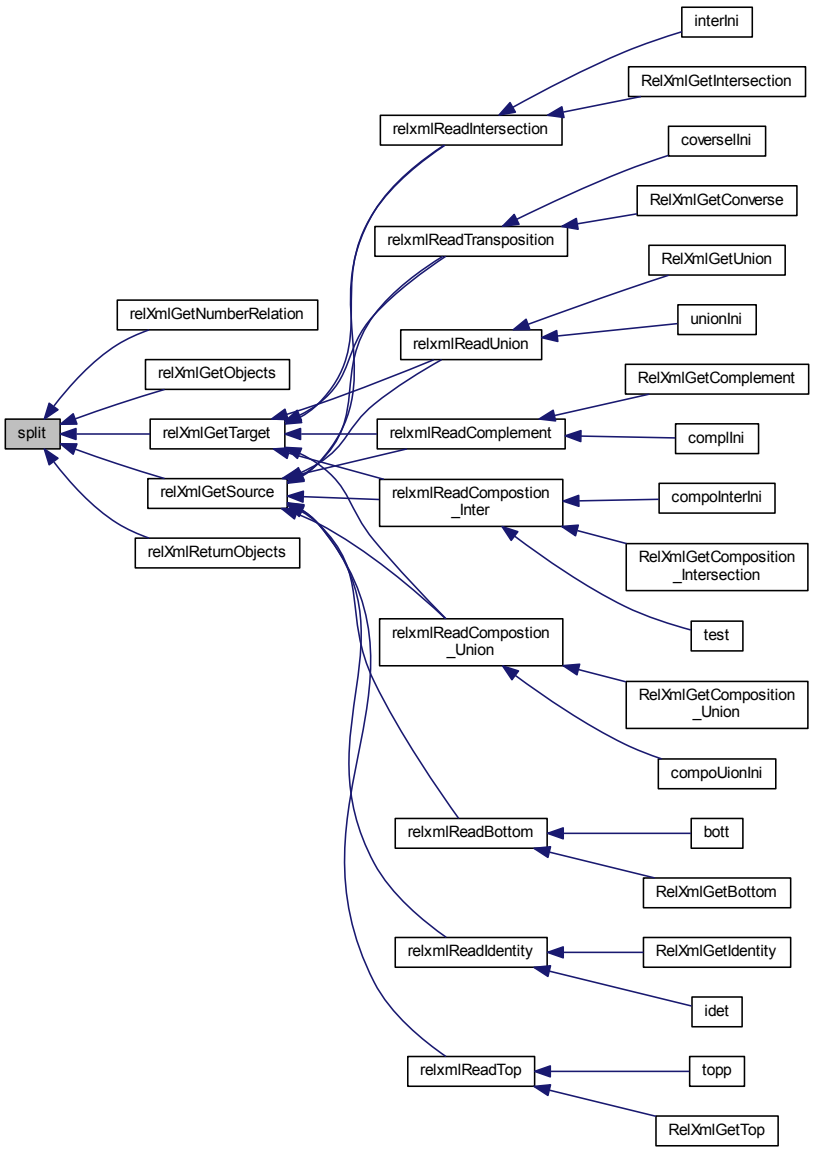
Definition at line 143 of file RelMDDXmlParser.c.

Referenced by relXmlGetNumberRelation(), relXmlGetObjects(), relXmlGetSource(), relXmlGetTarget(), and relXmlReturnObjects().

```
{

    char *delim = ".,:;'\"+-_(){}[]<>*&^%$#@!?~/|\\= \t\r\n";
    char **tokens = NULL;
    char *working = NULL;
    char *token = NULL;
    int idx = 0;
    int len = strlen(string) + 1;

    tokens = malloc(sizeof(char *) * len);
    if (tokens == NULL)
        return NULL;
    working = malloc(sizeof(char) * strlen(string) + 1);
    if (working == NULL)
        return NULL;

    /* to make sure, copy string to a safe place */
    strcpy(working, string);
    for (idx = 0; idx < len; idx++)
        tokens[idx] = NULL;

    token = strtok(working, delim);
```

```
    idx = 0;

    /* always keep the last entry NULL termindated */
    while ((idx < (len - 1)) && (token != NULL)) {
        tokens[idx] = malloc(sizeof(char) * strlen(token) + 1);
        if (tokens[idx] != NULL) {
            strcpy(tokens[idx], token);
            idx++;
            token = strtok(NULL, delim);
        }
    }

    free(working);
    return tokens;
}
```

Here is the caller graph for this function:



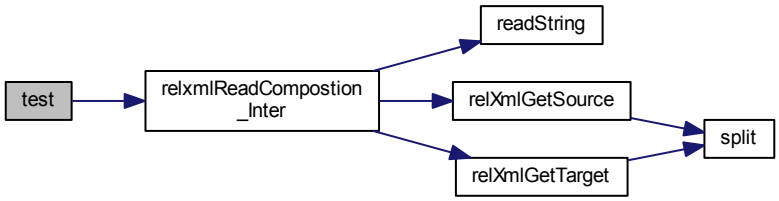
4.10.3.45 compoInterPtr test ( )

Definition at line 1340 of file RelMDDXmlParser.c.

References relxmlReadCompostion_Inter(), and xmlDocument.

```
                    {

//xmlDocPtr doc;
  //doc = xmlParseFile("w3.xml");

  if (xmlDocument== NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL ;
  root = xmlDocGetRootElement(xmlDocument);


   compoInterPtr ret;


   // memset(ret, 0, sizeof(identityPtr));
 ret=  relxmlReadCompostion_Inter(  root,xmlDocument);
  int k,kk;
printf("\n.................. in main.................................\n ");

  for(kk=0;kk<4;kk++){

printf("  %s || \t%s \n", *ret[kk].source, *ret[kk].target );
for(k=0;k<20;k++)
printf("  %f\n", ret[kk].content[k] );
}
 if( !root ||
      !root->name ||
      xmlStrcmp(root->name,(xmlChar *)"RelationBasis") )
  {
     xmlFreeDoc(xmlDocument);
     return NULL;
  }

return ret;
 }
```

Here is the call graph for this function:



4.10.3.46 topPtr topp ( )

Definition at line 1235 of file RelMDDXmlParser.c.

References relxmlReadTop(), and xmlDocument.

```
{

    //xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");
    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root;
```

```
    root = xmlDocGetRootElement(xmlDocument);

    topPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadTop(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}

    return ret;

}
```

Here is the call graph for this function:

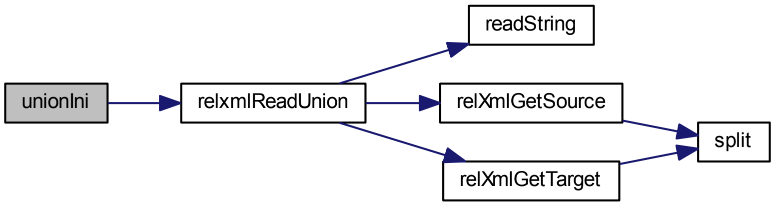
4.10.3.47 unionsPtr unionIni ( )

Definition at line 1147 of file RelMDDXmlParser.c.

References relxmlReadUnion(), and xmlDocument.

```
{

   // xmlDocPtr doc;
   // doc = xmlParseFile("w3.xml");

    if (xmlDocument == NULL)
        printf("error: could not parse file file.xml\n");
    xmlNode *root = NULL;
    root = xmlDocGetRootElement(xmlDocument);

    unionsPtr ret;

    // memset(ret, 0, sizeof(identityPtr));
    ret = relxmlReadUnion(root, xmlDocument);
    //int k,kk;
// printf("\n.................. in main.................................\n ");

    //  for(kk=0;kk<4;kk++){

//printf("  %s || \t%s \n", ret->source[kk], ret->target[kk] );
//for(k=0;k<9;k++)
//printf("  %f\n", ret[kk].content[k] );
//}
    if (!root || !root->name || xmlStrcmp(root->name, (xmlChar *)"RelationBasis
      ")) {
        xmlFreeDoc(xmlDocument);
        return NULL;
    }

    return ret;
}
```

Here is the call graph for this function:



4.10.3.48 int validationXml (const char ∗ *xsdPath,* const char ∗ *xmlPath*)

Definition at line 46 of file schmavalidation.c.

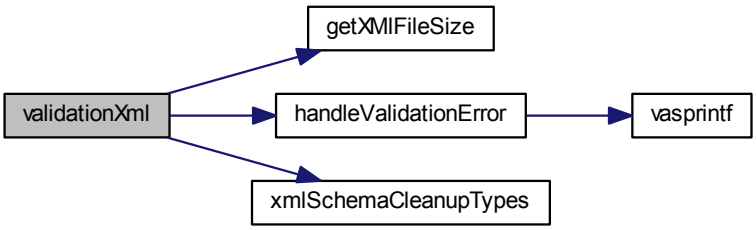
References getXMlFileSize(), handleValidationError(), and xmlSchemaCleanupTypes().

Referenced by RelMDDValidateXmlFile().

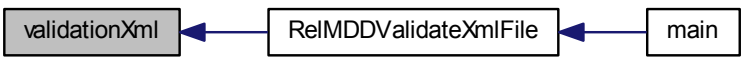
```
                                                            {
    printf("'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''");
    printf("\n");

    printf("XSD File: %s\n", xsdPath);
    printf("XML File: %s\n", xmlPath);

    int xmlLength = getXMlFileSize(xmlPath);
    char *xmlSource = (char *)malloc(sizeof(char) * xmlLength);

    FILE *p = fopen(xmlPath, "r");
    char c;
    unsigned int i = 0;
    while ((c = fgetc(p)) != EOF) {
        xmlSource[i++] = c;
    }
    printf("\n");

   // printf("XML Source:\n\n%s\n", xmlSource); \\ you can use this to print
       out the content of the xml file
    fclose(p);

    printf("\n");

    int result = 42;
    xmlSchemaParserCtxtPtr parserCtxt = NULL;
    xmlSchemaPtr schema = NULL;
    xmlSchemaValidCtxtPtr validCtxt = NULL;

    xmlDocPtr xmlDocumentPointer = xmlParseMemory(xmlSource, xmlLength);
    parserCtxt = xmlSchemaNewParserCtxt(xsdPath);

    if (parserCtxt == NULL) {
        fprintf(stderr, "Could not create XSD schema parsing context.\n");
        goto leave;
    }

    schema = xmlSchemaParse(parserCtxt);
    //xmlSchemaDump(stdout, schema);
    if (schema == NULL) {
        fprintf(stderr, "Could not parse XSD schema.\n");
        goto leave;
    }

    validCtxt = xmlSchemaNewValidCtxt(schema);

    if (!validCtxt) {
        fprintf(stderr, "Could not create XSD schema validation context.\n");
        goto leave;
    }
```

```
    xmlSetStructuredErrorFunc(NULL, NULL);
    xmlSetGenericErrorFunc(NULL, handleValidationError);
    xmlThrDefSetStructuredErrorFunc(NULL, NULL);
    xmlThrDefSetGenericErrorFunc(NULL, handleValidationError);

    result = xmlSchemaValidateDoc(validCtxt, xmlDocumentPointer);

leave:

    if (parserCtxt) {
        xmlSchemaFreeParserCtxt(parserCtxt);
    }

    if (schema) {
        xmlSchemaFree(schema);
    }

    if (validCtxt) {
        xmlSchemaFreeValidCtxt(validCtxt);
    }
    xmlSchemaCleanupTypes();
    xmlCleanupParser();
    xmlMemoryDump();
    printf("\n");
    printf("Validation successful: %s (result: %d)\n", (result == 0) ? "YES" :
      "NO", result);

    return result;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.4 Variable Documentation

4.10.4.1 int col_comp

Definition at line 58 of file RelMDDCompostion.c.

Referenced by RelMDDCompositionOperation(), and union_OperationComposition().

4.10.4.2 int* col_trackComp

Definition at line 54 of file RelMDDCompostion.c.

Referenced by interserction_OperationComposition(), multiplication(), RelMDDCompositionOperation(), and union_OperationComposition().

4.10.4.3 double* element_comp

Definition at line 52 of file RelMDDCompostion.c.

Referenced by interserction_OperationComposition(), multiplication(), RelMDDCompositionOperation(), and union_OperationComposition().

4.10.4.4 double* mat1

Definition at line 48 of file RelMDDCompostion.c.

Referenced by RelMDDCompositionOperation(), and union_OperationComposition().

4.10.4.5 double* mat2

Definition at line 49 of file RelMDDCompostion.c.

Referenced by RelMDDCompositionOperation(), and union_OperationComposition().

4.10.4.6 int row_comp

Definition at line 57 of file RelMDDCompostion.c.

Referenced by RelMDDCompositionOperation(), and union_OperationComposition().

4.10.4.7 int* row_trackComp

Definition at line 53 of file RelMDDCompostion.c.

Referenced by interserction_OperationComposition(), multiplication(), RelMDDCompositionOperation(), and union_OperationComposition().

4.10.4.8 char** source_comp

Definition at line 50 of file RelMDDCompostion.c.

Referenced by interserction_OperationComposition(), RelMDDCompositionOperation(), and union_OperationComposition().

4.10.4.9 char** target_Comp

Definition at line 51 of file RelMDDCompostion.c.

Referenced by interserction_OperationComposition(), RelMDDCompositionOperation(), and union_OperationComposition().

4.10.4.10 xmlDocPtr xmlDocument

CFile This file contains all the functions for maniputions of the xml file content. In this file we imported several functions from Libxml2(fucntions are copy right of .... Copyright (C) 1998-2003 Daniel Veillard.All Rights Reserved.)

.....

This functions accepts and xml file and extract the content, making it availble for the RelMDD system and is and contains all external functions for use by the user.

Definition at line 42 of file RelMDDXmlParser.c.

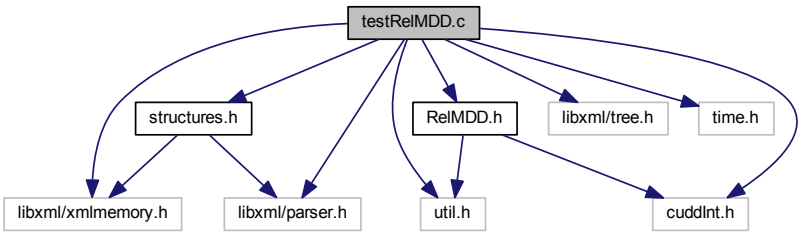
Referenced by bott(), complIni(), compoInterIni(), compoUionIni(), coversellni(), idet(), interIni(), RelMDDParseXmlFile(), relXmlDisplayComments(), RelXmlGetBottom(), RelXmlGetComplement(), RelXmlGetComposition_Intersection(), RelXmlGetComposition_Union(), RelXmlGetConverse(), RelXmlGetIdentity(), RelXmlGetIntersection(), relXmlGetObjects(), RelXmlGetRelation(), RelXmlGetTop(), RelXmlGetUnion(), test(), topp(), and unionIni().

4.10.4.11 const char* xmlfile

4.11 testRelMDD.c File Reference

```
#include "util.h"
#include "cuddInt.h"
#include "structures.h"
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>
#include <libxml/tree.h>
#include <time.h>
#include "RelMDD.h"
```

Include dependency graph for testRelMDD.c:



Functions

- int **main** (int argc, char **argv)

Variables

- char * **file1**
- char * **file2**
- char * **file3**

4.11.1 Function Documentation

4.11.1.1 int main (int *argc,* char ** *argv*)

Definition at line 14 of file testRelMDD.c.

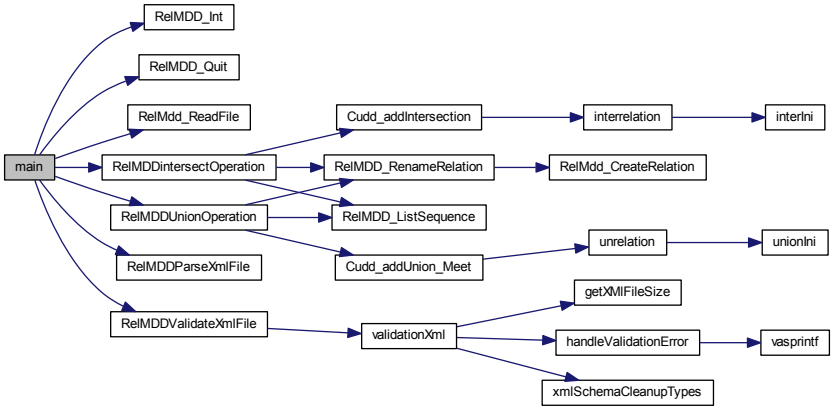
References fileInfor::c, file1, file2, file3, fileInfor::r, fileInfor::rel, RelMDDUnion::relation, RelMDDIntersection::relation, RelMDD_Int(), RelMDD_Quit(), RelMdd_ReadFile(), RelMDDintersectOperation(), RelMDDParseXmlFile(), RelMDDUnionOperation(), RelMDDValidateXmlFile(), fileInfor::sourcelist, and fileInfor::targetlist.

```
{
    RelMDD_Int();
    fileInforPtr cur, cat;
    file1 = argv[1];// first file fisrt matrix
    file2 = argv[2];// second file second matrix
    file3 = argv[3];// xmlfile
   cur = RelMdd_ReadFile(file1);        // set this to be ...
   cat = RelMdd_ReadFile(file2);
  int move;
  int ok;

  //validation of xml file
  ok=RelMDDValidateXmlFile(file3);// validation of xml file
  if (!ok){
  printf("\nvalidation success\n");
  }
  else { printf("Invalid xml file");}
  int i;

  char *source_Object1[cur->r] ; // source sobjects, you can also use those
       read from the file
  char *target_Object2[cur->c];// = { "A", "A", "B" }; // target objects
  for( i=0;i<cur->r; i++){

  source_Object1[i]=cur[i].sourcelist;
  }
  for( i=0;i<cur->c; i++){

  target_Object2[i]= cur[i].targetlist;
  }

// initialization of xml content
 RelMDDParseXmlFile(file3);
 /*

   //testing Top relation
   printf("\n.....................................\n");
     printf("\nTesting Top relation\n");
   RelMDDTopPtr ret;
   ret=   RelMDDTopRelation(cur->r, cur->c, source_Object1, target_Object2);
    printf("\nsource \t target\n");
   for (move = 0; move < cur->r; move++) {

        printf("\n%s\t%s\n",
       ret[move].source_Objects,ret[move].target_Objects);
         }

   printf("\nUniversal Relation \n");
   for (move = 0; move < cur->r*cur->c; move++) {

        printf("\n%f\n", ret->relation[move]);
         }


//testing bottom relation
printf("\n.....................................\n");
printf("\nTesting zero relation\n");
RelMDDBottomPtr ret1;
ret1= RelMDDBottomRelation(cur->r,cur->c, source_Object1,target_Object2);
printf("\nsource \t target\n");
   for (move = 0; move < cur->r; move++) {

        printf("\n%s\t%s\n",
       ret1[move].source_Objects,ret1[move].target_Objects);
         }

printf("\nempty Relation \n");
for (move = 0; move < cur->r*cur->c; move++) {

        printf("\n%f\n", ret1->relation[move]);
         }
```

```
 //testing identity relation
 printf("\n.....................................\n");
 printf("\nTesting identity relation\n");
 RelMDDIdentityPtr ret11 =RelMDDIdentityRelation(cur->r,cur->c,target_Object2);
 printf("\nobjects\n");
  for (move = 0; move < cur->r; move++) {

        printf("\n%s\n", ret11[move].object);
         }


  printf("\nindentity \n");
for (move = 0; move < cur->r*cur->c; move++) {

        printf("\n%f\n", ret11->relation[move]);
 }


*/
/*
//testing converse relaiton
   printf("\n.....................................\n");
     printf("\nTesting converse relation\n");

RelMDDConversePtr ret111 ;
 ret111 =RelMDDConverse_Operation(cur->r,cur->c, source_Object1 ,
       target_Object2, cur->rel);
printf("\nsource \t target\n");
   for (move = 0; move < cur->r; move++) {

        //printf("\n%s\t%s\n",
       ret111[0].source_Objects,ret111[move].target_Objects);
         }


  printf("\nconverse Relation \n");
for (move = 0; move < cur->r*cur->c; move++) {

        printf("\n%f\n", ret111->relation[move]);
         }


//testing complement relaiton
   printf("\n.....................................\n");
     printf("\nTesting complement relation\n");
    RelMDDComplementPtr reta=RelMDDComplement_Operation(cur->r,cur->c,
       source_Object1 , target_Object2, cur->rel);
printf("\nsource\t target\n");
   for (move = 0; move <  cur->r; move++) {

        printf("\n%s\t%s\n",
       reta[move].source_Objects,reta[move].target_Objects);
         }

   printf("\ncomplement Relation \n");
for (move = 0; move <cur->r*cur->c; move++) {

        printf("\n%f\n", reta->relation[move]);
         }
*/

   //testing meet relaiton
   printf("\n.....................................\n");
     printf("\nTesting meet relation\n");
    RelMDDIntersectionPtr reti=RelMDDintersectOperation(cur->r,cur->c,
      source_Object1 ,target_Object2,cur->rel,cat->rel);
    printf("\nsource\t target\n");
   for (move = 0; move < cur->r; move++) {

        printf("\n%s\t%s\n", reti[move].source_Objects,reti[move].
      target_Objects);
         }
          printf("\nMeet Relation \n");
 for (move = 0; move < cur->r*cur->c; move++) {

        printf("\n%f\t\n", reti->relation[move]);
    }

      //testing meet relaiton
   printf("\n.....................................\n");
     printf("\nTesting join relation\n");
    RelMDDUnionPtr retu=RelMDDUnionOperation(cur->r,cur->c, source_Object1 ,
      target_Object2,cur->rel,cat->rel);
     for (move = 0; move < cur->r; move++) {

        printf("\n%s\t%s\n", reti[move].source_Objects,reti[move].
```

```
      target_Objects);
         }
          printf("\njoin Relation \n");
     for (move = 0; move < cur->r*cur->c; move++) {

        printf("\n%f\t\n", retu->relation[move]);
    }
/*
      //testing compositoin relaiton
   printf("\n.....................................\n");
     printf("\nTesting compositon relation\n");
 RelMDDCompositionPtr retr =  RelMDDCompositionOperation(4, 4, 4,
       source_Object1 ,source_Object1,target_Object2,cat->rel,cur->rel);
 for (move = 0; move < 4; move++) {
       printf("\n%s\t%s\n",
       reti[move].source_Objects,reti[move].target_Objects);
  }

   printf("\ncompostion elements \n");
     for (move = 0; move < 16; move++) {

         printf("\n%f\n", retr->relation[move]);
    }
  */
 RelMDD_Quit();
 return 0;
}
```

Here is the call graph for this function:



4.11.2 Variable Documentation

4.11.2.1 char* file1

Definition at line 10 of file testRelMDD.c.

Referenced by main().

4.11.2.2 char* file2

Definition at line 11 of file testRelMDD.c.

Referenced by main().

4.11.2.3 char* file3

Definition at line 12 of file testRelMDD.c.

Referenced by main().

Index