SParse

Generated by Doxygen 1.9.7

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 Allocator Struct Reference

Public Attributes

- NewFunction **new**
- DeleteFunction **delete**
- AllocateFunction **allocate**
- DeAllocateFunction **deallocate**
- ReportFunction **report**
- unsigned int **nbAllocatedObjects**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Allocator/Allocator.h
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/include/Allocator.h

3.2 AllocInfo Struct Reference

Public Attributes

- Allocator ∗ **ptr**
- AllocInfo ∗ **prev**
- AllocInfo ∗ **next**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/ObjectStore/ObjectStore.c

3.3 Array Class Reference

Public Member Functions

- PUBLIC Array ∗ Array_new (ArrayParam ∗param)

  Create a new instance of the class Array.
- PUBLIC Array ∗ Array_newFromFile (FileIo ∗fileIo, ArrayParam ∗param)

  Create a new instance of the class Array from a fileIo stream.
- PUBLIC void **Array_delete** (Array ∗this)

  Delete an instance of the class Array.
- PUBLIC Array ∗ Array_copy (Array ∗this)

  Copy an instance of the class Array.
- PUBLIC int Array_compare (Array ∗this, Array ∗compared)

  Compare 2 instances of the class Array.
- PUBLIC void **Array_print** (Array ∗this)

  Print an instance of the class Array.

Public Attributes

- Object **object**
- unsigned int **nbElements**

3.3.1 Member Function Documentation

3.3.1.1 Array_compare()

```
PUBLIC int Array_compare (
          Array ∗ this,
          Array ∗ compared )
```

Compare 2 instances of the class Array.

Returns

0 if different, 1 if equal.

3.3.1.2 Array_copy()

```
PUBLIC Array ∗ Array_copy (
          Array ∗ this )
```

Copy an instance of the class Array.

Returns

Copy of the given instance.

3.3.1.3 Array_new()

```
PUBLIC Array * Array_new (
            ArrayParam * param )
```

Create a new instance of the class Array.

Returns

New instance.

3.3.1.4 Array_newFromFile()

```
PUBLIC Array * Array_newFromFile (
            FileIo * fileIo,
            ArrayParam * param )
```

Create a new instance of the class Array from a fileIo stream.

Returns

New instance.

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Array/Array.c

3.4 ArrayParam Struct Reference

Public Attributes

- unsigned int **defaultSize**
- unsigned int **storageMode**
- unsigned int **autoresize**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Array/Array.h
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/include/Array.h

3.5 BTree Class Reference

Public Attributes

- Object **object**
- Node * **root**
- unsigned int **order**
- unsigned int **depth**
- unsigned short int **nbObjects**
- unsigned int **nodeSize**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/BTree/BTree.c
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/BTree/BTree.rescue.c

3.6 Buffer Struct Reference

Public Attributes

- Object **object**
- String ∗ **string**
- char ∗ **currentPtr**
- char ∗ **startPtr**
- int **nbCharRead**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/TransUnit/Buffer.h

3.7 Class Struct Reference

Public Attributes

- Constructor **f_new**
- Destructor **f_delete**
- Copy_Operator **f_copy**
- Comp_Operator **f_comp**
- Printer **f_print**
- Sizer **f_size**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/include/Class.h
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Object/Class.h

3.8 Configuration Class Reference

Public Member Functions

- PUBLIC Configuration ∗ Configuration_new (String ∗input)

  Create an instance of configuration class from th input string.
- PUBLIC void **Configuration_delete** (Configuration ∗this)

  Destroy an instance of configuration class.
- PUBLIC void **Configuration_print** (Configuration ∗this)

  Print an instance of configuration class.
- PUBLIC unsigned int **Configuration_getSize** (Configuration ∗this)

  Destroy an instance of configuration class.
- PUBLIC List ∗ **Configuration_getProducts** (Configuration ∗this)

  TBD.
- PUBLIC void **Configuration_parseProducts** (Configuration ∗this)

  TBD.

Public Attributes

- Object **object**
- List ∗ **products**

3.8.1 Member Function Documentation

3.8.1.1 Configuration_new()

```
PUBLIC Configuration * Configuration_new (
            String * input )
```

Create an instance of configuration class from th input string.

Returns

Status.

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/Configuration/Configuration.c

3.9 ConnectionParam Struct Reference

Public Attributes

- int ∗ **client_fd**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/HTTPServer/HTTPServer.c

3.10 Declarator Struct Reference

Public Attributes

- DeclaratorType **type**
- DeclaratorScope **scope**
- char ∗ **name**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/C89Grammar/Declarator.c

3.11 FileDesc Class Reference

Public Member Functions

- PUBLIC FileDesc ∗ **FileDesc_new** ()

  TBD.
- PUBLIC void **FileDesc_delete** (FileDesc ∗this)

  TBD.
- PUBLIC FileDesc ∗ **FileDesc_copy** (FileDesc ∗this)

  TBD.
- PUBLIC void **FileDesc_setFullName** (FileDesc ∗this, String ∗fullName)

  TBD.
- PUBLIC String ∗ **FileDesc_getFullName** (FileDesc ∗this)

  TBD.
- PUBLIC String ∗ **FileDesc_getName** (FileDesc ∗this)

  TBD.
- PUBLIC String ∗ **FileDesc_load** (FileDesc ∗this)

  Load the content of a file.

Public Attributes

- Object **object**
- String ∗ **name**
- String ∗ **fullName**

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/FileMgr/FileDesc.c

3.12 Fileilo Class Reference

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/FileIo/FileIo.c

3.13 FileIo Struct Reference

Public Member Functions

- PUBLIC FileIo ∗ FileIo_new ()

  Create a new instance of the class FileIo.
- PUBLIC void FileIo_delete (FileIo ∗this)

  Delte an instance of the class FileIo.
- PUBLIC FileIo ∗ FileIo_copy (FileIo ∗this)

  Copy an instance of the class FileIo.
- PUBLIC int FileIo_comp (FileIo ∗this, FileIo ∗compare)

Compare an instance of the class FileIo to another one.
- PUBLIC void FileIo_print (FileIo ∗this)

  Print an instance of the class FileIo.
- PUBLIC unsigned int FileIo_getSize (FileIo ∗this)

  Return the size of an instance of the class FileIo.
- PUBLIC void FileIo_openFile (FileIo ∗this, String ∗fullFileName)

  Open an instance of the class FileIo for reading/writing.
- PUBLIC void FileIo_createFile (FileIo ∗this, String ∗fullFileName)

  Create a new file.
- PUBLIC void FileIo_openDir (FileIo ∗this, String ∗fullFileName)

  Create a new file.

Public Attributes

- Object **object**
- FILE ∗ **f**
- int **status**

3.13.1 Member Function Documentation

3.13.1.1 FileIo_comp()

```
PUBLIC int FileIo_comp (
             FileIo ∗ this,
             FileIo ∗ compare )
```

Compare an instance of the class FileIo to another one.

Returns

0 if equal.

3.13.1.2 FileIo_copy()

```
PUBLIC FileIo ∗ FileIo_copy (
             FileIo ∗ this )
```

Copy an instance of the class FileIo.

Returns

Copy of the instance.

3.13.1.3 FileIo_createFile()

```
PUBLIC void FileIo_createFile (
             FileIo ∗ this,
             String ∗ fullFileName )
```

Create a new file.

Parameters

in	*String*	Full path of file to create

Returns

none

3.13.1.4 FileIo_delete()

```
PUBLIC void FileIo_delete (
            FileIo * this )
```

Delte an instance of the class FileIo.

Returns

none

3.13.1.5 FileIo_getSize()

```
PUBLIC unsigned int FileIo_getSize (
            FileIo * this )
```

Return the size of an instance of the class FileIo.

Returns

Size in bytes.

3.13.1.6 FileIo_new()

```
PUBLIC FileIo * FileIo_new ( )
```

Create a new instance of the class FileIo.

Returns

New FileIo instance or NULL if failed to allocate.

3.13.1.7 FileIo_openDir()

```
PUBLIC void FileIo_openDir (
            FileIo * this,
            String * fullFileName )
```

Create a new file.

Parameters

in	*String*	Full path of file to create

Returns

none

3.13.1.8 FileIo_openFile()

```
PUBLIC void FileIo_openFile (
            FileIo * this,
            String * fullFileName )
```

Open an instance of the class FileIo for reading/writing.

Parameters

in	*String*	Full path of file to open

Returns

none

3.13.1.9 FileIo_print()

```
PUBLIC void FileIo_print (
            FileIo * this )
```

Print an instance of the class FileIo.

Returns

none.

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/FileIo/FileIo.c

3.14 FileMgr Class Reference

Public Member Functions

- PUBLIC void **FileMgr_delete** (FileMgr ∗this)

  Delete an instance of the class FileMgr.
- PUBLIC FileMgr ∗ FileMgr_copy (FileMgr ∗this)

Copy an instance of the class FileMgr.
- PUBLIC FileMgr ∗ FileMgr_getRef ()

  Get a reference to the singleton instance of FileMgr.
- PUBLIC unsigned int FileMgr_getSize (FileMgr ∗this)

  Return the size in byte of the class or object.
- PUBLIC unsigned int FileMgr_setRootLocation (FileMgr ∗this, const char ∗location)

  Set the root location.
- PUBLIC char ∗ FileMgr_getRootLocation (FileMgr ∗this)

  TBD.
- PUBLIC unsigned int FileMgr_addDirectory (FileMgr ∗this, const char ∗directoryName)

  Add all files in the given directory to the list of managed files.
- PUBLIC FileDesc ∗ FileMgr_addFile (FileMgr ∗this, const char ∗fileName)

  Add a files to the list of managed files.
- PUBLIC String ∗ FileMgr_load (FileMgr ∗this, const char ∗fileName)

  Load a managed file into a String.
  @parameter File Name.
- PUBLIC void FileMgr_write (FileMgr ∗this, const char ∗fileName, String ∗content)

  Write a string into a file.
- PUBLIC List ∗ FileMgr_filterFiles (FileMgr ∗this, const char ∗pattern)

  TBD.

Public Attributes

- Object **object**
- List ∗ **files**
- List ∗ **directories**
- char ∗ **separator**
- String ∗ **rootLocation**

3.14.1 Member Function Documentation

3.14.1.1 FileMgr_addDirectory()

```
PUBLIC unsigned int FileMgr_addDirectory (
          FileMgr ∗ this,
          const char ∗ directoryName )
```

Add all files in the given directory to the list of managed files.

Returns

Status.

3.14.1.2 FileMgr_addFile()

```
PUBLIC FileDesc ∗ FileMgr_addFile (
          FileMgr ∗ this,
          const char ∗ fileName )
```

Add a files to the list of managed files.

Returns

Status.

3.14.1.3 FileMgr_copy()

```
PUBLIC FileMgr * FileMgr_copy (
             FileMgr * this )
```

Copy an instance of the class FileMgr.

Returns

New instance

3.14.1.4 FileMgr_filterFiles()

```
PUBLIC List * FileMgr_filterFiles (
             FileMgr * this,
             const char * pattern )
```

TBD.

Returns

TBD

3.14.1.5 FileMgr_getRef()

```
PUBLIC FileMgr * FileMgr_getRef ( )
```

Get a reference to the singleton instance of FileMgr.

Returns

Reference to the singleton.

3.14.1.6 FileMgr_getRootLocation()

```
PUBLIC char * FileMgr_getRootLocation (
             FileMgr * this )
```

TBD.

Returns

Status.

3.14.1.7 FileMgr_getSize()

```
PUBLIC unsigned int FileMgr_getSize (
            FileMgr * this )
```

Return the size in byte of the class or object.

Returns

Size in byte of Class or instance.

3.14.1.8 FileMgr_load()

```
PUBLIC String * FileMgr_load (
            FileMgr * this,
            const char * fileName )
```

Load a managed file into a String.

@parameter File Name.

Returns

Content of file.

3.14.1.9 FileMgr_setRootLocation()

```
PUBLIC unsigned int FileMgr_setRootLocation (
            FileMgr * this,
            const char * location )
```

Set the root location.

Returns

Status.

3.14.1.10 FileMgr_write()

```
PUBLIC void FileMgr_write (
            FileMgr * this,
            const char * fileName,
            String * content )
```

Write a string into a file.

Returns

None

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/FileMgr/FileMgr.c

3.15 FileReader Class Reference

Public Member Functions

- PUBLIC FileReader ∗ FileReader_new (FileDesc ∗fileDesc, FileMgr ∗fileMgr)

    Create a new FileReader object.
- PUBLIC void **FileReader_delete** (FileReader ∗this)

    Delete an instance of a FileReader object.
- PUBLIC FileReader ∗ FileReader_copy (FileReader ∗this)

    Copy an instance of a FileReader object.
- PUBLIC void **FileReader_print** (FileReader ∗this)

    Print an instance of a FileReader object.
- PUBLIC unsigned int FileReader_getSize (FileReader ∗this)

    Return the size in bytes of an instance of a FileReader object.
- PUBLIC char ∗ FileReader_getBuffer (FileReader ∗this)

    Returns the buffer of a FileReader object.
- PUBLIC String ∗ FileReader_getName (FileReader ∗this)

    Returns the name of a FileReader object.
- PUBLIC char ∗ FileReader_addFile (FileReader ∗this, String ∗fileName)

    Add a new file buffer for filename.

Public Attributes

- Object **object**
- List ∗ **buffers**
- FileDesc ∗ **fileDesc**
- FileMgr ∗ **fileMgr**
- String ∗ **currentBuffer**
- List ∗ **preferredDirs**

3.15.1 Member Function Documentation

3.15.1.1 FileReader_addFile()

```
PUBLIC char * FileReader_addFile (
            FileReader * this,
            String * fileName )
```

Add a new file buffer for filename.

Returns

File buffer

3.15.1.2 FileReader_copy()

```
PUBLIC FileReader * FileReader_copy (
             FileReader * this )
```

Copy an instance of a FileReader object.

Returns

New instance

3.15.1.3 FileReader_getBuffer()

```
PUBLIC char * FileReader_getBuffer (
             FileReader * this )
```

Returns the buffer of a FileReader object.

Returns

Buffer of characters

3.15.1.4 FileReader_getName()

```
PUBLIC String * FileReader_getName (
             FileReader * this )
```

Returns the name of a FileReader object.

Returns

File name

3.15.1.5 FileReader_getSize()

```
PUBLIC unsigned int FileReader_getSize (
             FileReader * this )
```

Return the size in bytes of an instance of a FileReader object.

Returns

Size in bytes

3.15.1.6 FileReader_new()

```
PUBLIC FileReader * FileReader_new (
            FileDesc * fileDesc,
            FileMgr * fileMgr )
```

Create a new FileReader object.

Returns

Created FileReader object.

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/FileReader/FileReader.c

3.16 Grammar Struct Reference

Public Attributes

- Object **object**
- Grammar ∗(∗ **new**)(void)
- void(∗ **delete**)(Grammar ∗this)
- Grammar ∗(∗ **copy**)(Grammar ∗this)
- void(∗ **print**)(Grammar ∗this)

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/Grammar/Grammar.h
- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/include/Grammar.h

3.17 Grammar2 Class Reference

Public Member Functions

- PUBLIC Grammar2 ∗ Grammar2_new (FileReader ∗fr, SdbMgr ∗sdbMgr)

  Create an instance of the class Grammar2.
- PUBLIC void **Grammar2_delete** (Grammar2 ∗this)

  Delete an instance of the class Grammar2.
- PUBLIC Grammar2 ∗ Grammar2_copy (Grammar2 ∗this)

  Copy an instance of the class Grammar2.

Public Attributes

- Object **object**
- void ∗ **scanner**
- SdbMgr ∗ **sdbMgr**
- FileReader ∗ **reader**
- TransUnit ∗ **unit**
- char ∗ **buffer**
- int **node_text_position**
- GrammarContext ∗ **current**
- List ∗ **contexts**

3.17.1 Member Function Documentation

3.17.1.1 Grammar2_copy()

```
PUBLIC Grammar2 ∗ Grammar2_copy (
            Grammar2 ∗ this )
```

Copy an instance of the class Grammar2.

Returns

Copied instance.

3.17.1.2 Grammar2_new()

```
PUBLIC Grammar2 ∗ Grammar2_new (
            FileReader ∗ fr,
            SdbMgr ∗ sdbMgr )
```

Create an instance of the class Grammar2.

Returns

New instance.

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/Grammar2/Grammar2.c

3.18 GrammarC99 Class Reference

Public Member Functions

- PUBLIC Grammar ∗ GrammarC99_new (FileDesc ∗fileDesc, FileMgr ∗fm)
  Create an instance of the class GrammarC99.
- PUBLIC void **GrammarC99_delete** (Grammar ∗this)
  TBC.
- PUBLIC void **GrammarC99_print** (Grammar ∗this)
  TBC.
- PUBLIC unsigned int **GrammarC99_getSize** (Grammar ∗this)
  TBC.
- PUBLIC void **GrammarC99_process** (GrammarC99 ∗this)
  TBC.

Public Attributes

- Grammar **grammar**
- TransUnit ∗ **transUnit**
- FileMgr ∗ **fm**
- void ∗ **scanner**

3.18.1 Member Function Documentation

3.18.1.1 GrammarC99_new()

```
PUBLIC Grammar ∗ GrammarC99_new (
            FileDesc ∗ fileDesc,
            FileMgr ∗ fm )
```

Create an instance of the class GrammarC99.

Returns

New instance.

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/GrammarC99/GrammarC99.c

3.19 GrammarContext Struct Reference

Public Attributes

- Object **object**
- unsigned int **lastNode**
- unsigned int **includeNodeBranch**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/Grammar2/Grammar2.c

3.20 HTTPRequest Class Reference

Public Member Functions

- PRIVATE void **HTTPRequest_delete** (HTTPRequest ∗this)

  Delete an instance of the class HTTPRequest.
- PRIVATE HTTPRequest ∗ HTTPRequest_copy (HTTPRequest ∗this)

  Copy an instance of the class HTTPRequest.
- PRIVATE int HTTPRequest_compare (HTTPRequest ∗this, HTTPRequest ∗compared)

  Compare 2 instances of the class HTTPRequest.
- PRIVATE void **HTTPRequest_print** (HTTPRequest ∗this)

  Print an instance of the class HTTPRequest.
- PRIVATE unsigned int HTTPRequest_getSize (HTTPRequest ∗this)

  Get the size of an HTTPRequest. If parameter is 0 return the size of the class.

Public Attributes

- Object **object**
- enum Method **method**
- String ∗ **path**
- int **majorVersion**
- int **minorVersion**
- Map ∗ **headers**
- String ∗ **body**
- int **isValid**

3.20.1 Member Function Documentation

3.20.1.1 HTTPRequest_compare()

```
PRIVATE int HTTPRequest_compare (
            HTTPRequest ∗ this,
            HTTPRequest ∗ compared )
```

Compare 2 instances of the class HTTPRequest.

Returns

0 if different, 1 if equal.

3.20.1.2 HTTPRequest_copy()

```
PRIVATE HTTPRequest ∗ HTTPRequest_copy (
            HTTPRequest ∗ this )
```

Copy an instance of the class HTTPRequest.

Returns

Copy of the instance

3.20.1.3 HTTPRequest_getSize()

```
PRIVATE unsigned int HTTPRequest_getSize (
            HTTPRequest ∗ this )
```

Get the size of an HTTPRequest. If parameter is 0 return the size of the class.

Returns

Number of items.

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/HTTPServer/HTTPRequest.h

3.21 HTTPResponse Class Reference

Public Attributes

- Object **object**
- int **statusCode**
- enum Reason **reason**
- int **majorVersion**
- int **minorVersion**
- Map ∗ **headers**
- String ∗ **body**
- int **isValid**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/HTTPServer/HTTPResponse.h

3.22 HTTPServer Class Reference

Public Member Functions

- PRIVATE HTTPRequest ∗ HTTPRequest_new (char ∗buffer)

  Create a new instance of the class HTTPRequest.
- PRIVATE int HTTPResponse_compare (HTTPResponse ∗this, HTTPResponse ∗compared)

  Compare 2 instances of the class HTTPResponse.
- PRIVATE void **HTTPResponse_print** (HTTPResponse ∗this)

  Print an instance of the class HTTPResponse.
- PRIVATE unsigned int HTTPResponse_getSize (HTTPResponse ∗this)

  Get the size of an HTTPResponse. If parameter is 0 return the size of the class.
- PUBLIC HTTPServer ∗ HTTPServer_new ()

  Create a new instance of the class HTTPServer.

Public Attributes

- Object **object**
- int **port**
- struct sockaddr_in **server_addr**
- int **fd**

3.22.1 Member Function Documentation

3.22.1.1 HTTPRequest_new()

```
PRIVATE HTTPRequest ∗ HTTPRequest_new (
            char ∗ buffer )
```

Create a new instance of the class HTTPRequest.

Parameters

in	*none*	

Returns

New instance of class HTTPRequest.

3.22.1.2 HTTPResponse_compare()

```
PRIVATE int HTTPResponse_compare (
            HTTPResponse * this,
            HTTPResponse * compared )
```

Compare 2 instances of the class HTTPResponse.

Returns

0 if different, 1 if equal.

3.22.1.3 HTTPResponse_getSize()

```
PRIVATE unsigned int HTTPResponse_getSize (
            HTTPResponse * this )
```

Get the size of an HTTPResponse. If parameter is 0 return the size of the class.

Returns

Number of items.

3.22.1.4 HTTPServer_new()

```
PUBLIC HTTPServer * HTTPServer_new ( )
```

Create a new instance of the class HTTPServer.

Parameters

in	*none*	

Returns

New instance of class HHTPServer.

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/HTTPServer/HTTPServer.c
- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/HTTPServer/HTTPRequest.h
- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/HTTPServer/HTTPResponse.h

3.23 IncludeInfo Struct Reference

Public Attributes

- Object **object**
- String ∗ **pattern**
- List ∗ **dirs**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/FileReader/FileReader.c

3.24 List Class Reference

Public Member Functions

- PUBLIC List ∗ List_new ()

  Create a new instance of the class List.
- PUBLIC List ∗ List_newFromAllocator (Allocator ∗allocator)

  Create a new instance of the class List using a custom allocator.
- PUBLIC void **List_delete** (List ∗this)

  Delete an instance of the class List.
- PUBLIC List ∗ List_copy (List ∗this)

  Copy an instance of the class List.
- PUBLIC int List_compare (List ∗this, List ∗compared)

  Compare 2 instances of the class List.
- PUBLIC void **List_print** (List ∗this)

  Print an instance of the class List.
- PUBLIC void List_insertHead (List ∗this, void ∗item, int isOwner)

  Insert an item at the head of a list instance.
- PUBLIC void List_insertTail (List ∗this, void ∗item, int isOwner)

  Insert an item at the tail of a List instance.
- PUBLIC void List_merge (List ∗this, List ∗l1)

  Merge a list into a List instance.
- PUBLIC void List_forEach (List ∗this, void(∗method)(void ∗o))

  Execute a given function for each item in an instance of List..
- PUBLIC unsigned int List_getNbNodes (List ∗this)

  Get the number of items in List instance.
- PUBLIC unsigned int List_getSize (List ∗this)

  Get the size of a List obejct. If parameter is 0 return the size of the class.
- PUBLIC void ∗ **List_removeHead** (List ∗this)

  Remove the head item in an instance of LIst
- PUBLIC void ∗ **List_removeTail** (List ∗this)

Remove the tail item in an instance of List
- PUBLIC void ∗ **List_getHead** (List ∗this)

  Get the head item in an insatnce of LIst
- PRIVATE ListNode ∗ ListNode_new (Object ∗object, int isOwner)

  Create a new instance of the class ListNode.
- PRIVATE ListNode ∗ ListNode_newFromAllocator (Allocator ∗allocator, Object ∗object, int isOwner)

  Create a new instance of the class List using a custom allocator.
- PRIVATE void **ListNode_delete** (ListNode ∗this)

  Delete an instance of the class List.
- PRIVATE ListNode ∗ ListNode_copy (ListNode ∗this)

  Copy an instance of the class List.
- PRIVATE int ListNode_compare (ListNode ∗this, ListNode ∗compared)

  Compare 2 instances of the class List.
- PRIVATE void **ListNode_print** (ListNode ∗this)

  Print an instance of the class List.

Public Attributes

- Object **object**
- ListNode ∗ **head**
- ListNode ∗ **tail**
- ListNode ∗ **iterator**
- unsigned int **nbNodes**

3.24.1 Member Function Documentation

3.24.1.1 List_compare()

```
PUBLIC int List_compare (
            List ∗ this,
            List ∗ compared )
```

Compare 2 instances of the class List.

Returns

0 if different, 1 if equal.

3.24.1.2 List_copy()

```
PUBLIC List ∗ List_copy (
            List ∗ this )
```

Copy an instance of the class List.

Returns

Copy of the given instance.

3.24.1.3 List_forEach()

```
PUBLIC void List_forEach (
            List ∗ this,
            void(∗)(void ∗o) method )
```

Execute a given function for each item in an instance of List..

Parameters

in	*f*	Pointer to function.

3.24.1.4 List_getNbNodes()

```
PUBLIC unsigned int List_getNbNodes (
            List * this )
```

Get the number of items in List instance.

Returns

Number of items.

3.24.1.5 List_getSize()

```
PUBLIC unsigned int List_getSize (
            List * this )
```

Get the size of a List obejct. If parameter is 0 return the size of the class.

Returns

Number of items.

3.24.1.6 List_insertHead()

```
PUBLIC void List_insertHead (
            List * this,
            void * item,
            int isOwner )
```

Insert an item at the head of a list instance.

Parameters

in	*item*	Reference to item.

3.24.1.7 List_insertTail()

```
PUBLIC void List_insertTail (
            List * this,
            void * item,
            int isOwner )
```

Insert an item at the tail of a List instance.

Parameters

in	*item*	Reference to item.

3.24.1.8 List_merge()

```
PUBLIC void List_merge (
            List * this,
            List * l1 )
```

Merge a list into a List instance.

Parameters

in	*l1*	Reference to list to merge.

3.24.1.9 List_new()

```
PUBLIC List * List_new ( )
```

Create a new instance of the class List.

Returns

New instance.

3.24.1.10 List_newFromAllocator()

```
PUBLIC List * List_newFromAllocator (
            Allocator * allocator )
```

Create a new instance of the class List using a custom allocator.

Returns

New instance.

3.24.1.11 ListNode_compare()

```
PRIVATE int ListNode_compare (
            ListNode * this,
            ListNode * compared )
```

Compare 2 instances of the class List.

Returns

0 if different, 1 if equal.

3.24.1.12 ListNode_copy()

```
PRIVATE ListNode * ListNode_copy (
            ListNode * this )
```

Copy an instance of the class List.

Returns

Copy of the given instance.

3.24.1.13 ListNode_new()

```
PRIVATE ListNode * ListNode_new (
            Object * object,
            int isOwner )
```

Create a new instance of the class ListNode.

Returns

New instance.

3.24.1.14 ListNode_newFromAllocator()

```
PRIVATE ListNode * ListNode_newFromAllocator (
            Allocator * allocator,
            Object * object,
            int isOwner )
```

Create a new instance of the class List using a custom allocator.

Returns

New instance.

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/List/List.c
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/List/ListNode.h

3.25 MacroDefinition Struct Reference

Public Attributes

- Object **object**
- String ∗ **body**
- List ∗ **parameters**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/TransUnit/MacroDefinition.h

3.26 MacroStore Struct Reference

Public Attributes

- Object **object**
- struct MacroStoreNode ∗ **root**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/TransUnit/MacroStore.h

3.27 MacroStoreNode Struct Reference

Public Attributes

- int **isLeaf**
- MacroDefinition ∗ **def**
- void ∗ **children** [MAX_CHILDREN]

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/TransUnit/MacroStore.h

3.28 Malloc Struct Reference

Public Attributes

- Allocator **allocator**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Allocator/Malloc.c

3.29 Map Class Reference

Public Member Functions

- PRIVATE TaskMgr ∗ TaskMgr_new ()

    Create a new instance of the class TaskMgr.
- PUBLIC Map ∗ Map_new ()

    Create a new instance of the class Map.
- PUBLIC Map ∗ Map_newFromAllocator (Allocator ∗allocator)

    Create a new instance of the class Map using a specific allocator.
- PUBLIC void **Map_delete** (Map ∗this)

    Delete an instance of the class Map.
- PUBLIC Map ∗ Map_copy (Map ∗this)

    Copy an instance of the class Map.
- PUBLIC unsigned int Map_insert (Map ∗this, String ∗s, void ∗p, int isOwner)

    Insert an object into a Map instance.
- PUBLIC unsigned int **Map_find** (Map ∗this, String ∗s, void ∗∗p)

    TBD.
- PUBLIC void **Map_print** (Map ∗this)

    Print a Map instance.
- PUBLIC unsigned int Map_getSize (Map ∗this)

    Provide the size of a Map instance.
- PUBLIC List ∗ Map_getAll (Map ∗this)

    Get all the entries in an instance of a Map.

Public Attributes

- Object **object**
- List ∗ **htable** [HTABLE_SIZE]

3.29.1 Member Function Documentation

3.29.1.1 Map_copy()

```
PUBLIC Map ∗ Map_copy (
             Map ∗ this )
```

Copy an instance of the class Map.

Returns

Copy of instance of NULL if failed to allocate.

3.29.1.2 Map_getAll()

```
PUBLIC List ∗ Map_getAll (
             Map ∗ this )
```

Get all the entries in an instance of a Map.

Returns

List of map objects

3.29.1.3 Map_getSize()

```
PUBLIC unsigned int Map_getSize (
            Map * this )
```

Provide the size of a Map instance.

Returns

Size in bytes

3.29.1.4 Map_insert()

```
PUBLIC unsigned int Map_insert (
            Map * this,
            String * s,
            void * p,
            int isOwner )
```

Insert an object into a Map instance.

Returns

1 is inserted

3.29.1.5 Map_new()

```
PUBLIC Map * Map_new ( )
```

Create a new instance of the class Map.

Returns

New Map instance or NULL if failed to allocate.

3.29.1.6 Map_newFromAllocator()

```
PUBLIC Map * Map_newFromAllocator (
            Allocator * allocator )
```

Create a new instance of the class Map using a specific allocator.

Returns

New Map instance or NULL if failed to allocate.

3.29.1.7 TaskMgr_new()

```
PRIVATE TaskMgr * TaskMgr_new ( )
```

Create a new instance of the class TaskMgr.

Returns

New taskMgr instance or NULL if failed to allocate.

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Map/Map.c
- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/TaskMgr/TaskMgr.c

3.30 MapEntry Struct Reference

Public Attributes

- Object **object**
- String ∗ **s**
- void ∗ **item**
- int **isOwner**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Map/MapEntry.c

3.31 mem_align Union Reference

Public Attributes

- void ∗ **a**
- long int **b**
- long long **c**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/include/Types.h
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Types/Types.h

3.32 Mutex Struct Reference

Public Attributes

- pthread_mutex_t ∗ **mutex**
- pthread_cond_t **cond**

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/TaskMgr/Mutex.h

3.33 MyAllocator Struct Reference

Public Attributes

- Allocator **allocator**
- unsigned int **size**
- void ∗ **memory_start**
- void ∗ **memory_end**
- void ∗ **pointer**
- void ∗ **memory**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/List/tests/MyAllocator.c
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Map/tests/MyAllocator.c
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/ObjectStore/tests/MyAllocator.c
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/SkipList/tests/MyAllocator.c
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/String/tests/MyAllocator.c

3.34 MyType Struct Reference

Public Attributes

- char ∗ **sval**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/GrammarC99/MyType.h

3.35 Node Struct Reference

Public Attributes

- unsigned int **nbKeyUsed**
- unsigned int **isLeaf**
- Object ∗∗ **keys**
- Object ∗∗ **leaves**
- Node ∗∗ **children**
- Object ∗ **buffer** [18]

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/BTree/Node.h
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/include/Node.h

3.36 Object Struct Reference

Public Member Functions

- PUBLIC Object ∗ Object_new (unsigned int size, Class ∗class)

  Create an instance of the class Object.
- PUBLIC Object ∗ Object_newFromAllocator (Class ∗class, Allocator ∗allocator)

  TBD.
- PUBLIC void **Object_delete** (Object ∗this)

  Delete an instance of the class Object.
- PUBLIC void **Object_deallocate** (Object ∗this)

  De-allocate an instance of the class Object.
- PUBLIC Object ∗ Object_copy (Object ∗this)

  Copy an instance of the class Object.
- PUBLIC int Object_comp (Object ∗this, Object ∗compared)

  Compare 2 instances of the class Object.
- PUBLIC char ∗ Object_print (Object ∗this)

  Print an instance of the class Object into a buffer of characters.
- PUBLIC Object ∗ Object_getRef (Object ∗this)

  Get a reference to an instance of the class Object.
- PUBLIC void **Object_deRef** (Object ∗this)

  De-reference to an instance of the class Object.
- PUBLIC int Object_isValid (Object ∗this)

  Check the pointed object is allocated.

Public Attributes

- int **marker**
- unsigned int **id**
- unsigned int **uniqId**
- Class ∗ **class**
- void(∗ **delete**)(Object ∗this)
- Object ∗(∗ **copy**)(Object ∗this)
- unsigned int **refCount**
- unsigned int **size**
- Allocator ∗ **allocator**

3.36.1 Member Function Documentation

3.36.1.1 Object_comp()

```
PUBLIC int Object_comp (
             Object * this,
             Object * compared )
```

Compare 2 instances of the class Object.

Returns

0 if O1=O2, negative if O1<O2, positive if O1>O2

3.36.1.2 Object_copy()

```
PUBLIC Object * Object_copy (
             Object * this )
```

Copy an instance of the class Object.

Returns

New instance

3.36.1.3 Object_getRef()

```
PUBLIC Object * Object_getRef (
             Object * this )
```

Get a reference to an instance of the class Object.

Returns

Reference to instance

3.36.1.4 Object_isValid()

```
PUBLIC int Object_isValid (
             Object * this )
```

Check the pointed object is allocated.

Returns

1 if valid, 0 otherwise

3.36.1.5 Object_new()

```
PUBLIC Object * Object_new (
             unsigned int size,
             Class * class )
```

Create an instance of the class Object.

Parameters

in	*Class*	to instanciate

3.36.1.6 Object_newFromAllocator()

```
PUBLIC Object * Object_newFromAllocator (
             Class * class,
             Allocator * allocator )
```

TBD.

Parameters

in	*Class*	to instanciate

3.36.1.7 Object_print()

```
PUBLIC char * Object_print (
             Object * this )
```

Print an instance of the class Object into a buffer of characters.

Returns

Buffer of characters

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/include/Object.h
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Object/Object.h
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Object/Object.c

3.37 ObjectInfo Struct Reference

Public Attributes

- Object ∗ **ptr**
- unsigned int **prevId**
- unsigned int **nextId**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/ObjectMgr/ObjectMgr.c

3.38 ObjectMgr Class Reference

Public Member Functions

- PUBLIC void **ObjectMgr_delete** (ObjectMgr ∗this)

  Delete an instance of the class ObjectMgr.
- PUBLIC ObjectMgr ∗ ObjectMgr_copy (ObjectMgr ∗this)

  Copy an instance of the class ObjectMgr.
- PUBLIC ObjectMgr ∗ ObjectMgr_getRef ()

  Get a reference to the singleton instance of ObjectMgr.
- PUBLIC unsigned int **ObjectMgr_report** (ObjectMgr ∗this)

  Reports the usage statistics for an instance of ObjectMgr.
- PUBLIC Object ∗ ObjectMgr_allocate (ObjectMgr ∗this, unsigned int size)

  Allocate a new object memory footprint of a given size.
- PUBLIC void ObjectMgr_deallocate (ObjectMgr ∗this, Object ∗object)

  De Allocate a given object.
- PUBLIC void **ObjectMgr_reportUnallocated** (ObjectMgr ∗this)

  Report objects not deallocated.

Public Attributes

- Object **object**
- unsigned int maxNbObjectAllocated
- unsigned int **allocRequestId**
- unsigned int **freeRequestId**
- unsigned int **nbAllocatedObjects**
- ObjectInfo **allocatedObjects** [MAX_NB_OBJECTS]
- unsigned int **freeSpace**
- unsigned int **usedSpace**
- unsigned int **nextId**

3.38.1 Member Function Documentation

3.38.1.1 ObjectMgr_allocate()

```
PUBLIC Object ∗ ObjectMgr_allocate (
             ObjectMgr ∗ this,
             unsigned int size )
```

Allocate a new object memory footprint of a given size.

Parameters

in	*size*	size in bytes of the memory footprint.

Returns

Reference to a instance of Object.

3.38.1.2 ObjectMgr_copy()

```
PUBLIC ObjectMgr * ObjectMgr_copy (
             ObjectMgr * this )
```

Copy an instance of the class ObjectMgr.

Returns

New instance

3.38.1.3 ObjectMgr_deallocate()

```
PUBLIC void ObjectMgr_deallocate (
             ObjectMgr * this,
             Object * object )
```

De Allocate a given object.

Parameters

in	*object*	Reference to instance of Object.

3.38.1.4 ObjectMgr_getRef()

```
PUBLIC ObjectMgr * ObjectMgr_getRef ( )
```

Get a reference to the singleton instance of ObjectMgr.

Returns

Reference to the singleton.

3.38.2 Member Data Documentation

3.38.2.1 maxNbObjectAllocated

```
unsigned int ObjectMgr::maxNbObjectAllocated
```

This is member B

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/ObjectMgr/ObjectMgr.c

3.39 ObjectStore Class Reference

Public Member Functions

- PUBLIC void ObjectStore_delete (ObjectStore ∗this)

  Delete an instance of the class ObjectMgr.
- PUBLIC ObjectStore ∗ ObjectStore_copy (ObjectStore ∗this)

  Copy an instance of the class ObjectStore.
- PUBLIC ObjectStore ∗ ObjectStore_getRef ()

  Obtain the reference to the object store.
- PUBLIC AllocInfo ∗ ObjectStore_createAllocator (ObjectStore ∗this, Allocator ∗allocator)

  Register an Allocator with the objectStore.
- PUBLIC void ObjectStore_deleteAllocator (ObjectStore ∗this, AllocInfo ∗allocInfo)

  TBD.
- PUBLIC Object ∗ ObjectStore_createObject (ObjectStore ∗this, Class ∗class, Allocator ∗allocator)

  TBD.
- PUBLIC void ObjectStore_deleteObject (ObjectStore ∗this, Object ∗object)

  Delete an object from the object store.
- PUBLIC void **ObjectStore_report** (ObjectStore ∗this)

  Reports the usage statistics for an instance of ObjectStore.
- PUBLIC unsigned int ObjectStore_getNbAllocatedObjects (ObjectStore ∗this)

  Reports the number of allocated objects in the ObjectStore.
- PUBLIC int ObjectStore_compare (ObjectStore ∗this, ObjectStore ∗compared)

  Compare 2 instances of the class ObjjectStore. Since there is only one ObjectStore instance, always return 1.
- PUBLIC void **ObjectStore_print** (ObjectStore ∗this)

  Print an instance of the class ObjectStore.

Public Attributes

- Object **object**
- unsigned int **nbAllocatedObjects**
- AllocInfo ∗ **allocList**

3.39.1 Member Function Documentation

3.39.1.1 ObjectStore_compare()

```
PUBLIC int ObjectStore_compare (
            ObjectStore ∗ this,
            ObjectStore ∗ compared )
```

Compare 2 instances of the class ObjjectStore. Since there is only one ObjectStore instance, always return 1.

Returns

0 if different, 1 if equal.

3.39.1.2 ObjectStore_copy()

```
PUBLIC ObjectStore * ObjectStore_copy (
            ObjectStore * this )
```

Copy an instance of the class ObjectStore.

Returns

Copy of the given instance.

3.39.1.3 ObjectStore_createAllocator()

```
PUBLIC AllocInfo * ObjectStore_createAllocator (
            ObjectStore * this,
            Allocator * allocator )
```

Register an Allocator with the objectStore.

TBD

3.39.1.4 ObjectStore_createObject()

```
PUBLIC Object * ObjectStore_createObject (
            ObjectStore * this,
            Class * class,
            Allocator * allocator )
```

TBD.

TBD

3.39.1.5 ObjectStore_delete()

```
PUBLIC void ObjectStore_delete (
            ObjectStore * this )
```

Delete an instance of the class ObjectMgr.

TBD

3.39.1.6 ObjectStore_deleteAllocator()

```
PUBLIC void ObjectStore_deleteAllocator (
            ObjectStore * this,
            AllocInfo * allocInfo )
```

TBD.

TBD

3.39.1.7 ObjectStore_deleteObject()

```
PUBLIC void ObjectStore_deleteObject (
            ObjectStore * this,
            Object * object )
```

Delete an object from the object store.

TBD

3.39.1.8 ObjectStore_getNbAllocatedObjects()

```
PUBLIC unsigned int ObjectStore_getNbAllocatedObjects (
            ObjectStore * this )
```

Reports the number of allocated objects in the ObjectStore.

TBD

3.39.1.9 ObjectStore_getRef()

```
PUBLIC ObjectStore * ObjectStore_getRef ( )
```

Obtain the reference to the object store.

TBD

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/ObjectStore/ObjectStore.c

3.40 OptionDefault Struct Reference

Public Attributes

- char * **name**
- char * **flag**
- char * **value**

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/OptionMgr/OptionMgr.c

3.41 OptionMgr Class Reference

Public Member Functions

- PUBLIC void **OptionMgr_delete** (OptionMgr *this)

  TBD.
- PUBLIC OptionMgr * **OptionMgr_copy** (OptionMgr *this)

  TBD.
- PUBLIC OptionMgr * OptionMgr_getRef ()

  TBD.
- PUBLIC unsigned int **OptionMgr_getSize** (OptionMgr *this)

  TBD.
- PUBLIC String * **OptionMgr_getOption** (OptionMgr *this, const char *name)

  TBD.
- PUBLIC void **OptionMgr_setOption** (OptionMgr *this, const char *optionName, String *value)

  TBD.
- PUBLIC unsigned int **OptionMgr_readFromFile** (OptionMgr *this)

  TBD.
- PUBLIC unsigned int OptionMgr_readFromCmdLine (OptionMgr *this, const int argc, const char **argv)

  TBD.

Public Attributes

- Object **object**
- Map * **options**

3.41.1 Member Function Documentation

3.41.1.1 OptionMgr_getRef()

```
PUBLIC OptionMgr * OptionMgr_getRef ( )
```

TBD.

TBD

3.41.1.2 OptionMgr_readFromCmdLine()

```
PUBLIC unsigned int OptionMgr_readFromCmdLine (
            OptionMgr * this,
            const int argc,
            const char ** argv )
```

TBD.

Parameters

in	*argc*	Number of commandline arguments.
in	*argv*	List os commandline arguments.

Returns

Status of operation.

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/OptionMgr/OptionMgr.c

3.42 PoolCache Struct Reference

Public Attributes

- unsigned int **idx**
- unsigned int **isUsed**
- void ∗ **cache**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/include/Pool.h
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Pool/Pool.h

3.43 Product Class Reference

Public Attributes

- Object **object**
- String ∗ **name**
- String ∗ **location**
- List ∗ **sources**
- List ∗ **includes**
- List ∗ **uses**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/Configuration/Product.c

3.44 SdbMgr Class Reference

Public Member Functions

- PUBLIC void **SdbMgr_delete** (SdbMgr ∗this)

  Destroy an instance of the class SdbMgr.
- PUBLIC SdbMgr ∗ SdbMgr_copy (SdbMgr ∗this)

  Create a copy of an SdbMgr object.
- PUBLIC SdbMgr ∗ SdbMgr_getRef ()

  Get a reference to an object.
- PUBLIC unsigned int SdbMgr_execute (SdbMgr ∗this, const char ∗statement, List ∗result)

  Execute a Sdb request.

Public Attributes

- Object **object**
- sqlite3 ∗ **db**
- String ∗ **name**

3.44.1 Member Function Documentation

3.44.1.1 SdbMgr_copy()

```
PUBLIC SdbMgr ∗ SdbMgr_copy (
            SdbMgr ∗ this )
```

Create a copy of an SdbMgr object.

Returns

A copy of the SdbMgr object.

3.44.1.2 SdbMgr_execute()

```
PUBLIC unsigned int SdbMgr_execute (
            SdbMgr ∗ this,
            const char ∗ statement,
            List ∗ result )
```

Execute a Sdb request.

Returns

status

3.44.1.3 SdbMgr_getRef()

```
PUBLIC SdbMgr ∗ SdbMgr_getRef ( )
```

Get a reference to an object.

Returns

A reference to a SdbMgr object.

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/SdbMgr/SdbMgr.c

3.45 SdbRequest Class Reference

Public Member Functions

- PUBLIC SdbRequest ∗ SdbRequest_new (const char ∗fmt)

  Create a new SdbRequest instance
  @parameter SQL statement template.
- PUBLIC void SdbRequest_delete (SdbRequest ∗this)

  Create a new SdbRequest instance
  @parameter SQL statement template.
- PUBLIC void SdbRequest_execute (SdbRequest ∗this,...)

  Execute a SdbRequest
  @parameter Variable list of parameter to use with SQL template.

Public Attributes

- Object **object**
- char ∗ **buffer**
- unsigned int **size**
- const char ∗ **fmt**
- List ∗ **result**
- unsigned int **nbResults**
- unsigned int **nbColumns**

3.45.1 Member Function Documentation

3.45.1.1 SdbRequest_delete()

```
PUBLIC void SdbRequest_delete (
             SdbRequest ∗ this )
```

Create a new SdbRequest instance

@parameter SQL statement template.

Returns

Instance of an SdbRequest

3.45.1.2 SdbRequest_execute()

```
PUBLIC void SdbRequest_execute (
             SdbRequest ∗ this,
              ...  )
```

Execute a SdbRequest

@parameter Variable list of parameter to use with SQL template.

Returns

Instance of an SdbRequest

3.45.1.3 SdbRequest_new()

```
PUBLIC SdbRequest * SdbRequest_new (
             const char * fmt )
```

Create a new SdbRequest instance

@parameter SQL statement template.

Returns

Instance of an SdbRequest

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/SdbMgr/SdbRequest.c

3.46 SkipList Class Reference

Public Member Functions

- PUBLIC SkipList ∗ SkipList_new ()

  Create a new instance of the class SkipList.
- PUBLIC SkipList ∗ SkipList_newFromAllocator (Allocator ∗allocator)

  Create a new instance of the clss SkipLIst from an specified allocator.
- PUBLIC void SkipList_delete (SkipList ∗this)

  SkipList_delete.
- PUBLIC SkipList ∗ SkipList_copy (SkipList ∗this)

  SkipList_copy.
- PUBLIC void SkipList_add (SkipList ∗this, Object ∗key, Object ∗item)

  SkipList_add.
- PUBLIC Object ∗ SkipList_remove (SkipList ∗this, Object ∗key)

  SkipList_remove.
- PUBLIC int SkipList_compare (SkipList ∗this, SkipList ∗compared)

  SkipList_compare.
- PUBLIC void SkipList_print (SkipList ∗this)

  SkipList_print.
- PUBLIC unsigned int SkipList_getSize (SkipList ∗this)

  SkipList_getSize.

Public Attributes

- Object **object**
- unsigned int **level**
- unsigned int **nbObjects**
- unsigned int **pack**
- void ∗ **headerPtr**
- void ∗ **endPtr**

3.46.1 Member Function Documentation

3.46.1.1 SkipList_add()

```
PUBLIC void SkipList_add (
            SkipList * this,
            Object * key,
            Object * item )
```

SkipList_add.

Parameters

in	*Key*	to index object
in	*Object*	to add to SkipList object.

Returns

None

3.46.1.2 SkipList_compare()

```
PUBLIC int SkipList_compare (
             SkipList * this,
             SkipList * compared )
```

SkipList_compare.

Parameters

in	*Instance*	to be compared to.

Returns

0 if equal, <0 if S1<S2, >0 if S1>S2

3.46.1.3 SkipList_copy()

```
PUBLIC SkipList * SkipList_copy (
             SkipList * this )
```

SkipList_copy.

Parameters

in	*Instance*	to copy

Returns

A copy of the SkipList instance.

3.46.1.4 SkipList_delete()

```
PUBLIC void SkipList_delete (
             SkipList * this )
```

SkipList_delete.

Parameters

in	*Instance*	to destroy

Returns

None

3.46.1.5 SkipList_getSize()

```
PUBLIC unsigned int SkipList_getSize (
            SkipList * this )
```

SkipList_getSize.

Parameters

in	*None*	

Returns

None

3.46.1.6 SkipList_new()

```
PUBLIC SkipList * SkipList_new ( )
```

Create a new instance of the class SkipList.

Parameters

in	*none*	

Returns

New instance of class SkipList.

3.46.1.7 SkipList_newFromAllocator()

```
PUBLIC SkipList * SkipList_newFromAllocator (
            Allocator * allocator )
```

Create a new instance of the clss SkipLIst from an specified allocator.

Parameters

in	*none*	

Returns

New instance of class SkipList.

3.46.1.8 SkipList_print()

```
PUBLIC void SkipList_print (
             SkipList * this )
```

SkipList_print.

Parameters

in	*None*	

Returns

None

3.46.1.9 SkipList_remove()

```
PUBLIC Object * SkipList_remove (
             SkipList * this,
             Object * key )
```

SkipList_remove.

Parameters

in	*Key*	of object to remove

Returns

Object removed from SkipList object.

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/SkipList/SkipList.c

3.47 SkipNode Struct Reference

Public Attributes

- Object **object**
- Object * **key**
- Object * **item**
- unsigned int **level**
- void * **forward** [SKIPLIST_MAX_LEVEL]

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/SkipList/SkipNode.h

3.48 SParse Class Reference

Public Member Functions

- PUBLIC SParse ∗ SParse_new (String ∗sdbName)

    Create a new SParse object.
- PUBLIC void SParse_delete (SParse ∗this)

    Delete a SParse object.
- PUBLIC SParse ∗ SParse_copy (SParse ∗this)

    Copy a SParse object instance.
- PUBLIC void **SParse_print** (SParse ∗this)

    Print a SParse object.
- PUBLIC unsigned int SParse_parse (SParse ∗this, const char ∗extension)

    Parse all files with a given extension.

Public Attributes

- Object **object**
- Configuration ∗ **configuration**
- char ∗ **extension**
- SdbMgr ∗ **sdbMgr**

3.48.1 Member Function Documentation

3.48.1.1 SParse_copy()

```
PUBLIC SParse * SParse_copy (
          SParse * this )
```

Copy a SParse object instance.

Returns

Copy of instance.

3.48.1.2 SParse_delete()

```
PUBLIC void SParse_delete (
          SParse * this )
```

Delete a SParse object.

Parameters

Object	to delete.

3.48.1.3 SParse_new()

```
PUBLIC SParse * SParse_new (
            String * sdbName )
```

Create a new SParse object.

Returns

New SParse object.

3.48.1.4 SParse_parse()

```
PUBLIC unsigned int SParse_parse (
            SParse * this,
            const char * extension )
```

Parse all files with a given extension.

Parameters

in	*extension*	Extension of the files to parse.

Returns

Status of the operation.

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/SParse/SParse.c

3.49 String Struct Reference

Public Member Functions

- PUBLIC void **String_delete** (String ∗this)

    Delete an instance of class String.
- PUBLIC String ∗ String_copy (String ∗this)

    Copy an instance of class String.
- PUBLIC String ∗ String_getRef (String ∗this)

    Copy an instance of class String.
- PUBLIC int String_compare (String ∗this, String ∗compared)

    Compare this String with another String.
- PUBLIC String ∗ **String_subString** (String ∗this, unsigned int idx, unsigned int length)

    TBD.
- PUBLIC int **String_toInt** (String ∗this)

    TBD.

- PUBLIC unsigned int **String_getLength** (String ∗this)

  TBD.
- PUBLIC char ∗ **String_getBuffer** (String ∗this)

  TBD.
- PUBLIC void **String_setBuffer** (String ∗this, char ∗buffer, int isOwned)

  TBD.
- PUBLIC unsigned int **String_isContained** (String ∗this, String ∗s2)

  TBD.

Public Attributes

- Object **object**
- int **isOwned**
- char ∗ **buffer**
- unsigned int **length**

3.49.1 Detailed Description

/file String2.c

/brief The String class provide a dynamic array of char terminated by 0.

The class String is a container for text data. /class String

3.49.2 Member Function Documentation

3.49.2.1 String_compare()

```
PUBLIC int String_compare (
            String ∗ this,
            String ∗ compared )
```

Compare this String with another String.

Parameters

in	*compared*	String to compare

Returns

0 if S1=S2, negative if S1<S2, positive if S1>S2

3.49.2.2 String_copy()

```
PUBLIC String ∗ String_copy (
            String ∗ this )
```

Copy an instance of class String.

Returns

Copy of instance.

3.49.2.3 String_getRef()

```
PUBLIC String * String_getRef (
            String * this )
```

Copy an instance of class String.

Returns

Copy of instance.

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/String/String2.c

3.50 stub_data Struct Reference

Public Attributes

- void * **malloc_result**
- int **malloc_nb_calls**
- int **free_nb_calls**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Allocator/tests/Stub_Malloc.c

3.51 Task Struct Reference

Public Member Functions

- PUBLIC Task * Task_create (void *(*body)(void *p), int nbParams, void **params)

  Create a task object.
- PUBLIC void **Task_destroy** (Task *this)

  Destroy a task object.
- PUBLIC int Task_isReady (Task *this)

  Obtain the status of the ready flag.
- PUBLIC void **Task_setReady** (Task *this)

  Mark a task as ready.
- PUBLIC int Task_isRunning (Task *this)

  Obtain the status of the running flag.
- PUBLIC void **Task_setRunning** (Task *this)

  Mark a task as running.
- PUBLIC int Task_isCompleted (Task *this)

  Obtain the status of the completion flag.
- PUBLIC void **Task_setCompleted** (Task *this)

  Mark a task as completed.
- PUBLIC void **Task_start** (Task *this)

  Start a Task.
- PUBLIC void Task_executeBody (Task *this)

  Execute the body of the task.

Public Attributes

- void ∗(∗ **body**)(void ∗p)
- int **nbParams**
- void ∗ **params** [5]
- int **isReady**
- int **isRunning**
- int **isCompleted**
- int **execTime**

3.51.1 Member Function Documentation

3.51.1.1 Task_create()

```
PUBLIC Task * Task_create (
             void *(*)(void *p) body,
             int nbParams,
             void ** params )
```

Create a task object.

Returns

The new instance of a task.

3.51.1.2 Task_executeBody()

```
PUBLIC void Task_executeBody (
             Task * this )
```

Execute the body of the task.

Returns

TBD

3.51.1.3 Task_isCompleted()

```
PUBLIC int Task_isCompleted (
             Task * this )
```

Obtain the status of the completion flag.

Returns

Completion flag.

3.51.1.4 Task_isReady()

```
PUBLIC int Task_isReady (
             Task * this )
```

Obtain the status of the ready flag.

Returns

Ready flag.

3.51.1.5 Task_isRunning()

```
PUBLIC int Task_isRunning (
             Task * this )
```

Obtain the status of the running flag.

Returns

Running flag.

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/TaskMgr/Task.c

3.52 TaskMgr Class Reference

Public Member Functions

- PUBLIC TaskMgr ∗ TaskMgr_getRef ()

  Get reference to singleton TaskMgr.
- PUBLIC void TaskMgr_delete (TaskMgr ∗this)

  TBD.
- PUBLIC int TaskMgr_start (TaskMgr ∗this, Task ∗task)

  Queue a task for execution.
- PUBLIC void TaskMgr_stop (TaskMgr ∗this)

  Request all worker threads to stop.
- PUBLIC void **TaskMgr_print** (TaskMgr ∗this)

  Print the content of the TaskMgr.
- PUBLIC unsigned int TaskMgr_getSize (TaskMgr ∗this)

  TBD.
- PRIVATE void TaskMgr_waitForThread (TaskMgr ∗this)

  TBD.
- PRIVATE int TaskMgr_createWorkerThreads (TaskMgr ∗this)

  Create all worker threads.
- PRIVATE int TaskMgr_findAvailableTask (TaskMgr ∗this)

  TBD.
- PRIVATE int TaskMgr_initSemaphores (TaskMgr ∗this)

Initialise empty and full semaphores.

- PRIVATE int TaskMgr_destroySemaphores (TaskMgr ∗this)

  Destroy empty and full semaphores.
- PRIVATE int TaskMgr_waitNotFull (TaskMgr ∗this)

  Wait until there is a space to add a task.
- PRIVATE int TaskMgr_waitNotEmpty (TaskMgr ∗this)

  Wait until there is a task in the queue.
- PRIVATE int TaskMgr_signalNotFull (TaskMgr ∗this)

  Signal that task can be added to the queue.
- PRIVATE int TaskMgr_signalNotEmpty (TaskMgr ∗this)

  Signal that there are task to process.

Public Attributes

- Object **object**
- int **nbThreads**
- Task ∗ **taskId** [MAX_TASKS]
- pthread_t **threadHandle** [MAX_THREADS]
- sem_t **semEmpty**
- sem_t **semFull**
- pthread_mutex_t **mutex**
- int **isStopping**

3.52.1 Member Function Documentation

3.52.1.1 TaskMgr_createWorkerThreads()

```
PRIVATE int TaskMgr_createWorkerThreads (
             TaskMgr ∗ this )
```

Create all worker threads.

Returns

TBD

3.52.1.2 TaskMgr_delete()

```
PUBLIC void TaskMgr_delete (
             TaskMgr ∗ this )
```

TBD.

Returns

none

3.52.1.3 TaskMgr_destroySemaphores()

```
PRIVATE int TaskMgr_destroySemaphores (
             TaskMgr * this )
```

Destroy empty and full semaphores.

Returns

1 indicates if operation was successful.

3.52.1.4 TaskMgr_findAvailableTask()

```
PRIVATE int TaskMgr_findAvailableTask (
             TaskMgr * this )
```

TBD.

Returns

TBD

3.52.1.5 TaskMgr_getRef()

```
PUBLIC TaskMgr * TaskMgr_getRef ( )
```

Get reference to singleton TaskMgr.

Returns

Reference to the TaskMgr.

3.52.1.6 TaskMgr_getSize()

```
PUBLIC unsigned int TaskMgr_getSize (
             TaskMgr * this )
```

TBD.

Parameters

in	*TBD*	

Returns

TBD

3.52.1.7 TaskMgr_initSemaphores()

```
PRIVATE int TaskMgr_initSemaphores (
            TaskMgr * this )
```

Initialise empty and full semaphores.

Returns

1 indicates if operation was successful.

3.52.1.8 TaskMgr_signalNotEmpty()

```
PRIVATE int TaskMgr_signalNotEmpty (
            TaskMgr * this )
```

Signal that there are task to process.

Returns

1 indicates if operation was successful.

3.52.1.9 TaskMgr_signalNotFull()

```
PRIVATE int TaskMgr_signalNotFull (
            TaskMgr * this )
```

Signal that task can be added to the queue.

Returns

1 indicates if operation was successful.

3.52.1.10 TaskMgr_start()

```
PUBLIC int TaskMgr_start (
            TaskMgr * this,
            Task * task )
```

Queue a task for execution.

Parameters

in	*task*	

Returns

1 if successful.

3.52.1.11 TaskMgr_stop()

```
PUBLIC void TaskMgr_stop (
            TaskMgr * this )
```

Request all worker threads to stop.

Returns

1 if successful.

3.52.1.12 TaskMgr_waitForThread()

```
PRIVATE void TaskMgr_waitForThread (
            TaskMgr * this )
```

TBD.

Parameters

in	*TBD*	

Returns

TBD

3.52.1.13 TaskMgr_waitNotEmpty()

```
PRIVATE int TaskMgr_waitNotEmpty (
            TaskMgr * this )
```

Wait until there is a task in the queue.

Returns

1 indicates if operation was successful.

3.52.1.14 TaskMgr_waitNotFull()

```
PRIVATE int TaskMgr_waitNotFull (
            TaskMgr * this )
```

Wait until there is a space to add a task.

Returns

1 indicates if operation was successful.

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/TaskMgr/TaskMgr.c

3.53 TestClass Struct Reference

Public Attributes

- Object **object**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/ObjectStore/tests/UT_ObjectStore.c

3.54 TestFileMgr Struct Reference

Public Attributes

- Object **object**
- List ∗ **files**
- List ∗ **directories**
- char ∗ **separator**
- String ∗ **rootLocation**

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/FileMgr/tests/UT_FileMgr_01.c

3.55 TestObject Struct Reference

Public Attributes

- Object **object**
- int **id**
- int **testValue**

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

- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Array/tests/UT_Array_01.c
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/BTree/tests/TestObject.c
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/List/tests/TestObject.c
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/Map/tests/TestObject.c
- /home/thomas/Projects/SParse-master/SParse/src/CommonLib/SkipList/tests/TestObject.c

3.56 testOptionMgr Struct Reference

Public Attributes

- Object **object**
- Map ∗ **options**

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/OptionMgr/tests/UT_OptionMgr_01.c

3.57 TestSdbMgr Struct Reference

Public Attributes

- Object **object**

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/SdbMgr/tests/UT_SdbMgr_01.c

3.58 TestTimeMgr Struct Reference

Public Attributes

- Object **object**
- Map ∗ **timers**

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/TimeMgr/tests/UT_TimeMgr_01.c

3.59 TimeMgr Class Reference

Public Member Functions

- PUBLIC void TimeMgr_delete (TimeMgr ∗this)

  Delete a TimeMgr object.
- PUBLIC TimeMgr ∗ TimeMgr_copy (TimeMgr ∗this)

  Copy an instance of the class TimeMgr.
- PUBLIC TimeMgr ∗ TimeMgr_getRef ()

  Get a reference to the singleton instance of TimeMgr.
- PUBLIC unsigned int TimeMgr_getSize (TimeMgr ∗this)

  Provide the size of the class or an instance.
- PUBLIC void TimeMgr_latchTime (TimeMgr ∗this, String ∗s)

  Latch the current time under the specified name.

Public Attributes

- Object **object**
- Map ∗ **timers**

3.59.1 Member Function Documentation

3.59.1.1 TimeMgr_copy()

```
PUBLIC TimeMgr * TimeMgr_copy (
             TimeMgr * this )
```

Copy an instance of the class TimeMgr.

Returns

New instance

3.59.1.2 TimeMgr_delete()

```
PUBLIC void TimeMgr_delete (
             TimeMgr * this )
```

Delete a TimeMgr object.

Parameters

Object	to delete.

3.59.1.3 TimeMgr_getRef()

```
PUBLIC TimeMgr * TimeMgr_getRef ( )
```

Get a reference to the singleton instance of TimeMgr.

Returns

Reference to the singleton.

3.59.1.4 TimeMgr_getSize()

```
PUBLIC unsigned int TimeMgr_getSize (
             TimeMgr * this )
```

Provide the size of the class or an instance.

Returns

Size in byte

3.59.1.5 TimeMgr_latchTime()

```
PUBLIC void TimeMgr_latchTime (
             TimeMgr * this,
             String * s )
```

Latch the current time under the specified name.

Parameters

name	of the timer to create

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/TimeMgr/TimeMgr.c

3.60 Timer Class Reference

Public Member Functions

- PUBLIC Timer ∗ Timer_new (String ∗name)

  Create an instance of the class Timer.
- PUBLIC void **Timer_delete** (Timer ∗this)

  Delete an instance of the class Timer.
- PUBLIC Timer ∗ Timer_copy (Timer ∗this)

  Copy an instance of the class Timer.
- PUBLIC unsigned int **Timer_getSize** (Timer ∗this)

  TBD.
- PUBLIC unsigned int **Timer_isEqual** (Timer ∗this, Timer ∗compared)

  TBD.
- PUBLIC void **Timer_print** (Timer ∗this)

  TBD.
- PUBLIC void **Timer_latchTime** (Timer ∗this)

  TBD.

Public Attributes

- Object **object**
- String ∗ **name**
- unsigned int **state**
- unsigned int **nbCalls**
- long double **cpuDurationS**
- long double **wallDurationS**
- long double **cpuLatchedTimeS**
- long double **wallLatchedTimeS**

3.60.1 Member Function Documentation

3.60.1.1 Timer_copy()

```
PUBLIC Timer ∗ Timer_copy (
          Timer ∗ this )
```

Copy an instance of the class Timer.

Returns

Copied instance.

3.60.1.2 Timer_new()

```
PUBLIC Timer * Timer_new (
            String * name )
```

Create an instance of the class Timer.

Returns

New instance.

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

- /home/thomas/Projects/SParse-master/SParse/src/AppliLib/TimeMgr/Timer.c

3.61 TransUnit Class Reference

Public Member Functions

- PUBLIC TransUnit ∗ TransUnit_new (FileDesc ∗file, FileMgr ∗fileMgr)

  Create a new TransUnit object.
- PUBLIC void **TransUnit_delete** (TransUnit ∗this)

  Delete an instance of a TransUnit object.
- PUBLIC void **TransUnit_print** (TransUnit ∗this)

  Print a new TransUnit object.
- PUBLIC unsigned int TransUnit_getSize (TransUnit ∗this)

  Returns the size a new TransUnit object.
- PUBLIC char ∗ TransUnit_getName (TransUnit ∗this)

  Returns the filename a new TransUnit object.
- PUBLIC String ∗ TransUnit_getNextBuffer (TransUnit ∗this)

  Returns the buffer of a new TransUnit object.

Public Attributes

- Object **object**
- FileDesc ∗ **file**
- FileMgr ∗ **fm**
- List ∗ **buffers**
- MacroStore ∗ **store**
- Buffer ∗ **currentBuffer**
- int **nbCharRead**
- char ∗ **outputBuffer**
- int **outputBufferSize**
- int **nbCharWritten**

3.61.1 Member Function Documentation

3.61.1.1 TransUnit_getName()

```
PUBLIC char * TransUnit_getName (
             TransUnit * this )
```

Returns the filename a new TransUnit object.

Returns

Filename

3.61.1.2 TransUnit_getNextBuffer()

```
PUBLIC String * TransUnit_getNextBuffer (
             TransUnit * this )
```

Returns the buffer of a new TransUnit object.

Returns

Buffer

3.61.1.3 TransUnit_getSize()

```
PUBLIC unsigned int TransUnit_getSize (
             TransUnit * this )
```

Returns the size a new TransUnit object.

Returns

Size in bytes.

3.61.1.4 TransUnit_new()

```
PUBLIC TransUnit * TransUnit_new (
             FileDesc * file,
             FileMgr * fileMgr )
```

Create a new TransUnit object.

Returns

Created TransUnit instance.

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/TransUnit/TransUnit.c

3.62 yy_buffer_state Struct Reference

Public Attributes

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

3.62.1 Member Data Documentation

3.62.1.1 yy_bs_column

```
int yy_buffer_state::yy_bs_column
```

The column count.

3.62.1.2 yy_bs_lineno

```
int yy_buffer_state::yy_bs_lineno
```

The line count.

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/Grammar2/Grammar2.lex.c
- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/GrammarC99/GrammarC99.lex.c

3.63 yy_trans_info Struct Reference

Public Attributes

- flex_int16_t **yy_verify**
- flex_int16_t **yy_nxt**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/Grammar2/Grammar2.lex.c
- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/GrammarC99/GrammarC99.lex.c

3.64 yyalloc Union Reference

Public Attributes

- yy_state_t **yyss_alloc**
- YYSTYPE **yyvs_alloc**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/Grammar2/Grammar2.parse.c
- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/GrammarC99/GrammarC99.parse.c

3.65 yyguts_t Struct Reference

Public Attributes

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

3.65.1 Member Data Documentation

3.65.1.1 yy_buffer_stack

```
YY_BUFFER_STATE ∗ yyguts_t::yy_buffer_stack
```

Stack as an array.

3.65.1.2 yy_buffer_stack_max

```
size_t yyguts_t::yy_buffer_stack_max
```

capacity of stack.

3.65.1.3 yy_buffer_stack_top

```
size_t yyguts_t::yy_buffer_stack_top
```

index of top of stack.

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/Grammar2/Grammar2.lex.c
- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/GrammarC99/GrammarC99.lex.c

3.66 YYSTYPE Union Reference

Public Attributes

- String $*$ **text**

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

- /home/thomas/Projects/SParse-master/SParse/src/ParseLib/Grammar2/Grammar2.parse.h

Chapter 4

File Documentation

4.1 /home/thomas/Projects/SParse-master/SParse/src/AppliLib/FileMgr/↩ FileDesc.c File Reference

The FileDesc class describe a File in the FIleMgr.

```
#include "FileDesc.h"
#include "String2.h"
#include "FileIo.h"
#include "Class.h"
#include "Object.h"
#include "Memory.h"
```

Classes

- class FileDesc

Functions

- PRIVATE String ∗ **FileDesc_getBasename** (FileDesc ∗this)
- PUBLIC unsigned int **FileDesc_getSize** (FileDesc ∗this)

4.1.1 Detailed Description

The FileDesc class describe a File in the FIleMgr.

The class FileDesc is TBD

4.2 FileDesc.h

```
/* FileDesc.h */

#ifndef _FILEDESC_H_
#define _FILEDESC_H_

#include "Types.h"
#include "String2.h"

typedef struct FileDesc FileDesc;

PUBLIC FileDesc * FileDesc_new();
PUBLIC void FileDesc_delete(FileDesc * this);
PUBLIC FileDesc * FileDesc_copy(FileDesc * this);
PUBLIC unsigned int FileDesc_getSize(FileDesc* this);
PUBLIC void FileDesc_setFullName(FileDesc * this, String * fullName);
PUBLIC String * FileDesc_getFullName(FileDesc * this);
PUBLIC void FileDesc_setName(FileDesc * this, String * name);
PUBLIC String * FileDesc_getName(FileDesc * this);
PUBLIC String * FileDesc_load(FileDesc * this);

#endif /* _FILEDESC_H_ */
```

4.3 FileDesc.h

```
/* FileDesc.h */

#ifndef _FILEDESC_H_
#define _FILEDESC_H_

#include "Types.h"
#include "String2.h"

typedef struct FileDesc FileDesc;

PUBLIC FileDesc * FileDesc_new();
PUBLIC void FileDesc_delete(FileDesc * this);
PUBLIC FileDesc * FileDesc_copy(FileDesc * this);
PUBLIC unsigned int FileDesc_getSize(FileDesc* this);
PUBLIC void FileDesc_setFullName(FileDesc * this, String * fullName);
PUBLIC String * FileDesc_getFullName(FileDesc * this);
PUBLIC void FileDesc_setName(FileDesc * this, String * name);
PUBLIC String * FileDesc_getName(FileDesc * this);
PUBLIC String * FileDesc_load(FileDesc * this);

#endif /* _FILEDESC_H_ */
```

4.4 /home/thomas/Projects/SParse-master/SParse/src/AppliLib/FileMgr/↩ FileMgr.c File Reference

The FileMgr class manages a list of files contained in a group of locations.

```
#include "FileMgr.h"
#include "String2.h"
#include "Class.h"
#include "Object.h"
#include "List.h"
#include "FileDesc.h"
#include "Memory.h"
#include "Error.h"
#include "Debug.h"
#include "FileIo.h"
```

Classes

- class FileMgr

Macros

- #define **DEBUG** (0)
- #define **FILEMGR_MAX_PATH** (1024)

Functions

- PRIVATE void **FileMgr_listFiles** (FileMgr ∗this, String ∗directory)
- PRIVATE unsigned int **FileMgr_existFS** (FileMgr ∗this, String ∗fullName)
- PUBLIC void **FileMgr_print** (FileMgr ∗this)
- PUBLIC FileDesc ∗ **FileMgr_createFile** (FileMgr ∗this, const char ∗fileName)
- PUBLIC FileDesc ∗ **FileMgr_searchFile** (FileMgr ∗this, String ∗name, List ∗preferredDir)
- PUBLIC FileDesc ∗ **FileMgr_isManaged** (FileMgr ∗this, String ∗fullName)

4.4.1 Detailed Description

The FileMgr class manages a list of files contained in a group of locations.

The class FileMgr is TBD

4.5 FileMgr.h

```
/* FileMgr.h */

#ifndef _FILEMGR_H_
#define _FILEMGR_H_

#include "Types.h"
#include "List.h"
#include "String2.h"
#include "FileDesc.h"

typedef struct FileMgr FileMgr;

PUBLIC FileMgr* FileMgr_new();
PUBLIC void FileMgr_delete(FileMgr * this);
PUBLIC FileMgr * FileMgr_copy(FileMgr * this);
PUBLIC void FileMgr_print(FileMgr * this);
PUBLIC String* FileMgr_load(FileMgr* this, const char * fileName);
PUBLIC void FileMgr_write(FileMgr* this, const char* fileName, String* buffer);
PUBLIC void FileMgr_close(FileMgr* this, String* fileName);
PUBLIC unsigned int FileMgr_setRootLocation(FileMgr* this, const char * location);
PUBLIC char * FileMgr_getRootLocation(FileMgr* this);
PUBLIC FileMgr* FileMgr_getRef();
PUBLIC unsigned int FileMgr_getSize(FileMgr * this);
PUBLIC unsigned int FileMgr_addDirectory(FileMgr * this, const char * directoryName);
PUBLIC FileDesc * FileMgr_addFile(FileMgr * this, const char * fileName);
PUBLIC FileDesc* FileMgr_createFile(FileMgr* this, const char* fileName);
PUBLIC List * FileMgr_filterFiles(FileMgr * this, const char * pattern);
PUBLIC FileDesc * FileMgr_searchFile(FileMgr * this, String * name, List * preferredDir);
PUBLIC FileDesc* FileMgr_isManaged(FileMgr* this, String* fullName);
#endif /* _FILEMGR_H_ */
```

4.6 FileMgr.h

```
/* FileMgr.h */

#ifndef _FILEMGR_H_
#define _FILEMGR_H_

#include "Types.h"
#include "List.h"
#include "String2.h"
#include "FileDesc.h"

typedef struct FileMgr FileMgr;

PUBLIC FileMgr* FileMgr_new();
PUBLIC void FileMgr_delete(FileMgr * this);
PUBLIC FileMgr * FileMgr_copy(FileMgr * this);
PUBLIC void FileMgr_print(FileMgr * this);
PUBLIC String* FileMgr_load(FileMgr* this, const char * fileName);
PUBLIC void FileMgr_write(FileMgr* this, const char* fileName, String* buffer);
PUBLIC void FileMgr_close(FileMgr* this, String* fileName);
PUBLIC unsigned int FileMgr_setRootLocation(FileMgr* this, const char * location);
PUBLIC char * FileMgr_getRootLocation(FileMgr* this);
PUBLIC FileMgr* FileMgr_getRef();
PUBLIC unsigned int FileMgr_getSize(FileMgr * this);
PUBLIC unsigned int FileMgr_addDirectory(FileMgr * this, const char * directoryName);
PUBLIC FileDesc * FileMgr_addFile(FileMgr * this, const char * fileName);
PUBLIC FileDesc* FileMgr_createFile(FileMgr* this, const char* fileName);
PUBLIC List * FileMgr_filterFiles(FileMgr * this, const char * pattern);
PUBLIC FileDesc * FileMgr_searchFile(FileMgr * this, String * name, List * preferredDir);
PUBLIC FileDesc* FileMgr_isManaged(FileMgr* this, String* fullName);
#endif /* _FILEMGR_H_ */
```

4.7 /home/thomas/Projects/SParse-master/SParse/src/AppliLib/Option↩ Mgr/OptionMgr.c File Reference

The OptionMgr class manages the application configuration.

```
#include "OptionMgr.h"
#include "Class.h"
#include "Object.h"
#include "String2.h"
#include "Map.h"
#include "FileMgr.h"
#include "Memory.h"
#include "Error.h"
#include "Debug.h"
```

Classes

- class OptionMgr
- struct OptionDefault

Functions

- PRIVATE unsigned int **OptionMgr_parseFile** (OptionMgr ∗this, String ∗fileContent)

4.7.1 Detailed Description

The OptionMgr class manages the application configuration.

The class OptionMgr is TBD

4.8 OptionMgr.h

```
/* OptionMgr.h */

#ifndef _OPTIONMGR_H_
#define _OPTIONMGR_H_

#include "Types.h"
#include "String2.h"

typedef struct OptionMgr OptionMgr;

PUBLIC void OptionMgr_delete(OptionMgr * this);
PUBLIC OptionMgr * OptionMgr_copy(OptionMgr * this);
PUBLIC PUBLIC OptionMgr* OptionMgr_getRef();
PUBLIC unsigned int OptionMgr_getSize(OptionMgr * this);
PUBLIC String * OptionMgr_getOption(OptionMgr * this, const char * name);
PUBLIC void OptionMgr_setOption(OptionMgr * this, const char * optionName, String * value);
PUBLIC unsigned int OptionMgr_readFromFile(OptionMgr * this);
PUBLIC unsigned int OptionMgr_readFromCmdLine(OptionMgr * this, const int argc, const char ** argv);
PUBLIC unsigned int OptionMgr_isOptionEnabled(OptionMgr* this, const char * optionName);

#endif /* _OPTIONMGR_H_ */
```

4.9 OptionMgr.h

```
/* OptionMgr.h */

#ifndef _OPTIONMGR_H_
#define _OPTIONMGR_H_

#include "Types.h"
#include "String2.h"

typedef struct OptionMgr OptionMgr;

PUBLIC void OptionMgr_delete(OptionMgr * this);
PUBLIC OptionMgr * OptionMgr_copy(OptionMgr * this);
PUBLIC PUBLIC OptionMgr* OptionMgr_getRef();
PUBLIC unsigned int OptionMgr_getSize(OptionMgr * this);
PUBLIC String * OptionMgr_getOption(OptionMgr * this, const char * name);
PUBLIC void OptionMgr_setOption(OptionMgr * this, const char * optionName, String * value);
PUBLIC unsigned int OptionMgr_readFromFile(OptionMgr * this);
PUBLIC unsigned int OptionMgr_readFromCmdLine(OptionMgr * this, const int argc, const char ** argv);
PUBLIC unsigned int OptionMgr_isOptionEnabled(OptionMgr* this, const char * optionName);

#endif /* _OPTIONMGR_H_ */
```

4.10 /home/thomas/Projects/SParse-master/SParse/src/AppliLib/Sdb↩ Mgr/SdbMgr.c File Reference

TBD.

```
#include ¨SdbMgr.h¨
#include ¨Class.h¨
#include ¨Object.h¨
#include ¨String2.h¨
#include ¨Memory.h¨
#include ¨Error.h¨
#include ¨List.h¨
#include <sqlite3.h>
```

Classes

- class SdbMgr

Functions

- PRIVATE unsigned int **SdbMgr_open** (SdbMgr ∗this, String ∗sdbName)
- PRIVATE void **SdbMgr_close** (SdbMgr ∗this)
- PUBLIC unsigned int **SdbMgr_getSize** (SdbMgr ∗this)

Variables

- PRIVATE SdbMgr ∗ **sdbMgr** = 0

4.10.1 Detailed Description

TBD.

TBD

4.11 SdbMgr.h

```
/* SdbMgr.h */

#ifndef _SDBMGR_H_
#define _SDBMGR_H_

#include "Types.h"
#include "String2.h"
#include "List.h"

typedef struct SdbMgr SdbMgr;

PUBLIC SdbMgr * SdbMgr_new(String * name);
PUBLIC void SdbMgr_delete(SdbMgr* this);
PUBLIC SdbMgr * SdbMgr_copy(SdbMgr* this);
PUBLIC SdbMgr * SdbMgr_getRef();
PUBLIC unsigned int SdbMgr_getSize(SdbMgr* this);
PUBLIC unsigned int SdbMgr_execute(SdbMgr* this, const char* statement, List * result);

#endif /* _SDBMGR_H_ */
```

4.12 SdbMgr.h

```
/* SdbMgr.h */

#ifndef _SDBMGR_H_
#define _SDBMGR_H_

#include "Types.h"
#include "String2.h"
#include "List.h"

typedef struct SdbMgr SdbMgr;

PUBLIC SdbMgr * SdbMgr_new(String * name);
PUBLIC void SdbMgr_delete(SdbMgr* this);
PUBLIC SdbMgr * SdbMgr_copy(SdbMgr* this);
PUBLIC SdbMgr * SdbMgr_getRef();
PUBLIC unsigned int SdbMgr_getSize(SdbMgr* this);
PUBLIC unsigned int SdbMgr_execute(SdbMgr* this, const char* statement, List * result);

#endif /* _SDBMGR_H_ */
```

4.13 SdbRequest.h

```
/* SdbRequest.h */
#ifndef _SDBREQUEST_H_
#define _SDBREQUEST_H_

#include "Types.h"
#include "List.h"

typedef struct SdbRequest SdbRequest;

PUBLIC SdbRequest * SdbRequest_new(const char * fmt);
PUBLIC void SdbRequest_delete(SdbRequest * this);
PUBLIC SdbRequest * SdbRequest_copy(SdbRequest * this);
PUBLIC unsigned int SdbRequest_getSize(SdbRequest * this);
PUBLIC void SdbRequest_execute(SdbRequest * this, ...);
PUBLIC unsigned int SdbRequest_getNbResult(SdbRequest * this);
PUBLIC List * SdbRequest_getResults(SdbRequest * this);

#endif /* _SDBREQUEST_H_ */
```

4.14 SdbRequest.h

```
/* SdbRequest.h */
#ifndef _SDBREQUEST_H_
#define _SDBREQUEST_H_

#include "Types.h"
#include "List.h"

typedef struct SdbRequest SdbRequest;

PUBLIC SdbRequest * SdbRequest_new(const char * fmt);
PUBLIC void SdbRequest_delete(SdbRequest * this);
PUBLIC SdbRequest * SdbRequest_copy(SdbRequest * this);
PUBLIC unsigned int SdbRequest_getSize(SdbRequest * this);
PUBLIC void SdbRequest_execute(SdbRequest * this, ...);
PUBLIC unsigned int SdbRequest_getNbResult(SdbRequest * this);
PUBLIC List * SdbRequest_getResults(SdbRequest * this);

#endif /* _SDBREQUEST_H_ */
```

4.15 Storage.h

```
/* Storage.h */
#ifndef _STORAGE_H_
#define _STORAGE_H_

#include "Types.h"

typedef struct Storage Storage;

PUBLIC Storage * Storage_new();
PUBLIC void Storage_delete(Storage * this);
// "SELECT * FROM Include_Nodes WHERE Name='%s'
// "SELECT * FROM Nodes WHERE NodeId=%d;
PUBLIC void Storage_select();
// INSERT INTO Include_Nodes (NodeId, Name, EntryNode)
// INSERT INTO Code_Nodes (NodeId, Code)
//   "INSERT INTO Comment_Nodes (NodeId, Comment) "
PUBLIC void Storage_insert();
// UPDATE Include_Nodes SET EntryNode = %d WHERE NodeId = %d;

// UPDATE Nodes SET NodeNext = %d WHERE NodeId = %d;
PUBLIC void Storage_update();

/* "CREATE TABLE Nodes ("
   "NodeId integer PRIMARY_KEY,"
   "NodeType integer NOT NULL,"
   "NodePtr integer NOT NULL,"
   "NodeNext integer,"
   "NodePrev integer"
   ");");*/
PUBLIC void Storage_create();
// "DROP TABLE "
PUBLIC void STorage_drop();
#endif /* _STORAGE_H_ */

```

4.16 Mutex.h

```
/* Mutex.h */
#ifndef _MUTEX_H_
#define _MUTEX_H_

#include "Types.h"
#ifndef WIN32
#include <pthread.h>
#else
#include <windows.h>
#endif

typedef struct Mutex
{
#ifndef WIN32
  pthread_mutex_t *mutex;
  pthread_cond_t cond;
#else
  HANDLE mutex;
#endif
} Mutex;

PRIVATE void Mutex_new(Mutex * this, int initState);
PRIVATE void Mutex_delete(Mutex * this);
PRIVATE void Mutex_take(Mutex * this);
PRIVATE void Mutex_release(Mutex * this);
PRIVATE void Mutex_waitAvailability(Mutex * this);

PRIVATE void Mutex_new(Mutex * this, int initState)
{
#ifndef WIN32
//int pthread_mutex_init(this->mutex, const pthread_mutexattr_t *restrict attr);
//int pthread_cond_init(pthread_cond_t *restrict cond, const pthread_condattr_t *restrict attr);
//this->cond = PTHREAD_COND_INITIALIZER;
#else
//this->mutex = CreateMutexW(NULL, TRUE, NULL);      // Set
#endif
}

PRIVATE void Mutex_delete(Mutex * this)
{
//int pthread_mutex_destroy(this->mutex);
//int pthread_cond_destroy(pthread_cond_t *cond);
//CloseHandle(this->mutex);
  //if (hScreenMutex) CloseHandle(hScreenMutex);
  //if (hRunMutex) CloseHandle(hRunMutex);
}

PRIVATE void Mutex_take(Mutex * this)
{
#ifndef WIN32
//int pthread_mutex_lock(pthread_mutex_t *mutex);
#else
#endif
}

PRIVATE void Mutex_release(Mutex * this)
{
#ifndef WIN32
//int pthread_mutex_unlock(pthread_mutex_t *mutex)
//pthread_cond_signal(&condition); //wake up thread 1
#else
//ReleaseMutex(this->mutex);
#endif
}

PRIVATE void Mutex_waitAvailability(Mutex * this)
{
  //dwWaitResult = WaitForSingleObject(this->mutex, INFINITE);  // no time-out interval
  //pthread_cond_wait(&cond, &lock);
  //int pthread_cond_timedwait(pthread_cond_t *restrict cond,
  //      pthread_mutex_t *restrict mutex,
  //      const struct timespec *restrict abstime
}
#endif /* _MUTEX_H_ */
```

4.17 Task.h

```
/* Task.h */
#ifndef _TASK_H_
#define _TASK_H_

```

```
#include "Types.h"

typedef struct Task Task;

PUBLIC Task* Task_create(void * (*body)(void* p), int nbParams, void ** params);
PUBLIC void Task_destroy(Task * this);
PUBLIC void Task_start(Task * this);
PUBLIC int Task_isReady(Task * this);
PUBLIC void Task_setReady(Task * this);
PUBLIC int Task_isRunning(Task * this);
PUBLIC void Task_setRunning(Task * this);
PUBLIC int Task_isCompleted(Task * this);
PUBLIC void Task_setCompleted(Task * this);
PUBLIC void Task_destroy(Task * this);
PUBLIC void Task_executeBody(Task * this);

#endif /* _TASK_H_ */
```

4.18 Task.h

```
/* Task.h */
#ifndef _TASK_H_
#define _TASK_H_

#include "Types.h"

typedef struct Task Task;

PUBLIC Task* Task_create(void * (*body)(void* p), int nbParams, void ** params);
PUBLIC void Task_destroy(Task * this);
PUBLIC void Task_start(Task * this);
PUBLIC int Task_isReady(Task * this);
PUBLIC void Task_setReady(Task * this);
PUBLIC int Task_isRunning(Task * this);
PUBLIC void Task_setRunning(Task * this);
PUBLIC int Task_isCompleted(Task * this);
PUBLIC void Task_setCompleted(Task * this);
PUBLIC void Task_destroy(Task * this);
PUBLIC void Task_executeBody(Task * this);

#endif /* _TASK_H_ */
```

4.19 TaskMgr.h

```
/* TaskMgr.h */
#ifndef _TASKMGR_H_
#define _TASKMGR_H_

#include "Types.h"

typedef struct Task Task;
typedef struct TaskMgr TaskMgr;

PUBLIC TaskMgr * TaskMgr_getRef();
PUBLIC void TaskMgr_delete(TaskMgr * this);
PUBLIC void TaskMgr_print(TaskMgr * this);
PUBLIC unsigned int TaskMgr_getSize(TaskMgr * this);
PUBLIC int TaskMgr_start(TaskMgr * this, Task * task);
PUBLIC void TaskMgr_stop(TaskMgr* this);
#endif /* _TASKMGR_H_ */
```

4.20 TaskMgr.h

```
/* TaskMgr.h */
#ifndef _TASKMGR_H_
#define _TASKMGR_H_

#include "Types.h"

typedef struct Task Task;
typedef struct TaskMgr TaskMgr;

PUBLIC TaskMgr * TaskMgr_getRef();
PUBLIC void TaskMgr_delete(TaskMgr * this);
PUBLIC void TaskMgr_print(TaskMgr * this);
PUBLIC unsigned int TaskMgr_getSize(TaskMgr * this);
PUBLIC int TaskMgr_start(TaskMgr * this, Task * task);
PUBLIC void TaskMgr_stop(TaskMgr* this);
#endif /* _TASKMGR_H_ */
```

4.21 /home/thomas/Projects/SParse-master/SParse/src/AppliLib/Time↩ Mgr/TimeMgr.c File Reference

This file contains the implementation for the class TimeMgr.

```
#include ¨TimeMgr.h¨
#include ¨Timer.h¨
#include ¨Class.h¨
#include ¨Object.h¨
#include ¨Map.h¨
```

Classes

- class TimeMgr

Functions

- PUBLIC void **TimeMgr_report** (TimeMgr ∗this)

Variables

- PRIVATE TimeMgr ∗ **timeMgr** = 0

4.21.1 Detailed Description

This file contains the implementation for the class TimeMgr.

The class TimeMgr provides an interface to the creation of timers.

4.22 TimeMgr.h

```
/* TimeMgr.h */

#ifndef _TIMEMGR_H_
#define _TIMEMGR_H_

#include "Types.h"
#include "String2.h"

typedef struct TimeMgr TimeMgr;

PUBLIC void TimeMgr_delete(TimeMgr * this);
PUBLIC TimeMgr * TimeMgr_copy(TimeMgr * this);
PUBLIC TimeMgr * TimeMgr_getRef();
PUBLIC unsigned int TimeMgr_getSize(TimeMgr * this);
PUBLIC void TimeMgr_latchTime(TimeMgr * this, String * s);
PUBLIC void TimeMgr_report(TimeMgr * this);

#endif /* _TIMEMGR_H_ */
```

4.23 TimeMgr.h

```
/* TimeMgr.h */

#ifndef _TIMEMGR_H_
#define _TIMEMGR_H_

#include "Types.h"
#include "String2.h"

typedef struct TimeMgr TimeMgr;

PUBLIC void TimeMgr_delete(TimeMgr * this);
PUBLIC TimeMgr * TimeMgr_copy(TimeMgr * this);
PUBLIC TimeMgr * TimeMgr_getRef();
PUBLIC unsigned int TimeMgr_getSize(TimeMgr * this);
PUBLIC void TimeMgr_latchTime(TimeMgr * this, String * s);
PUBLIC void TimeMgr_report(TimeMgr * this);

#endif /* _TIMEMGR_H_ */
```

4.24 Timer.h

```
/* Timer.h */

#include "Types.h"

typedef struct Timer Timer;

PUBLIC Timer * Timer_new();
PUBLIC void Timer_delete(Timer * this);
PUBLIC Timer * Timer_copy(Timer * this);
PUBLIC unsigned int Timer_getSize(Timer * this);
PUBLIC unsigned int Timer_isEqual(Timer * this, Timer * compared);
PUBLIC void Timer_print(Timer * this);
PUBLIC void Timer_latchTime(Timer * this);
```

4.25 Timer.h

```
/* Timer.h */

#include "Types.h"

typedef struct Timer Timer;

PUBLIC Timer * Timer_new();
PUBLIC void Timer_delete(Timer * this);
PUBLIC Timer * Timer_copy(Timer * this);
PUBLIC unsigned int Timer_getSize(Timer * this);
PUBLIC unsigned int Timer_isEqual(Timer * this, Timer * compared);
PUBLIC void Timer_print(Timer * this);
PUBLIC void Timer_latchTime(Timer * this);
```

4.26 Allocator.h

```
/* Allocator.h */
#ifndef _ALLOCATOR_H_
#define _ALLOCATOR_H_

typedef struct Allocator Allocator;

typedef void * (*NewFunction)();
typedef void (*DeleteFunction)(Allocator * allocator);
typedef void* (*AllocateFunction)(Allocator * allocator, unsigned int size);
typedef void (*DeAllocateFunction)(Allocator * allocator, void * ptr);
typedef unsigned int (*ReportFunction)(Allocator * allocator);

struct Allocator
{
  NewFunction new;
  DeleteFunction delete;
  AllocateFunction allocate;
  DeAllocateFunction deallocate;
  ReportFunction report;
  unsigned int nbAllocatedObjects;
};

Allocator * Allocator_new();
void * Allocator_allocate(Allocator * this, unsigned int size);
//void * Allocator_allocFromClass(Allocator * this, /* Class class*/);
void Allocator_deallocate(Allocator * this, void * ptr);
void Allocator_delete(Allocator * this);
#endif /* _ALLOCATOR_H_ */
```

4.27 Allocator.h

```
/* Allocator.h */
#ifndef _ALLOCATOR_H_
#define _ALLOCATOR_H_

typedef struct Allocator Allocator;

typedef void * (*NewFunction)();
typedef void (*DeleteFunction)(Allocator * allocator);
typedef void* (*AllocateFunction)(Allocator * allocator, unsigned int size);
typedef void (*DeAllocateFunction)(Allocator * allocator, void * ptr);
typedef unsigned int (*ReportFunction)(Allocator * allocator);

struct Allocator
{
  NewFunction new;
  DeleteFunction delete;
  AllocateFunction allocate;
  DeAllocateFunction deallocate;
  ReportFunction report;
  unsigned int nbAllocatedObjects;
};

Allocator * Allocator_new();
void * Allocator_allocate(Allocator * this, unsigned int size);
//void * Allocator_allocFromClass(Allocator * this, /* Class class*/);
void Allocator_deallocate(Allocator * this, void * ptr);
void Allocator_delete(Allocator * this);
#endif /* _ALLOCATOR_H_ */
```

4.28 Malloc.h

```
#ifndef _MALLOC_H_
#define _MALLOC_H_
#include "Allocator.h"
#include "Types.h"

typedef struct Malloc Malloc;

PUBLIC Malloc * Malloc_getRef();
PUBLIC void Malloc_delete(Allocator * this);
PUBLIC void * Malloc_allocate(Allocator * this, unsigned int size);
PUBLIC void Malloc_deallocate(Allocator * this, void * ptr);
PUBLIC unsigned int Malloc_report(Allocator * this);
#endif /* _MALLOC_H_ */
```

4.29 Malloc.h

```
#ifndef _MALLOC_H_
#define _MALLOC_H_
#include "Allocator.h"
#include "Types.h"

typedef struct Malloc Malloc;

PUBLIC Malloc * Malloc_getRef();
PUBLIC void Malloc_delete(Allocator * this);
PUBLIC void * Malloc_allocate(Allocator * this, unsigned int size);
PUBLIC void Malloc_deallocate(Allocator * this, void * ptr);
PUBLIC unsigned int Malloc_report(Allocator * this);
#endif /* _MALLOC_H_ */
```

4.30 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/↩ Array/Array.c File Reference

This file contains the implementation of the class Array.

```
#include "Array.h"
#include "Class.h"
#include "Object.h"
#include "Debug.h"
```

Classes

- class Array

Macros

- #define **NB_ELEMENT_MAX** (100)
- #define **ELEMENT_SIZE_BYTES** (100)

Functions

- PUBLIC unsigned int **Array_getSize** (Array ∗this)

4.30.1 Detailed Description

This file contains the implementation of the class Array.

The class Array implement the Array operations:

- init
- put
- get

4.31 Array.h

```
#ifndef _ARRAY_H_
#define _ARRAY_H_
/*******************************************************************************
* Array.h
*
*******************************************************************************/
#include "Types.h"
#include "Object.h"
#include "FileIo.h"

typedef struct Array Array;

typedef struct ArrayParam
{
  unsigned int defaultSize;
  unsigned int storageMode;
  unsigned int autoresize;
} ArrayParam;

PUBLIC Array * Array_new(ArrayParam * param);
PUBLIC Array * Array_newFromFile(FileIo * fileIo, ArrayParam * param);
PUBLIC void Array_delete(Array* this);
PUBLIC Array * Array_copy(Array* this);
PUBLIC int Array_compare(Array * this, Array * compared);
PUBLIC void Array_print(Array * this);
PUBLIC void Array_put(Array * this, unsigned int index);
PUBLIC Object * Array_get(Array * this, unsigned int index);
PUBLIC unsigned int Array_getSize(Array * this);

#endif /* _ARRAY_H_ */
```

4.32 Array.h

```
#ifndef _ARRAY_H_
#define _ARRAY_H_
/**************************************************************************************
* Array.h
*
**************************************************************************************/
#include "Types.h"
#include "Object.h"
#include "FileIo.h"

typedef struct Array Array;

typedef struct ArrayParam
{
  unsigned int defaultSize;
  unsigned int storageMode;
  unsigned int autoresize;
} ArrayParam;

PUBLIC Array * Array_new(ArrayParam * param);
PUBLIC Array * Array_newFromFile(FileIo * fileIo, ArrayParam * param);
PUBLIC void Array_delete(Array* this);
PUBLIC Array * Array_copy(Array* this);
PUBLIC int Array_compare(Array * this, Array * compared);
PUBLIC void Array_print(Array * this);
PUBLIC void Array_put(Array * this, unsigned int index);
PUBLIC Object * Array_get(Array * this, unsigned int index);
PUBLIC unsigned int Array_getSize(Array * this);

#endif /* _ARRAY_H_ */
```

4.33 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/↩ BTree/BTree.c File Reference

This file contains the implementation of the class BTree.

```
#include "BTree.h"
#include "Node.h"
#include "Memory.h"
#include "Debug.h"
```

Classes

- class BTree

Functions

- PUBLIC BTree ∗ **BTree_new** (unsigned int order)
- PUBLIC void **BTree_delete** (BTree ∗this)
- PUBLIC BTree ∗ **BTree_copy** (BTree ∗this)
- PUBLIC int **BTree_comp** (BTree ∗this, BTree ∗compared)
- PUBLIC void **BTree_add** (BTree ∗tree, Object ∗key, Object ∗object, int isOwner)
- PUBLIC Object ∗ **BTree_get** (BTree ∗tree, Object ∗key)
- PUBLIC Object ∗ **BTree_remove** (BTree ∗tree, Object ∗key)
- PUBLIC void **BTree_print** (BTree ∗tree)
- PUBLIC BTree ∗ **BTree_newFromFile** (char ∗fileName)
- PUBLIC unsigned int **BTree_getSize** (BTree ∗this)
- PUBLIC unsigned int **BTree_getNbNodes** (BTree ∗this)

4.33.1 Detailed Description

This file contains the implementation of the class BTree.

The class BTree implements the BTree operations:

- init
- add
- remove

4.34 BTree.h

```
#ifndef _BTREE_
#define _BTREE_
/********************************************************************************
* BTree.h
*
********************************************************************************/
#include "Types.h"
#include "Object.h"
#include "Allocator.h"

typedef struct BTree BTree;

PUBLIC BTree * BTree_new(unsigned int order);
PUBLIC BTree * BTree_newFromAllocator(Allocator * allocator);
PUBLIC BTree * BTree_newFromFile(char* fileName);
PUBLIC void BTree_delete(BTree * tree);
PUBLIC BTree * BTree_copy(BTree * this);
PUBLIC int BTree_comp(BTree * this, BTree * compared);
PUBLIC void BTree_add(BTree * tree, Object * key, Object * object, int isOwner);
PUBLIC Object * BTree_get(BTree * tree, Object * key);
PUBLIC Object * BTree_remove(BTree * tree, Object * key);
PUBLIC void BTree_print(BTree * tree);
PUBLIC unsigned int BTree_sizeof(BTree* tree);
PUBLIC unsigned int BTree_reportSizeInBytes(BTree * tree);
PUBLIC unsigned int BTree_getSize(BTree * this);
PUBLIC unsigned int BTree_getNbNodes(BTree* this);

#endif /* _BTREE_ */

```

4.35 BTree.h

```
#ifndef _BTREE_
#define _BTREE_
/********************************************************************************
* BTree.h
*
********************************************************************************/
#include "Types.h"
#include "Object.h"
#include "Allocator.h"

typedef struct BTree BTree;

PUBLIC BTree * BTree_new(unsigned int order);
PUBLIC BTree * BTree_newFromAllocator(Allocator * allocator);
PUBLIC BTree * BTree_newFromFile(char* fileName);
PUBLIC void BTree_delete(BTree * tree);
PUBLIC BTree * BTree_copy(BTree * this);
PUBLIC int BTree_comp(BTree * this, BTree * compared);
PUBLIC void BTree_add(BTree * tree, Object * key, Object * object, int isOwner);
PUBLIC Object * BTree_get(BTree * tree, Object * key);
PUBLIC Object * BTree_remove(BTree * tree, Object * key);
PUBLIC void BTree_print(BTree * tree);
PUBLIC unsigned int BTree_sizeof(BTree* tree);
PUBLIC unsigned int BTree_reportSizeInBytes(BTree * tree);
PUBLIC unsigned int BTree_getSize(BTree * this);
PUBLIC unsigned int BTree_getNbNodes(BTree* this);

#endif /* _BTREE_ */

```

4.36 Node.h

```
/*
* Node.h
*/
#ifndef _NODE_
#define _NODE_

#include "Types.h"
#include "Object.h"

typedef struct Node Node;

typedef struct Node
{
    unsigned int nbKeyUsed;
    unsigned int isLeaf;
    Object ** keys;
    Object ** leaves;
    Node ** children;
    Object* buffer[18];
} Node;

PUBLIC Node * Node_new(unsigned short int isLeaf, unsigned int order);
PUBLIC Node* Node_splitNode(Node * node, unsigned int order, Node* nodeToSplit, Object * key);
PUBLIC void Node_insert(Node * node, unsigned int order, Object * key, Object * object, int isOwner);
PUBLIC Object * Node_remove(Node * node, unsigned int order, Object * key, Object ** keyToUpdate);
PUBLIC Object * Node_search(Node * node, unsigned int order, Object * key, unsigned int
isFoundAlready);
PUBLIC void Node_free(Node * node, unsigned int order);
PUBLIC void Node_print(Node * node, unsigned int order, unsigned int depth);
PUBLIC unsigned int Node_getNbNodes(Node * node);
#endif /* _NODE_ */
```

4.37 Node.h

```
/*
* Node.h
*/
#ifndef _NODE_
#define _NODE_

#include "Types.h"
#include "Object.h"

typedef struct Node Node;

typedef struct Node
{
    unsigned int nbKeyUsed;
    unsigned int isLeaf;
    Object ** keys;
    Object ** leaves;
    Node ** children;
    Object* buffer[18];
} Node;

PUBLIC Node * Node_new(unsigned short int isLeaf, unsigned int order);
PUBLIC Node* Node_splitNode(Node * node, unsigned int order, Node* nodeToSplit, Object * key);
PUBLIC void Node_insert(Node * node, unsigned int order, Object * key, Object * object, int isOwner);
PUBLIC Object * Node_remove(Node * node, unsigned int order, Object * key, Object ** keyToUpdate);
PUBLIC Object * Node_search(Node * node, unsigned int order, Object * key, unsigned int
isFoundAlready);
PUBLIC void Node_free(Node * node, unsigned int order);
PUBLIC void Node_print(Node * node, unsigned int order, unsigned int depth);
PUBLIC unsigned int Node_getNbNodes(Node * node);
#endif /* _NODE_ */
```

4.38 TestObject.h

```
/* TestObject.h */
#ifndef _TESTOBJECT_H_
#define _TESTOBJECT_H_

#include "Types.h"
#include "Allocator.h"

```

```
typedef struct TestObject TestObject;

PUBLIC TestObject * TestObject_new();
PUBLIC TestObject * TestObject_newFromAllocator(Allocator* allocator);
PUBLIC void TestObject_delete(TestObject * this);
PUBLIC int TestObject_compare(TestObject* this, TestObject* compare);
PUBLIC TestObject* TestObject_copy();
PUBLIC void TestObject_print(TestObject* this);
PUBLIC unsigned int TestObject_getSize(TestObject* this);
#endif /* _TESTOBJECT_H_ */
```

4.39 TestObject.h

```
/* TestObject.h */
#ifndef _TESTOBJECT_H_
#define _TESTOBJECT_H_

#include "Types.h"
#include "Allocator.h"

typedef struct TestObject TestObject;

PUBLIC TestObject * TestObject_new();
PUBLIC TestObject * TestObject_newFromAllocator(Allocator* allocator);
PUBLIC void TestObject_delete(TestObject * this);
PUBLIC int TestObject_compare(TestObject* this, TestObject* compare);
PUBLIC TestObject* TestObject_copy();
PUBLIC void TestObject_print(TestObject* this);
PUBLIC unsigned int TestObject_getSize(TestObject* this);
#endif /* _TESTOBJECT_H_ */
```

4.40 TestObject.h

```
/* TestObject.h */
#ifndef _TESTOBJECT_H_
#define _TESTOBJECT_H_

#include "Types.h"
#include "Allocator.h"

typedef struct TestObject TestObject;

PUBLIC TestObject * TestObject_new();
PUBLIC TestObject * TestObject_newFromAllocator(Allocator* allocator);
PUBLIC void TestObject_delete(TestObject * this);
PUBLIC int TestObject_compare(TestObject* this, TestObject* compare);
PUBLIC TestObject* TestObject_copy();
PUBLIC void TestObject_print(TestObject* this);
PUBLIC unsigned int TestObject_getSize(TestObject* this);
#endif /* _TESTOBJECT_H_ */
```

4.41 TestObject.h

```
/* TestObject.h */
#ifndef _TESTOBJECT_H_
#define _TESTOBJECT_H_

#include "Types.h"
#include "Allocator.h"

typedef struct TestObject TestObject;

PUBLIC TestObject * TestObject_new();
PUBLIC TestObject * TestObject_newFromAllocator(Allocator* allocator);
PUBLIC void TestObject_delete(TestObject * this);
PUBLIC int TestObject_compare(TestObject* this, TestObject* compare);
PUBLIC TestObject* TestObject_copy();
PUBLIC void TestObject_print(TestObject* this);
PUBLIC unsigned int TestObject_getSize(TestObject* this);
#endif /* _TESTOBJECT_H_ */
```

4.42 Words1000.h

```
/* Words1000.h */

#ifndef _WORDS1000_H_
#define _WORDS1000_H_

char words1000[] = "hang holistic oceanic development decorous fence question songs chase relation adamant guitar writer ruthless "
"harass unnatural invite bite aloof numberless answer yam carriage apparatus graceful wonderful demonic gruesome "
"imaginary bewildered hypnotic deer therapeutic nasty relax righteous apathetic wrong mother clever impress mix harm "
"aspiring hook drip frog fairies workable mass monkey key sneeze window chunky ambiguous ice well-off matter thoughtful "
"number cure messy frame mere fax cheat shoes voiceless alert kind woozy impulse wander kick beds elated drop north babies "
"cumbersome sense theory note calculator yummy paper enormous drink slope agreement jump incredible oafish aboriginal deeply "
"animal pump cooperative kittens letters support pies connection excellent peaceful fertile lavish efficacious office "
"statement puncture many tin good overwrought believe rush remarkable keen abounding uncle aromatic tight dizzy taste lumber "
"cars obey stitch poke embarrass hard-to-find glossy grumpy fortunate fire elderly puny repeat jumpy erect ignorant tongue "
"fine oranges rejoice madly border squirrel equable baseball size zoo umbrella belong hole mountain cake volatile hungry "
"flesh attract tumble advice vague unequal narrow victorious optimal pushy trashy hop blush warn earthquake houses undesirable "
"harmonious same punch motionless allow children color homely large subsequent instruct flowery stretch property advertisement "
"page straight frightening glistening joyous high-pitched ticket reading dream can uncovered low unbecoming chivalrous belief "
"pretty nose crush refuse fence stew blot colour envious wandering oven art famous witty two hard kaput cat move delightful "
"memorize berserk inject wise kind monkey flow release industrious substantial rainy last agreeable wealth celery choke protect "
"existence pick far-flung ruddy fire trick knee humor mess up occur grade back nail help zonked spare crowd right sail left "
"stare crooked instrument night recondite subdued pray deadpan knotty locket creator pastoral plausible ambiguous rest nondescript "
"fowl unused macabre innate push torpid educated penitent chase governor plantation debonair irate faint crawl thundering judicious "
"righteous sleep bury mind button wax effect afterthought pushy radiate tender glass business stale filthy tranquil jellyfish "
"rule gaping finicky fall suggestion diligent sordid bless gate grey whole eight willing scratch houses hall north notice "
"distribution song nasty amazing fax card fog pie whip utter troubled cause gifted paddle difficult fish bee engine enchanting "
"ring familiar glove design save medical mark obeisant functional yam likeable kneel desire ossified toe subtract maniacal analyze "
"long-term run available enter alive vulgar grouchy thinkable pan fragile laughable quarrelsome tasty money finger tongue confuse "
"beam calculator return industry gratis whine neighborly coherent prevent past haircut flame gaze reading dream can uncovered "
"low unbecoming chivalrous belief pretty nose crush refuse fence stew blot colour envious wandering oven art famous witty two "
"hard kaput cat move delightful memorize berserk inject wise kind monkey flow release industrious substantial rainy last agreeable "
"wealth celery choke protect existence pick far-flung ruddy fire trick knee humor mess up occur grade back nail help zonked spare "
"crowd right sail left stare crooked instrument night recondite subdued pray deadpan knotty locket creator pastoral plausible "
"ambiguous rest nondescript fowl unused macabre innate push torpid educated penitent chase governor plantation debonair irate "
"faint crawl thundering judicious righteous sleep bury mind button wax effect afterthought pushy radiate tender glass business "
"stale filthy tranquil jellyfish rule gaping finicky fall suggestion diligent sordid bless gate grey whole eight willing scratch "
"houses hall north notice distribution song nasty amazing fax card fog pie whip utter troubled cause gifted paddle difficult fish "
"bee engine enchanting ring familiar glove design save medical mark obeisant functional yam likeable kneel desire ossified toe "
"subtract maniacal analyze long-term run available enter alive vulgar grouchy thinkable pan fragile laughable quarrelsome tasty "
"money finger tongue confuse beam calculator return industry gratis whine neighborly coherent prevent past haircut flame gaze "
"yak delay duck confuse wave subdued trot cumbersome burst rainy bang dysfunctional remain lyrical yummy red fluttering plucky "
"winter scandalous grotesque celery nimble powder spiffy hope fabulous terrible interrupt diligent daffy watch short flash ambiguous "
"distinct geese offend scarecrow jump quaint inquisitive hysterical wide bait wax wish children tap infamous pale flagrant "
"unbecoming rescue canvas cast harass snails person distance succinct watery curious familiar lonely oatmeal approve spell meddle "
"kettle vein plausible jazzy quicksand hard object luxuriant instinctive approval important spill bitter fixed level gusty concern "
```

```
"courageous argument name hydrant gruesome fireman mellow invite woozy risk quarter zoo bed rigid insurance puzzling amusement "
"spooky weather authority bite sincere scale quince grouchy left song aware wretched ear magenta story blue holiday sudden "
"direction friendly gray earn befitting cap fall suggestion regular tumble park slip jeans stomach warn appliance pink false "
"double airport cheer glib demonic hollow teeny-tiny year dance advertisement tub brown thankful chicken book attend crook "
"saw wry entertaining pets spurious chess wood uninterested macho lacking hands stingy fuzzy super error clover agreement erratic "
"explain drip four finger condemned creature spade root downtown materialistic concerned decide bat mate silly breathe psychotic "
"stiff phobic front disarm rob pop health assorted public known clean lunchroom cause scissors dynamic tearful scene range "
"elite outgoing giraffe payment seat acrid trees entertain rabid north sidewalk song cloth tasty grotesque icky stew grip cherry "
"mighty push hushed short ashamed plate teeny-tiny sparkle lopsided spoon shelf release puffy abaft record deeply zephyr found "
"lame rude colorful advise kill border big line slim unhealthy bake love unique crash hospitable warn vast picture oatmeal "
"harbor ill crazy orange expensive afford whimsical scream ship sudden relieved grubby last add blind zealous obsequious screw "
"jar science throat answer daughter annoyed riddle itchy questionable throne shiny confused vulgar squirrel jaded vivacious "
"cute snow order chief smash wing night defeated murky keen hesitant neat believe rampant lake voyage bubble tick current "
"quarter dirty blot wander care rambunctious spoil scary treatment trashy mine expect trick laughable undress rub selective "
"channel delay hammer overconfident quince craven befitting colour arch mask knit jumpy bouncy psychotic belligerent stare "
"understood freezing milky long accidental attach hot discreet ready house string maddening loutish spurious door birthday gabby "
"mute step spotless scrape explain used carpenter heavy argument bump paste feigned coherent driving stroke separate tenuous "
"letters analyze unarmed bawdy different absurd six hurt shame fast pour church vague real whole";


#endif /* _WORDS1000_H_ */
```

4.43 Words1000.h

```
/* Words1000.h */

#ifndef _WORDS1000_H_
#define _WORDS1000_H_

char words1000[] = "hang holistic oceanic development decorous fence question songs chase relation adamant guitar writer ruthless "
"harass unnatural invite bite aloof numberless answer yam carriage apparatus graceful wonderful demonic gruesome "
"imaginary bewildered hypnotic deer therapeutic nasty relax righteous apathetic wrong mother clever impress mix harm "
"aspiring hook drip frog fairies workable mass monkey key sneeze window chunky ambiguous ice well-off matter thoughtful "
"number cure messy frame mere fax cheat shoes voiceless alert kind woozy impulse wander kick beds elated drop north babies "
"cumbersome sense theory note calculator yummy paper enormous drink slope agreement jump incredible oafish aboriginal deeply "
"animal pump cooperative kittens letters support pies connection excellent peaceful fertile lavish efficacious office "
"statement puncture many tin good overwrought believe rush remarkable keen abounding uncle aromatic tight dizzy taste lumber "
"cars obey stitch poke embarrass hard-to-find glossy grumpy fortunate fire elderly puny repeat jumpy erect ignorant tongue "
"fine oranges rejoice madly border squirrel equable baseball size zoo umbrella belong hole mountain cake volatile hungry "
"flesh attract tumble advice vague unequal narrow victorious optimal pushy trashy hop blush warn earthquake houses undesirable "
"harmonious same punch motionless allow children color homely large subsequent instruct flowery stretch property advertisement "
"page straight frightening glistening joyous high-pitched ticket reading dream can uncovered low unbecoming chivalrous belief "
"pretty nose crush refuse fence stew blot colour envious wandering oven art famous witty two hard kaput cat move delightful "
"memorize berserk inject wise kind monkey flow release industrious substantial rainy last agreeable wealth celery choke protect "
"existence pick far-flung ruddy fire trick knee humor mess up occur grade back nail help zonked spare crowd right sail left "
"stare crooked instrument night recondite subdued pray deadpan knotty locket creator pastoral plausible ambiguous rest nondescript "
"fowl unused macabre innate push torpid educated penitent chase governor plantation debonair irate faint crawl thundering judicious "
"righteous sleep bury mind button wax effect afterthought pushy radiate tender glass business stale filthy tranquil jellyfish "
```

```
"rule gaping finicky fall suggestion diligent sordid bless gate grey whole eight willing scratch houses hall north notice "
"distribution song nasty amazing fax card fog pie whip utter troubled cause gifted paddle difficult fish bee engine enchanting "
"ring familiar glove design save medical mark obeisant functional yam likeable kneel desire ossified toe subtract maniacal analyze "
"long-term run available enter alive vulgar grouchy thinkable pan fragile laughable quarrelsome tasty money finger tongue confuse "
"beam calculator return industry gratis whine neighborly coherent prevent past haircut flame gaze reading dream can uncovered "
"low unbecoming chivalrous belief pretty nose crush refuse fence stew blot colour envious wandering oven art famous witty two "
"hard kaput cat move delightful memorize berserk inject wise kind monkey flow release industrious substantial rainy last agreeable "
"wealth celery choke protect existence pick far-flung ruddy fire trick knee humor mess up occur grade back nail help zonked spare "
"crowd right sail left stare crooked instrument night recondite subdued pray deadpan knotty locket creator pastoral plausible "
"ambiguous rest nondescript fowl unused macabre innate push torpid educated penitent chase governor plantation debonair irate "
"faint crawl thundering judicious righteous sleep bury mind button wax effect afterthought pushy radiate tender glass business "
"stale filthy tranquil jellyfish rule gaping finicky fall suggestion diligent sordid bless gate grey whole eight willing scratch "
"houses hall north notice distribution song nasty amazing fax card fog pie whip utter troubled cause gifted paddle difficult fish "
"bee engine enchanting ring familiar glove design save medical mark obeisant functional yam likeable kneel desire ossified toe "
"subtract maniacal analyze long-term run available enter alive vulgar grouchy thinkable pan fragile laughable quarrelsome tasty "
"money finger tongue confuse beam calculator return industry gratis whine neighborly coherent prevent past haircut flame gaze "
"yak delay duck confuse wave subdued trot cumbersome burst rainy bang dysfunctional remain lyrical yummy red fluttering plucky "
"winter scandalous grotesque celery nimble powder spiffy hope fabulous terrible interrupt diligent daffy watch short flash ambiguous "
"distinct geese offend scarecrow jump quaint inquisitive hysterical wide bait wax wish children tap infamous pale flagrant "
"unbecoming rescue canvas cast harass snails person distance succinct watery curious familiar lonely oatmeal approve spell meddle "
"kettle vein plausible jazzy quicksand hard object luxuriant instinctive approval important spill bitter fixed level gusty concern "
"courageous argument name hydrant gruesome fireman mellow invite woozy risk quarter zoo bed rigid insurance puzzling amusement "
"spooky weather authority bite sincere scale quince grouchy left song aware wretched ear magenta story blue holiday sudden "
"direction friendly gray earn befitting cap fall suggestion regular tumble park slip jeans stomach warn appliance pink false "
"double airport cheer glib demonic hollow teeny-tiny year dance advertisement tub brown thankful chicken book attend crook "
"saw wry entertaining pets spurious chess wood uninterested macho lacking hands stingy fuzzy super error clover agreement erratic "
"explain drip four finger condemned creature spade root downtown materialistic concerned decide bat mate silly breathe psychotic "
"stiff phobic front disarm rob pop health assorted public known clean lunchroom cause scissors dynamic tearful scene range "
"elite outgoing giraffe payment seat acrid trees entertain rabid north sidewalk song cloth tasty grotesque icky stew grip cherry "
"mighty push hushed short ashamed plate teeny-tiny sparkle lopsided spoon shelf release puffy abaft record deeply zephyr found "
"lame rude colorful advise kill border big line slim unhealthy bake love unique crash hospitable warn vast picture oatmeal "
"harbor ill crazy orange expensive afford whimsical scream ship sudden relieved grubby last add blind zealous obsequious screw "
"jar science throat answer daughter annoyed riddle itchy questionable throne shiny confused vulgar squirrel jaded vivacious "
"cute snow order chief smash wing night defeated murky keen hesitant neat believe rampant lake voyage bubble tick current "
"quarter dirty blot wander care rambunctious spoil scary treatment trashy mine expect trick laughable undress rub selective "
"channel delay hammer overconfident quince craven befitting colour arch mask knit jumpy bouncy psychotic belligerent stare "
"understood freezing milky long accidental attach hot discreet ready house string maddening loutish spurious door birthday gabby "
"mute step spotless scrape explain used carpenter heavy argument bump paste feigned coherent driving stroke separate tenuous "
"letters analyze unarmed bawdy different absurd six hurt shame fast pour church vague real whole";


#endif /* _WORDS1000_H_ */
```

4.44 Words1000.h

```
/* Words1000.h */
```

```

#ifndef _WORDS1000_H_
#define _WORDS1000_H_

const char words1000[] = "hang holistic oceanic development decorous fence question songs chase relation adamant guitar writer ruthless "
"harass unnatural invite bite aloof numberless answer yam carriage apparatus graceful wonderful demonic gruesome "
"imaginary bewildered hypnotic deer therapeutic nasty relax righteous apathetic wrong mother clever impress mix harm "
"aspiring hook drip frog fairies workable mass monkey key sneeze window chunky ambiguous ice well-off matter thoughtful "
"number cure messy frame mere fax cheat shoes voiceless alert kind woozy impulse wander kick beds elated drop north babies "
"cumbersome sense theory note calculator yummy paper enormous drink slope agreement jump incredible oafish aboriginal deeply "
"animal pump cooperative kittens letters support pies connection excellent peaceful fertile lavish efficacious office "
"statement puncture many tin good overwrought believe rush remarkable keen abounding uncle aromatic tight dizzy taste lumber "
"cars obey stitch poke embarrass hard-to-find glossy grumpy fortunate fire elderly puny repeat jumpy erect ignorant tongue "
"fine oranges rejoice madly border squirrel equable baseball size zoo umbrella belong hole mountain cake volatile hungry "
"flesh attract tumble advice vague unequal narrow victorious optimal pushy trashy hop blush warn earthquake houses undesirable "
"harmonious same punch motionless allow children color homely large subsequent instruct flowery stretch property advertisement "
"page straight frightening glistening joyous high-pitched ticket reading dream can uncovered low unbecoming chivalrous belief "
"pretty nose crush refuse fence stew blot colour envious wandering oven art famous witty two hard kaput cat move delightful "
"memorize berserk inject wise kind monkey flow release industrious substantial rainy last agreeable wealth celery choke protect "
"existence pick far-flung ruddy fire trick knee humor mess up occur grade back nail help zonked spare crowd right sail left "
"stare crooked instrument night recondite subdued pray deadpan knotty locket creator pastoral plausible ambiguous rest nondescript "
"fowl unused macabre innate push torpid educated penitent chase governor plantation debonair irate faint crawl thundering judicious "
"righteous sleep bury mind button wax effect afterthought pushy radiate tender glass business stale filthy tranquil jellyfish "
"rule gaping finicky fall suggestion diligent sordid bless gate grey whole eight willing scratch houses hall north notice "
"distribution song nasty amazing fax card fog pie whip utter troubled cause gifted paddle difficult fish bee engine enchanting "
"ring familiar glove design save medical mark obeisant functional yam likeable kneel desire ossified toe subtract maniacal analyze "
"long-term run available enter alive vulgar grouchy thinkable pan fragile laughable quarrelsome tasty money finger tongue confuse "
"beam calculator return industry gratis whine neighborly coherent prevent past haircut flame gaze reading dream can uncovered "
"low unbecoming chivalrous belief pretty nose crush refuse fence stew blot colour envious wandering oven art famous witty two "
"hard kaput cat move delightful memorize berserk inject wise kind monkey flow release industrious substantial rainy last agreeable "
"wealth celery choke protect existence pick far-flung ruddy fire trick knee humor mess up occur grade back nail help zonked spare "
"crowd right sail left stare crooked instrument night recondite subdued pray deadpan knotty locket creator pastoral plausible "
"ambiguous rest nondescript fowl unused macabre innate push torpid educated penitent chase governor plantation debonair irate "
"faint crawl thundering judicious righteous sleep bury mind button wax effect afterthought pushy radiate tender glass business "
"stale filthy tranquil jellyfish rule gaping finicky fall suggestion diligent sordid bless gate grey whole eight willing scratch "
"houses hall north notice distribution song nasty amazing fax card fog pie whip utter troubled cause gifted paddle difficult fish "
"bee engine enchanting ring familiar glove design save medical mark obeisant functional yam likeable kneel desire ossified toe "
"subtract maniacal analyze long-term run available enter alive vulgar grouchy thinkable pan fragile laughable quarrelsome tasty "
"money finger tongue confuse beam calculator return industry gratis whine neighborly coherent prevent past haircut flame gaze "
"yak delay duck confuse wave subdued trot cumbersome burst rainy bang dysfunctional remain lyrical yummy red fluttering plucky "
"winter scandalous grotesque celery nimble powder spiffy hope fabulous terrible interrupt diligent daffy watch short flash ambiguous "
"distinct geese offend scarecrow jump quaint inquisitive hysterical wide bait wax wish children tap infamous pale flagrant "
"unbecoming rescue canvas cast harass snails person distance succinct watery curious familiar lonely oatmeal approve spell meddle "
"kettle vein plausible jazzy quicksand hard object luxuriant instinctive approval important spill bitter fixed level gusty concern "
"courageous argument name hydrant gruesome fireman mellow invite woozy risk quarter zoo bed rigid insurance puzzling amusement "
"spooky weather authority bite sincere scale quince grouchy left song aware wretched ear magenta story
```

```
      blue holiday sudden "
      "direction friendly gray earn befitting cap fall suggestion regular tumble park slip jeans stomach
      warn appliance pink false "
      "double airport cheer glib demonic hollow teeny-tiny year dance advertisement tub brown thankful
      chicken book attend crook "
      "saw wry entertaining pets spurious chess wood uninterested macho lacking hands stingy fuzzy super
      error clover agreement erratic "
      "explain drip four finger condemned creature spade root downtown materialistic concerned decide bat
      mate silly breathe psychotic "
      "stiff phobic front disarm rob pop health assorted public known clean lunchroom cause scissors dynamic
      tearful scene range "
      "elite outgoing giraffe payment seat acrid trees entertain rabid north sidewalk song cloth tasty
      grotesque icky stew grip cherry "
      "mighty push hushed short ashamed plate teeny-tiny sparkle lopsided spoon shelf release puffy abaft
      record deeply zephyr found "
      "lame rude colorful advise kill border big line slim unhealthy bake love unique crash hospitable warn
      vast picture oatmeal "
      "harbor ill crazy orange expensive afford whimsical scream ship sudden relieved grubby last add blind
      zealous obsequious screw "
      "jar science throat answer daughter annoyed riddle itchy questionable throne shiny confused vulgar
      squirrel jaded vivacious "
      "cute snow order chief smash wing night defeated murky keen hesitant neat believe rampant lake voyage
      bubble tick current "
      "quarter dirty blot wander care rambunctious spoil scary treatment trashy mine expect trick laughable
      undress rub selective "
      "channel delay hammer overconfident quince craven befitting colour arch mask knit jumpy bouncy
      psychotic belligerent stare "
      "understood freezing milky long accidental attach hot discreet ready house string maddening loutish
      spurious door birthday gabby "
      "mute step spotless scrape explain used carpenter heavy argument bump paste feigned coherent driving
      stroke separate tenuous "
      "letters analyze unarmed bawdy different absurd six hurt shame fast pour church vague real whole";


      #endif /* _WORDS1000_H_ */
```

4.45 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/↩ Debug/Debug.c File Reference

This file contains debugging functions.

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

Functions

- void **dbg_printf** (const char *fmt,...)

4.45.1 Detailed Description

This file contains debugging functions.

The debugging function are TBD

4.46 Debug.h

```
/* Debug.h */

#include <stdarg.h>
#include <stdio.h>

#define TRACE(x) do { if (DEBUG) dbg_printf x; } while (0)

#define PRINT(x) do { dbg_printf x; } while (0)

void dbg_printf(const char *fmt, ...);
```

4.47 Debug.h

```
/* Debug.h */

#include <stdarg.h>
#include <stdio.h>

#define TRACE(x) do { if (DEBUG) dbg_printf x; } while (0)

#define PRINT(x) do { dbg_printf x; } while (0)

void dbg_printf(const char *fmt, ...);
```

4.48 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/↩ Error/Error.c File Reference

Reports errors.

```
#include "Error.h"
#include "Debug.h"
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
```

Macros

- #define **DEBUG** (1)

Functions

- PUBLIC void Error_new (ErrorSeverity severity, char ∗msg,...)

  Reports errors.

4.48.1 Detailed Description

Reports errors.

This file contains error reporting functions.

4.48.2 Function Documentation

4.48.2.1 Error_new()

```
PUBLIC void Error_new (
              ErrorSeverity severity,
              char * msg,
               ... )
```

Reports errors.

Parameters

severity	Enum
msg	Variable list of parameters

This function reports errors using different formatting according to severity.

4.49 Error.h

```
/* Error.h */

#include "Types.h"

typedef enum
{
  ERROR_DBG,
  ERROR_INFO,
  ERROR_NORMAL,
  ERROR_FATAL
} ErrorSeverity;

PUBLIC void Error_new(ErrorSeverity severity, char * msg, ...);
```

4.50 Error.h

```
/* Error.h */

#include "Types.h"

typedef enum
{
  ERROR_DBG,
  ERROR_INFO,
  ERROR_NORMAL,
  ERROR_FATAL
} ErrorSeverity;

PUBLIC void Error_new(ErrorSeverity severity, char * msg, ...);
```

4.51 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/↩ FileIo/FileIo.c File Reference

A FileIo class. This class provides a status and operation for various File I/O operations.

```
#include ¨FileIo.h¨
#include ¨String2.h¨
#include ¨List.h¨
#include ¨Object.h¨
#include ¨Memory.h¨
#include ¨Error.h¨
#include ¨Debug.h¨
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>
```

Classes

- struct FileIo

Macros

- #define **DEBUG** (0)

Functions

- PUBLIC void **FileIo_write** (FileIo ∗this, char ∗buffer, int length)
- PUBLIC void **FileIo_read** (FileIo ∗this, char ∗buffer, int length)
- PUBLIC void **FileIo_remove** (FileIo ∗this, String ∗fullFileName)
- PUBLIC void **FileIo_createDir** (FileIo ∗this, String ∗fullDirName)
- PUBLIC List ∗ **FileIo_listDirs** (FileIo ∗this, String ∗directory)
- PUBLIC List ∗ **FileIo_listFiles** (FileIo ∗this, String ∗directory)
- PUBLIC int **FileIo_fSeekEnd** (FileIo ∗this, int pos)
- PUBLIC String ∗ **FileIo_getCwd** (FileIo ∗this)
- PUBLIC int **FileIo_fSeekSet** (FileIo ∗this, int pos)
- PUBLIC int **FileIo_ftell** (FileIo ∗this)
- PUBLIC FileIoStatus **FileIo_isOpen** (FileIo ∗this)

4.51.1 Detailed Description

A FileIo class. This class provides a status and operation for various File I/O operations.

4.52 FileIo.h

```
#ifndef _FILEIO_H_
#define _FILEIO_H_
#include "String2.h"
#include "Types.h"

typedef enum FileIoStatus
{
    UNKNOWN=0,
    FILE_OPEN,
    DIR_OPEN
} FileIoStatus;

typedef struct FileIo FileIo;

PUBLIC FileIo * FileIo_new();
PUBLIC void FileIo_delete();
PUBLIC FileIo* FileIo_copy(FileIo* this);
PUBLIC int FileIo_comp(FileIo* this, FileIo* compare);
PUBLIC void FileIo_print(FileIo* this);
PUBLIC unsigned int FileIo_getSize(FileIo* this);
PUBLIC void FileIo_openFile(FileIo* this, String* fullFileName);
PUBLIC void FileIo_createFile(FileIo* this, String* fullFileName);
PUBLIC void FileIo_openDir(FileIo* this, String* fullFileName);
PUBLIC void FileIo_createDir(FileIo* this, String* fullDirName);
PUBLIC void FileIo_write(FileIo* this, char* buffer, int length);
PUBLIC void FileIo_read(FileIo* this, char* buffer, int length);
PUBLIC void FileIo_remove(FileIo* this, String* fullFileName);
PUBLIC String * FileIo_getCwd(FileIo* this);
PUBLIC List * FileIo_listDirs(FileIo * this, String * directory);
PUBLIC List* FileIo_listFiles(FileIo* this, String * directory);
PUBLIC int FileIo_fSeekEnd(FileIo * this, int pos);
PUBLIC int FileIo_fSeekSet(FileIo * this, int pos);
PUBLIC int FileIo_ftell(FileIo* this);
PUBLIC FileIoStatus FileIo_isOpen(FileIo * this);
//Opendir
//Readdir

#endif /* _FILEIO_H_ */
```

4.53 FileIo.h

```
#ifndef _FILEIO_H_
#define _FILEIO_H_
#include "String2.h"
#include "Types.h"

typedef enum FileIoStatus
{
    UNKNOWN=0,
    FILE_OPEN,
    DIR_OPEN
} FileIoStatus;

typedef struct FileIo FileIo;

PUBLIC FileIo * FileIo_new();
PUBLIC void FileIo_delete();
PUBLIC FileIo* FileIo_copy(FileIo* this);
PUBLIC int FileIo_comp(FileIo* this, FileIo* compare);
PUBLIC void FileIo_print(FileIo* this);
PUBLIC unsigned int FileIo_getSize(FileIo* this);
PUBLIC void FileIo_openFile(FileIo* this, String* fullFileName);
PUBLIC void FileIo_createFile(FileIo* this, String* fullFileName);
PUBLIC void FileIo_openDir(FileIo* this, String* fullFileName);
PUBLIC void FileIo_createDir(FileIo* this, String* fullDirName);
PUBLIC void FileIo_write(FileIo* this, char* buffer, int length);
PUBLIC void FileIo_read(FileIo* this, char* buffer, int length);
PUBLIC void FileIo_remove(FileIo* this, String* fullFileName);
PUBLIC String * FileIo_getCwd(FileIo* this);
PUBLIC List * FileIo_listDirs(FileIo * this, String * directory);
PUBLIC List* FileIo_listFiles(FileIo* this, String * directory);
PUBLIC int FileIo_fSeekEnd(FileIo * this, int pos);
PUBLIC int FileIo_fSeekSet(FileIo * this, int pos);
PUBLIC int FileIo_ftell(FileIo* this);
PUBLIC FileIoStatus FileIo_isOpen(FileIo * this);
//Opendir
//Readdir

#endif /* _FILEIO_H_ */
```

4.54 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/↵ List/List.c File Reference

This file contains the implementation of the class List.

```
#include "List.h"
#include "Class.h"
#include "Object.h"
#include "Memory.h"
#include "Debug.h"
#include "ListNode.h"
```

Classes

- class List

Functions

- PUBLIC void ∗ **List_getNext** (List ∗this)
- PUBLIC void **List_resetIterator** (List ∗this)

4.54.1 Detailed Description

This file contains the implementation of the class List.

The class List implement the List operations:

- init
- add

4.55 List.h

```
/* List.h */

#ifndef _LIST_H_
#define _LIST_H_

#include "Types.h"
#include "Allocator.h"

typedef struct List List;

PUBLIC List * List_new();
PUBLIC List * List_newFromAllocator(Allocator * allocator);
PUBLIC void List_delete(List* this);
PUBLIC List * List_copy(List* this);
PUBLIC int List_compare(List * this, List * compared);
PUBLIC void List_print(List * this);
PUBLIC void List_insertHead(List* this, void* item, int isOwner);
PUBLIC void List_insertTail(List* this, void* item, int isOwner);
PUBLIC void List_merge(List* this, List* l1);
PUBLIC void List_forEach(List* this, void (*method)(void* o));
PUBLIC void * List_getNext(List* this);
PUBLIC void * List_removeHead(List * this);
PUBLIC void* List_removeTail(List* this);
PUBLIC void * List_getHead(List * this);
PUBLIC unsigned int List_getSize(List * this);
PUBLIC unsigned int List_getNbNodes(List * this);
PUBLIC void List_resetIterator(List * this);

#endif /* _LIST_H_ */
```

4.56 List.h

```
/* List.h */

#ifndef _LIST_H_
#define _LIST_H_

#include "Types.h"
#include "Allocator.h"

typedef struct List List;

PUBLIC List * List_new();
PUBLIC List * List_newFromAllocator(Allocator * allocator);
PUBLIC void List_delete(List* this);
PUBLIC List * List_copy(List* this);
PUBLIC int List_compare(List * this, List * compared);
PUBLIC void List_print(List * this);
PUBLIC void List_insertHead(List* this, void* item, int isOwner);
PUBLIC void List_insertTail(List* this, void* item, int isOwner);
PUBLIC void List_merge(List* this, List* l1);
PUBLIC void List_forEach(List* this, void (*method)(void* o));
PUBLIC void * List_getNext(List* this);
PUBLIC void * List_removeHead(List * this);
PUBLIC void* List_removeTail(List* this);
PUBLIC void * List_getHead(List * this);
PUBLIC unsigned int List_getSize(List * this);
PUBLIC unsigned int List_getNbNodes(List * this);
PUBLIC void List_resetIterator(List * this);

#endif /* _LIST_H_ */
```

4.57 ListNode.h

```
/* ListNode.h */

#ifndef _LISTNODE_H_
#define _LISTNODE_H_

#include "Types.h"
#include "Allocator.h"
#include "Class.h"
#include "Object.h"
#include "Memory.h"
#include "ObjectStore.h"
#include "Error.h"

typedef struct ListNode ListNode;

PRIVATE ListNode * ListNode_new();
PRIVATE ListNode * ListNode_newFromAllocator(Allocator * allocator, Object * object, int isOwner);
PRIVATE void ListNode_delete(ListNode* this);
PRIVATE ListNode * ListNode_copy(ListNode* this);
PRIVATE int ListNode_compare(ListNode * this, ListNode * compared);
PRIVATE void ListNode_print(ListNode * this);
PRIVATE unsigned int ListNode_getSize(ListNode * this);

/**********************************************/
struct ListNode
{
  Object object;
  void* item;
  int isOwner;
  ListNode* next;
  ListNode* prev;
};

/**********************************************/
PRIVATE Class listNodeClass =
{
  .f_new = (Constructor)0,
  .f_delete = (Destructor)&ListNode_delete,
  .f_copy = (Copy_Operator)&ListNode_copy,
  .f_comp = (Comp_Operator)&ListNode_compare,
  .f_print = (Printer)&ListNode_print,
  .f_size = (Sizer)&ListNode_getSize
};

/**********************************************/
PRIVATE ListNode* ListNode_new(Object* object, int isOwner)
{
  ListNode* this = 0;

  this = (ListNode*)Object_new(sizeof(ListNode), &listNodeClass);

  if (this != 0)
  {
    if (isOwner)
      this->item = object;
    else
      this->item = Object_getRef(object);
    this->isOwner = isOwner;
    this->next = 0;
    this->prev = 0;
  }

  return this;
}

/**********************************************/
PRIVATE ListNode* ListNode_newFromAllocator(Allocator* allocator, Object* object, int isOwner)
{
  ListNode* this = 0;

  this = (ListNode*)Object_newFromAllocator(&listNodeClass, allocator);

  if (this != 0)
  {
    if (isOwner)
      this->item = object;
    else
      this->item = Object_getRef(object);
    this->isOwner = isOwner;
    this->next = 0;
    this->prev = 0;
  }

  return this;
}
```

```

/**************************************************/
PRIVATE void ListNode_delete(ListNode* this)
{
  if (this != 0)
  {
    if ((this->item) && (((Object*)this->item)->delete != 0))
    {
      if (((Object*)this->item)->marker != 0x0B5EC7)
      {
          Error_new(ERROR_NORMAL, "List Node: item is not an object and is being deleted\n");
      }
      else
      {
      if (this->isOwner)
        ((Object*)this->item)->delete(this->item);
      else
        Object_deRef((Object*)this->item);
      }
    }
    Object_deallocate(&this->object);
  }
}

/**************************************************/
PRIVATE ListNode* ListNode_copy(ListNode* this)
{
  ListNode* copy = 0;

  if (this != 0)
  {
    copy = ListNode_new(((Object*)this->item), 0);
  }

  return copy;
}

/**************************************************/
PRIVATE int ListNode_compare(ListNode* this, ListNode* compared)
{
  unsigned int result = 0;

  return result;
}

/**************************************************/
PRIVATE void ListNode_print(ListNode* this)
{
}

PRIVATE unsigned int ListNode_getSize(ListNode* this)
{
  return sizeof(ListNode);
}

#endif /* _LISTNODE_H_ */
```

4.58 MyAllocator.h

```
/* MyAllocator.h */
#ifndef _MYALLOCATOR_H_
#define _MYALLOCATOR_H_

#include "Allocator.h"
#include "Types.h"

typedef struct MyAllocator MyAllocator;

PUBLIC MyAllocator * MyAllocator_new(unsigned int size);
PUBLIC void MyAllocator_delete(MyAllocator * this);
void MyAllocator_reset(Allocator * this);
PUBLIC void * MyAllocator_allocate(Allocator * allocator, unsigned int size);
PUBLIC void MyAllocator_deallocate(Allocator * allocator, void * ptr);
PUBLIC unsigned int MyAllocator_report(Allocator * this);

#endif /* _MYALLOCATOR_H_ */
```

4.59 MyAllocator.h

```
/* MyAllocator.h */
```

```
#ifndef _MYALLOCATOR_H_
#define _MYALLOCATOR_H_

#include "Allocator.h"
#include "Types.h"

typedef struct MyAllocator MyAllocator;

PUBLIC MyAllocator * MyAllocator_new(unsigned int size);
PUBLIC void MyAllocator_delete(MyAllocator * this);
void MyAllocator_reset(Allocator * this);
PUBLIC void * MyAllocator_allocate(Allocator * allocator, unsigned int size);
PUBLIC void MyAllocator_deallocate(Allocator * allocator, void * ptr);
PUBLIC unsigned int MyAllocator_report(Allocator * this);

#endif /* _MYALLOCATOR_H_ */
```

4.60 MyAllocator.h

```
/* MyAllocator.h */
#ifndef _MYALLOCATOR_H_
#define _MYALLOCATOR_H_

#include "Allocator.h"
#include "Types.h"

typedef struct MyAllocator MyAllocator;

PUBLIC MyAllocator * MyAllocator_new();
PUBLIC void MyAllocator_delete(MyAllocator * this);
PUBLIC void * MyAllocator_allocate(Allocator * allocator, unsigned int size);
PUBLIC void MyAllocator_deallocate(Allocator * allocator, void * ptr);
PUBLIC unsigned int MyAllocator_report(Allocator * this);

#endif /* _MYALLOCATOR_H_ */
```

4.61 MyAllocator.h

```
/* MyAllocator.h */
#ifndef _MYALLOCATOR_H_
#define _MYALLOCATOR_H_

#include "Allocator.h"
#include "Types.h"

typedef struct MyAllocator MyAllocator;

PUBLIC MyAllocator * MyAllocator_new(unsigned int size);
PUBLIC void MyAllocator_delete(MyAllocator * this);
void MyAllocator_reset(Allocator * this);
PUBLIC void * MyAllocator_allocate(Allocator * allocator, unsigned int size);
PUBLIC void MyAllocator_deallocate(Allocator * allocator, void * ptr);
PUBLIC unsigned int MyAllocator_report(Allocator * this);

#endif /* _MYALLOCATOR_H_ */
```

4.62 MyAllocator.h

4.63 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/↩ Map/Map.c File Reference

A Map class. This class provides a container indexed by a string.

```
#include ¨Map.h¨
#include ¨MapEntry.h¨
#include ¨List.h¨
```

```
#include "Class.h"
#include "Object.h"
#include "String2.h"
#include "Memory.h"
#include "Debug.h"
#include "Error.h"
```

Classes

- class Map

Macros

- #define **DEBUG** (0)
- #define **HTABLE_SIZE** (50)

Functions

- PRIVATE MapEntry ∗ **Map_findEntry** (Map ∗this, String ∗s)
- PUBLIC int **Map_comp** (Map ∗this, Map ∗compared)

4.63.1 Detailed Description

A Map class. This class provides a container indexed by a string.

4.64 Map.h

```
/* Map.h */

#ifndef _MAP_H_
#define _MAP_H_

#include "Types.h"
#include "String2.h"
#include "List.h"
#include "Allocator.h"

typedef struct Map Map;

PUBLIC Map * Map_new();
PUBLIC Map* Map_newFromAllocator(Allocator * allocator);
PUBLIC void Map_delete(Map * this);
PUBLIC Map * Map_copy(Map * this);
PUBLIC int Map_comp(Map* this, Map* compared);
PUBLIC unsigned int Map_insert(Map * this,String* s, void * p, int isOwner);
PUBLIC unsigned int Map_find(Map * this, String* s, void ** p);
PUBLIC void Map_print(Map * this);
PUBLIC unsigned int Map_getSize(Map * this);
PUBLIC List * Map_getAll(Map * this);

#endif /* _MAP_H_ */
```

4.65 Map.h

```
/* Map.h */

#ifndef _MAP_H_
#define _MAP_H_

#include "Types.h"
#include "String2.h"
#include "List.h"
#include "Allocator.h"

typedef struct Map Map;

PUBLIC Map * Map_new();
PUBLIC Map* Map_newFromAllocator(Allocator * allocator);
PUBLIC void Map_delete(Map * this);
PUBLIC Map * Map_copy(Map * this);
PUBLIC int Map_comp(Map* this, Map* compared);
PUBLIC unsigned int Map_insert(Map * this,String* s, void * p, int isOwner);
PUBLIC unsigned int Map_find(Map * this, String* s, void ** p);
PUBLIC void Map_print(Map * this);
PUBLIC unsigned int Map_getSize(Map * this);
PUBLIC List * Map_getAll(Map * this);

#endif /* _MAP_H_ */
```

4.66 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/←↩ Map/MapEntry.c File Reference

A support class for the Map class.

```
#include "MapEntry.h"
#include "Class.h"
#include "Object.h"
#include "String2.h"
```

Classes

- struct MapEntry

Functions

- PUBLIC MapEntry ∗ **MapEntry_new** (String ∗s, void ∗item, int isOwner)
- PUBLIC MapEntry ∗ **MapEntry_newFromAllocator** (Allocator ∗allocator, String ∗s, void ∗item, int isOwner)
- PUBLIC void **MapEntry_delete** (MapEntry ∗this)
- PUBLIC MapEntry ∗ **MapEntry_copy** (MapEntry ∗this)
- PUBLIC unsigned int **MapEntry_getSize** (MapEntry ∗this)
- PUBLIC String ∗ **MapEntry_getString** (MapEntry ∗this)
- PUBLIC void **MapEntry_setString** (MapEntry ∗this, String ∗s)
- PUBLIC void ∗ **MapEntry_getItem** (MapEntry ∗this)
- PUBLIC void **MapEntry_setItem** (MapEntry ∗this, void ∗item)

4.66.1 Detailed Description

A support class for the Map class.

4.67 MapEntry.h

```
/* MapEntry.h */

#ifndef _MAPENTRY_H_
#define _MAPENTRY_H_

#include "Types.h"
#include "String2.h"
#include "Allocator.h"

typedef struct MapEntry MapEntry;

PUBLIC MapEntry * MapEntry_new(String * s, void * p, int isOwner);
PUBLIC MapEntry * MapEntry_newFromAllocator(Allocator * allocator, String *s, void * p, int isOwner);
PUBLIC void MapEntry_delete(MapEntry * this);
PUBLIC MapEntry * MapEntry_copy(MapEntry * this);
PUBLIC unsigned int MapEntry_getSize(MapEntry * this);
PUBLIC String * MapEntry_getString(MapEntry * this);
PUBLIC void * MapEntry_getItem(MapEntry * this);
PUBLIC void MapEntry_setString(MapEntry * this, String * s);
PUBLIC void MapEntry_setItem(MapEntry * this, void * item);

#endif /* _MAPENTRY_H_ */
```

4.68 MapEntry.h

```
/* MapEntry.h */

#ifndef _MAPENTRY_H_
#define _MAPENTRY_H_

#include "Types.h"
#include "String2.h"
#include "Allocator.h"

typedef struct MapEntry MapEntry;

PUBLIC MapEntry * MapEntry_new(String * s, void * p, int isOwner);
PUBLIC MapEntry * MapEntry_newFromAllocator(Allocator * allocator, String *s, void * p, int isOwner);
PUBLIC void MapEntry_delete(MapEntry * this);
PUBLIC MapEntry * MapEntry_copy(MapEntry * this);
PUBLIC unsigned int MapEntry_getSize(MapEntry * this);
PUBLIC String * MapEntry_getString(MapEntry * this);
PUBLIC void * MapEntry_getItem(MapEntry * this);
PUBLIC void MapEntry_setString(MapEntry * this, String * s);
PUBLIC void MapEntry_setItem(MapEntry * this, void * item);

#endif /* _MAPENTRY_H_ */
```

4.69 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/↩ Memory/Memory.c File Reference

This file provides the implementation of the memory functions.

```
#include "Memory.h"
#include "Debug.h"
#include "Error.h"
#include <stdlib.h>
#include <string.h>
```

Macros

- #define **DEBUG** (0)

Variables

- PRIVATE unsigned int **Memory_allocRequestId** = 0
- PRIVATE unsigned int **Memory_freeRequestId** = 0
- PRIVATE unsigned int **Memory_nbBytesAllocated** = 0

4.69.1 Detailed Description

This file provides the implementation of the memory functions.

TBD

4.70 Memory.h

```
/* Memory.h */

#ifndef _MEMORY_H_
#define _MEMORY_H_

#include "Types.h"

#ifdef _WIN32
#define MEMORY_ISVALID(p) (*(long int*)p!=0xCDCDCDCD)
#elif _WIN64
#define MEMORY_ISVALID(p) (p!=0xCDCDCDCD)
#else
#define MEMORY_ISVALID(p) (p!=0)
#endif

PUBLIC void* Memory_alloc(unsigned int nbBytes);
PUBLIC void * Memory_realloc(void * pointer, unsigned int prevSizeBytes, unsigned int newSizeBytes);
PUBLIC void Memory_free(void* pointer, unsigned int nbBytes);
PUBLIC void Memory_set(void * pointer, unsigned char val, unsigned int nbBytes);
PUBLIC void Memory_copy(void * pointer, void * src, unsigned int nbBytes);
PUBLIC int Memory_ncmp(void * pointer, void * compared, unsigned int nbBytes);
PUBLIC int Memory_cmp(void * pointer, void * compared);
PUBLIC unsigned int Memory_len(const void * pointer);
PUBLIC void Memory_report();
PUBLIC int Memory_getAllocRequestNb();
PUBLIC int Memory_getFreeRequestNb();

#endif /* _MEMORY_H_ */
```

4.71 Memory.h

```
/* Memory.h */

#ifndef _MEMORY_H_
#define _MEMORY_H_

#include "Types.h"

#ifdef _WIN32
#define MEMORY_ISVALID(p) (*(long int*)p!=0xCDCDCDCD)
#elif _WIN64
#define MEMORY_ISVALID(p) (p!=0xCDCDCDCD)
#else
#define MEMORY_ISVALID(p) (p!=0)
#endif

PUBLIC void* Memory_alloc(unsigned int nbBytes);
PUBLIC void * Memory_realloc(void * pointer, unsigned int prevSizeBytes, unsigned int newSizeBytes);
PUBLIC void Memory_free(void* pointer, unsigned int nbBytes);
PUBLIC void Memory_set(void * pointer, unsigned char val, unsigned int nbBytes);
PUBLIC void Memory_copy(void * pointer, void * src, unsigned int nbBytes);
PUBLIC int Memory_ncmp(void * pointer, void * compared, unsigned int nbBytes);
PUBLIC int Memory_cmp(void * pointer, void * compared);
PUBLIC unsigned int Memory_len(const void * pointer);
PUBLIC void Memory_report();
PUBLIC int Memory_getAllocRequestNb();
PUBLIC int Memory_getFreeRequestNb();

#endif /* _MEMORY_H_ */
```

4.72 Class.h

```
/* Class. h */

#ifndef _CLASS_H_
#define _CLASS_H_

typedef struct Class Class;

struct Object;
typedef struct Object* (*Constructor)();
typedef void (*Destructor)(struct Object*);
typedef struct Object* (*Copy_Operator)(struct Object*);
typedef int (*Comp_Operator)(struct Object*, struct Object*);
typedef char* (*Printer)(struct Object*);
typedef unsigned int (*Sizer)();

struct Class
{
  Constructor f_new;
  Destructor f_delete;
  Copy_Operator f_copy;
  Comp_Operator f_comp;
  Printer f_print;
  Sizer f_size;
};

#endif /* _CLASS_H_ */
```

4.73 Class.h

```
/* Class. h */

#ifndef _CLASS_H_
#define _CLASS_H_

typedef struct Class Class;

struct Object;
typedef struct Object* (*Constructor)();
typedef void (*Destructor)(struct Object*);
typedef struct Object* (*Copy_Operator)(struct Object*);
typedef int (*Comp_Operator)(struct Object*, struct Object*);
typedef char* (*Printer)(struct Object*);
typedef unsigned int (*Sizer)();

struct Class
{
  Constructor f_new;
  Destructor f_delete;
  Copy_Operator f_copy;
  Comp_Operator f_comp;
  Printer f_print;
  Sizer f_size;
};

#endif /* _CLASS_H_ */
```

4.74 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/↩ Object/Object.c File Reference

This file contains the implementation for the class Object.

```
#include "Class.h"
#include "Object.h"
#include "ObjectMgr.h"
#include "ObjectStore.h"
#include "Allocator.h"
#include "Memory.h"
#include "Error.h"
```

Macros

- #define **OBJECT_MARKER** (0x0B5EC7)

Variables

- PRIVATE ObjectStore ∗ **Object_objectStore** = 0

4.74.1 Detailed Description

This file contains the implementation for the class Object.

The class Object is TBD

4.75 Object.h

```
/* Object.h */

#ifndef _OBJECT_H_
#define _OBJECT_H_

#include "Types.h"
#include "Class.h"
#include "Allocator.h"

typedef struct Object Object;

struct Object
{
  int marker;
  unsigned int id;
  unsigned int uniqId;
  Class * class;
  void (*delete)(Object * this);
  Object * (*copy)(Object * this);
  unsigned int refCount;
  unsigned int size;
  Allocator * allocator;
};

PUBLIC Object * Object_new(unsigned int size, Class * class);
PUBLIC Object* Object_newFromAllocator(Class* class, Allocator* allocator);
PUBLIC void Object_delete(Object * this);
PUBLIC void Object_deallocate(Object* this);
PUBLIC Object * Object_copy(Object * this);
PUBLIC int Object_comp(Object * this, Object * compared);
PUBLIC char * Object_print(Object * this);
PUBLIC Object* Object_getRef(Object* this);
PUBLIC void Object_deRef(Object * this);
PUBLIC int Object_isValid(Object* this);

#endif /* _OBJECT_H_ */
```

4.76 Object.h

```
/* Object.h */

#ifndef _OBJECT_H_
#define _OBJECT_H_

#include "Types.h"
#include "Class.h"
#include "Allocator.h"

typedef struct Object Object;

struct Object
{
  int marker;
  unsigned int id;
  unsigned int uniqId;
  Class * class;
  void (*delete)(Object * this);
  Object * (*copy)(Object * this);
  unsigned int refCount;
  unsigned int size;
  Allocator * allocator;
};

PUBLIC Object * Object_new(unsigned int size, Class * class);
PUBLIC Object* Object_newFromAllocator(Class* class, Allocator* allocator);
PUBLIC void Object_delete(Object * this);
PUBLIC void Object_deallocate(Object* this);
PUBLIC Object * Object_copy(Object * this);
PUBLIC int Object_comp(Object * this, Object * compared);
PUBLIC char * Object_print(Object * this);
PUBLIC Object* Object_getRef(Object* this);
PUBLIC void Object_deRef(Object * this);
PUBLIC int Object_isValid(Object* this);

#endif /* _OBJECT_H_ */
```

4.77 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/←ObjectMgr/ObjectMgr.c File Reference

An object management class.

```
#include "ObjectMgr.h"
#include "Object.h"
#include "Memory.h"
#include "Debug.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

Classes

- struct ObjectInfo
- class ObjectMgr

Macros

- #define **MAX_NB_OBJECTS** (40000)
- #define **END_OF_QUEUE** (0xFFFFFFFF)

Typedefs

- typedef struct ObjectInfo **ObjectInfo**

Variables

- PRIVATE ObjectMgr ∗ **objectMgr** = 0

4.77.1 Detailed Description

An object management class.

This class provides an object allocation and de-allocation service. Only one instance of this class can be created.

4.78 ObjectMgr.h

```
/* ObjectMgr.h */

#ifndef _OBJECTMGR_H_
#define _OBJECTMGR_H_

#include "Object.h"
#include "Types.h"

typedef struct ObjectMgr ObjectMgr;

PUBLIC void ObjectMgr_delete(ObjectMgr * this);
PUBLIC ObjectMgr * ObjectMgr_copy(ObjectMgr * this);
PUBLIC ObjectMgr * ObjectMgr_getRef();
PUBLIC Object * ObjectMgr_allocate(ObjectMgr * this, unsigned int size);
PUBLIC void ObjectMgr_deallocate(ObjectMgr * this, Object * object);
PUBLIC void ObjectMgr_reportUnallocated(ObjectMgr* this);
PUBLIC unsigned int ObjectMgr_report(ObjectMgr * this);

#endif /* _OBJECTMGR_H_ */
```

4.79 ObjectMgr.h

```
/* ObjectMgr.h */

#ifndef _OBJECTMGR_H_
#define _OBJECTMGR_H_

#include "Object.h"
#include "Types.h"

typedef struct ObjectMgr ObjectMgr;

PUBLIC void ObjectMgr_delete(ObjectMgr * this);
PUBLIC ObjectMgr * ObjectMgr_copy(ObjectMgr * this);
PUBLIC ObjectMgr * ObjectMgr_getRef();
PUBLIC Object * ObjectMgr_allocate(ObjectMgr * this, unsigned int size);
PUBLIC void ObjectMgr_deallocate(ObjectMgr * this, Object * object);
PUBLIC void ObjectMgr_reportUnallocated(ObjectMgr* this);
PUBLIC unsigned int ObjectMgr_report(ObjectMgr * this);

#endif /* _OBJECTMGR_H_ */
```

4.80 ObjectStore.h

```
/* ObjectStore.h */

#ifndef _OBJECTSTORE_H_
#define _OBJECTSTORE_H_

#include "Types.h"
#include "Object.h"
#include "Allocator.h"

typedef struct AllocInfo AllocInfo;
typedef struct ObjectStore ObjectStore;

PUBLIC void ObjectStore_delete(ObjectStore * this);
PUBLIC ObjectStore * ObjectStore_copy(ObjectStore* this);
PUBLIC ObjectStore * ObjectStore_getRef();
PUBLIC AllocInfo * ObjectStore_createAllocator(ObjectStore * this, Allocator * allocator);
PUBLIC void ObjectStore_deleteAllocator(ObjectStore * this, AllocInfo * allocInfo);
PUBLIC Object * ObjectStore_createObject(ObjectStore * this, Class * class, Allocator * allocator);
PUBLIC void ObjectStore_deleteObject(ObjectStore * this, Object * object);
PUBLIC void ObjectStore_reportUnallocated(ObjectStore * this);
PUBLIC void ObjectStore_report(ObjectStore * this);
PUBLIC unsigned int ObjectStore_getNbAllocatedObjects(ObjectStore * this);
PUBLIC int ObjectStore_compare(ObjectStore * this, ObjectStore * compared);
PUBLIC void ObjectStore_print(ObjectStore * this);

#endif /* _OBJECTSTORE_H_ */
```

4.81 ObjectStore.h

```
/* ObjectStore.h */

#ifndef _OBJECTSTORE_H_
#define _OBJECTSTORE_H_

#include "Types.h"
#include "Object.h"
#include "Allocator.h"

typedef struct AllocInfo AllocInfo;
typedef struct ObjectStore ObjectStore;

PUBLIC void ObjectStore_delete(ObjectStore * this);
PUBLIC ObjectStore * ObjectStore_copy(ObjectStore* this);
PUBLIC ObjectStore * ObjectStore_getRef();
PUBLIC AllocInfo * ObjectStore_createAllocator(ObjectStore * this, Allocator * allocator);
PUBLIC void ObjectStore_deleteAllocator(ObjectStore * this, AllocInfo * allocInfo);
PUBLIC Object * ObjectStore_createObject(ObjectStore * this, Class * class, Allocator * allocator);
PUBLIC void ObjectStore_deleteObject(ObjectStore * this, Object * object);
PUBLIC void ObjectStore_reportUnallocated(ObjectStore * this);
PUBLIC void ObjectStore_report(ObjectStore * this);
PUBLIC unsigned int ObjectStore_getNbAllocatedObjects(ObjectStore * this);
PUBLIC int ObjectStore_compare(ObjectStore * this, ObjectStore * compared);
PUBLIC void ObjectStore_print(ObjectStore * this);

#endif /* _OBJECTSTORE_H_ */
```

4.82 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/↩ Pool/Pool.c File Reference

This file contains the implementation of the class Pool.

```
#include "Pool.h"
#include "Memory.h"
#include "Error.h"
#include <stdio.h>
```

Macros

- #define **CACHE_NB** (6)
- #define **END_OF_QUEUE** (0xFFFFFFFF)
- #define **END_OF_ALLOC** (0xFFFFFFFE)
- #define **START_OF_AVAIL** (0xFFFFFFFD)

Functions

- PRIVATE AllocStatus Pool_allocInFile (Pool ∗pool, unsigned int ∗ptrIdx)

  Pool_allocInFile.
- PRIVATE void Pool_deallocInMemory (Pool ∗pool, unsigned int idx)

  Pool_deallocInMemory.
- PRIVATE void Pool_deallocInFile (Pool ∗pool, unsigned int idx)

Pool_deallocInFile.

- PRIVATE void Pool_reportInFile (Pool ∗pool)

  Pool_reportInFile.
- PRIVATE void Pool_reportInMemory (Pool ∗pool)

  Pool_reportInMemory.
- PRIVATE void Pool_readInFile (Pool ∗pool, unsigned int idx, void ∗p)

  Pool_readInFile.
- PRIVATE void Pool_readInMemory (Pool ∗pool, unsigned int idx, void ∗p)

  Pool_readInMemory.
- PRIVATE void Pool_writeInFile (Pool ∗pool, unsigned int idx, void ∗p)

  Pool_writeInFile.
- PRIVATE void Pool_writeInMemory (Pool ∗pool, unsigned int idx, void ∗p)

  Pool_writeInMemory.
- PUBLIC Pool ∗ Pool_new (unsigned int nbMemChunks, unsigned int memChunkSize)

  Create a new instance of the class Pool in RAM.
- PUBLIC Pool ∗ Pool_newFromFile (char ∗fileName, unsigned int nbMemChunks, unsigned int memChunk←↩ Size)

  Create a new instance of the class Pool in a file.
- PUBLIC void Pool_delete (Pool ∗pool)

  Pool_delete.
- PUBLIC void ∗ Pool_alloc (Pool ∗pool, unsigned int ∗ptrIdx)

  Pool_alloc.
- PUBLIC void Pool_dealloc (Pool ∗pool, unsigned int idx)

  Pool_dealloc.
- PUBLIC void Pool_write (Pool ∗pool, unsigned int idx, void ∗ptrContent)

  Pool_writeCache.
- PUBLIC void ∗ Pool_read (Pool ∗pool, unsigned int idx)

  Pool_read.
- PUBLIC void Pool_report (Pool ∗pool)

  Pool_report.
- PUBLIC unsigned int Pool_reportSizeInBytes (Pool ∗pool)

  Pool_reportSizeInBytes input: none.
- PUBLIC unsigned int Pool_reportNbNodes (Pool ∗pool)

  Pool_reportNbNodes.
- PUBLIC void **Pool_discardCache** (Pool ∗pool, unsigned int idx)
- PUBLIC void **Pool_discardAllCache** (Pool ∗pool)
- PUBLIC unsigned int **Pool_reportCacheUsed** (Pool ∗pool)

4.82.1 Detailed Description

This file contains the implementation of the class Pool.

The class List implement the Pool operations

- Alloc
- De-alloc

4.82.2 Function Documentation

4.82.2.1 Pool_alloc()

```
PUBLIC void * Pool_alloc (
            Pool * pool,
            unsigned int * ptrIdx )
```

Pool_alloc.

Parameters

in	*none*	

Returns

Reference to cache position, NULL is cache full

4.82.2.2 Pool_allocInFile()

```
PRIVATE AllocStatus Pool_allocInFile (
            Pool * pool,
            unsigned int * ptrIdx )
```

Pool_allocInFile.

Parameters

in	*none*	

Returns

none

4.82.2.3 Pool_dealloc()

```
PUBLIC void Pool_dealloc (
            Pool * pool,
            unsigned int idx )
```

Pool_dealloc.

Parameters

in	*none*	

Returns

none

4.82.2.4 Pool_deallocInFile()

```
PRIVATE void Pool_deallocInFile (
            Pool * pool,
            unsigned int idx )
```

Pool_deallocInFile.

Parameters

in	*none*	

Returns

none

4.82.2.5 Pool_deallocInMemory()

```
PRIVATE void Pool_deallocInMemory (
            Pool * pool,
            unsigned int idx )
```

Pool_deallocInMemory.

Parameters

in	*none*	

Returns

none

4.82.2.6 Pool_delete()

```
PUBLIC void Pool_delete (
            Pool * pool )
```

Pool_delete.

Parameters

in	*none*	

Returns

none

4.82.2.7 Pool_new()

```
PUBLIC Pool * Pool_new (
             unsigned int nbMemChunks,
             unsigned int memChunkSize )
```

Create a new instance of the class Pool in RAM.

Parameters

in	*number*	of memory chunks to allocate.
in	*size*	of memory chunk.

Returns

New instance.

4.82.2.8 Pool_newFromFile()

```
PUBLIC Pool * Pool_newFromFile (
             char * fileName,
             unsigned int nbMemChunks,
             unsigned int memChunkSize )
```

Create a new instance of the class Pool in a file.

Parameters

in	*File*	name
in	*Number*	of memory chunks to allocate
in	*Size*	of memory chunk return A pool of memory

4.82.2.9 Pool_read()

```
PUBLIC void * Pool_read (
             Pool * pool,
             unsigned int idx )
```

Pool_read.

Parameters

in	*none*	

Returns

none

4.82.2.10 Pool_readInFile()

```
PRIVATE void Pool_readInFile (
             Pool * pool,
             unsigned int idx,
             void * p )
```

Pool_readInFile.

Parameters

in	*none*	

Returns

none

4.82.2.11 Pool_readInMemory()

```
PRIVATE void Pool_readInMemory (
             Pool * pool,
             unsigned int idx,
             void * p )
```

Pool_readInMemory.

Parameters

in	*none*	

Returns

none

4.82.2.12 Pool_report()

```
PUBLIC void Pool_report (
             Pool * pool )
```

Pool_report.

Parameters

in	*none*	

Returns

none

4.82.2.13 Pool_reportInFile()

```
PRIVATE void Pool_reportInFile (
            Pool * pool )
```

Pool_reportInFile.

Parameters

in	*none*	

Returns

none

4.82.2.14 Pool_reportInMemory()

```
PRIVATE void Pool_reportInMemory (
            Pool * pool )
```

Pool_reportInMemory.

Parameters

in	*none*	

Returns

none

4.82.2.15 Pool_reportNbNodes()

```
PUBLIC unsigned int Pool_reportNbNodes (
            Pool * pool )
```

Pool_reportNbNodes.

Parameters

in	*none*	

Returns

none

4.82.2.16 Pool_reportSizeInBytes()

```
PUBLIC unsigned int Pool_reportSizeInBytes (
            Pool * pool )
```

Pool_reportSizeInBytes input: none.

Returns

none

4.82.2.17 Pool_write()

```
PUBLIC void Pool_write (
            Pool * pool,
            unsigned int idx,
            void * ptrContent )
```

Pool_writeCache.

Parameters

in	*none*	

Returns

none none

4.82.2.18 Pool_writeInFile()

```
PRIVATE void Pool_writeInFile (
            Pool * pool,
            unsigned int idx,
            void * p )
```

Pool_writeInFile.

Parameters

in	*none*	

Returns

none

4.82.2.19 Pool_writeInMemory()

```
PRIVATE void Pool_writeInMemory (
            Pool * pool,
            unsigned int idx,
            void * p )
```

Pool_writeInMemory.

Parameters

in	*none*	

Returns

none

4.83 Pool.h

```
#ifndef _POOL_
#define _POOL_
/*************************************************************************************
* Pool.h
*
*************************************************************************************/
#include "Types.h"
#include "Pool.h"

typedef enum AllocStatus
{
    ALLOC_OK = 0,
    ALLOC_FAIL = 1
} AllocStatus;

typedef struct PoolCache
{
    unsigned int idx;
    unsigned int isUsed;
    void* cache;
} PoolCache;

typedef struct Pool Pool;

PUBLIC Pool* Pool_new(unsigned int nbMemChunks, unsigned int memChunkSize);
PUBLIC Pool* Pool_newFixed(unsigned int nbMemChunks, unsigned int memChunkSize);
PUBLIC Pool* Pool_newFromFile(char* fileName, unsigned int nbMemChunks, unsigned int memChunkSize);
PUBLIC void Pool_delete(Pool* pool);
PUBLIC void * Pool_alloc(Pool* pool, unsigned int * ptrIdx);
PUBLIC void Pool_dealloc(Pool* pool, unsigned int p);
PUBLIC void Pool_write(Pool* pool, unsigned int idx, void* ptrContent);
PUBLIC void* Pool_read(Pool* pool, unsigned int idx);
PUBLIC unsigned int Pool_addToChunkCache(Pool* pool, void* p, unsigned int length);
PUBLIC void Pool_report(Pool* pool);
PUBLIC unsigned int Pool_reportSizeInBytes(Pool* pool);
PUBLIC unsigned int Pool_reportNbNodes(Pool* pool);
PUBLIC void Pool_discardCache(Pool* pool, unsigned int idx);
PUBLIC void Pool_discardAllCache(Pool* pool);
PUBLIC unsigned int Pool_reportCacheUsed(Pool * pool);
#endif /* _POOL_ */
```

4.84 Pool.h

```
#ifndef _POOL_
#define _POOL_
/*************************************************************************************
* Pool.h
*
*************************************************************************************/
#include "Types.h"
```

```
#include "Pool.h"

typedef enum AllocStatus
{
    ALLOC_OK = 0,
    ALLOC_FAIL = 1
} AllocStatus;

typedef struct PoolCache
{
    unsigned int idx;
    unsigned int isUsed;
    void* cache;
} PoolCache;

typedef struct Pool Pool;

PUBLIC Pool* Pool_new(unsigned int nbMemChunks, unsigned int memChunkSize);
PUBLIC Pool* Pool_newFixed(unsigned int nbMemChunks, unsigned int memChunkSize);
PUBLIC Pool* Pool_newFromFile(char* fileName, unsigned int nbMemChunks, unsigned int memChunkSize);
PUBLIC void Pool_delete(Pool* pool);
PUBLIC void * Pool_alloc(Pool* pool, unsigned int * ptrIdx);
PUBLIC void Pool_dealloc(Pool* pool, unsigned int p);
PUBLIC void Pool_write(Pool* pool, unsigned int idx, void* ptrContent);
PUBLIC void* Pool_read(Pool* pool, unsigned int idx);
PUBLIC unsigned int Pool_addToChunkCache(Pool* pool, void* p, unsigned int length);
PUBLIC void Pool_report(Pool* pool);
PUBLIC unsigned int Pool_reportSizeInBytes(Pool* pool);
PUBLIC unsigned int Pool_reportNbNodes(Pool* pool);
PUBLIC void Pool_discardCache(Pool* pool, unsigned int idx);
PUBLIC void Pool_discardAllCache(Pool* pool);
PUBLIC unsigned int Pool_reportCacheUsed(Pool * pool);
#endif /* _POOL_ */
```

4.85 /home/thomas/Projects/SParse-master/SParse/src/CommonLib/↩ SkipList/SkipList.c File Reference

This file contains the implementation of the class SkipList. The class List implement the SkipList operations.

```
#include ¨SkipList.h¨
#include ¨Pool.h¨
#include ¨Class.h¨
#include ¨Object.h¨
#include ¨Debug.h¨
#include <stdlib.h>
#include ¨SkipNode.h¨
```

Classes

- class SkipList

Typedefs

- typedef struct SkipList **SkipList**

Functions

- PUBLIC Object ∗ **SkipList_get** (SkipList ∗this, Object ∗key)

4.85.1 Detailed Description

This file contains the implementation of the class SkipList. The class List implement the SkipList operations.

- Add
- Remove
- Get

4.86 SkipList.h

```
/* SkipList.h */

#ifndef _SKIPLIST_
#define _SKIPLIST_

#include "Types.h"
#include "Object.h"
#include "Allocator.h"

typedef struct SkipList SkipList;

PUBLIC SkipList * SkipList_new();
PUBLIC SkipList * SkipList_newFromAllocator(Allocator * allocator);
PUBLIC void SkipList_delete(SkipList* skipList);
PUBLIC SkipList * SkipList_copy(SkipList * this);
PUBLIC void SkipList_add(SkipList* this, Object * key, Object * item);
PUBLIC Object * SkipList_remove(SkipList* this, Object * key);
PUBLIC Object * SkipList_get(SkipList* this, Object * key);
PUBLIC int SkipList_compare(SkipList * this, SkipList * compared);
PUBLIC void SkipList_print(SkipList* this);
PUBLIC unsigned int SkipList_getSize(SkipList* this);

#endif  /* _SKIPLIST_ */
```

4.87 SkipList.h

```
/* SkipList.h */

#ifndef _SKIPLIST_
#define _SKIPLIST_

#include "Types.h"
#include "Object.h"
#include "Allocator.h"

typedef struct SkipList SkipList;

PUBLIC SkipList * SkipList_new();
PUBLIC SkipList * SkipList_newFromAllocator(Allocator * allocator);
PUBLIC void SkipList_delete(SkipList* skipList);
PUBLIC SkipList * SkipList_copy(SkipList * this);
PUBLIC void SkipList_add(SkipList* this, Object * key, Object * item);
PUBLIC Object * SkipList_remove(SkipList* this, Object * key);
PUBLIC Object * SkipList_get(SkipList* this, Object * key);
PUBLIC int SkipList_compare(SkipList * this, SkipList * compared);
PUBLIC void SkipList_print(SkipList* this);
PUBLIC unsigned int SkipList_getSize(SkipList* this);

#endif  /* _SKIPLIST_ */
```

4.88 SkipNode.h

```
/* SkipNode.h */

#ifndef _SKIPNODE_H_
#define _SKIPNODE_H_

#include "Types.h"
```

```
#include "Object.h"
#include "Allocator.h"
#include <limits.h>

#define SKIPLIST_MAX_LEVEL (6)
#define END_NODE (0)

typedef struct SkipNode SkipNode;

PUBLIC SkipNode * SkipNode_new();
PUBLIC SkipNode * SkipNode_newFromAllocator();
PUBLIC void SkipNode_delete(SkipNode * this);
PUBLIC SkipNode * SkipNode_copy(SkipNode * this);
PUBLIC int SkipNode_compare(SkipNode * this, SkipNode * compared);
PUBLIC void SkipNode_print(SkipNode * this);
PUBLIC unsigned int SkipNode_getSize(SkipNode * this);

/**********************************************/
PRIVATE Class skipNodeClass =
{
  .f_new = 0,
  .f_delete = (Destructor)&SkipNode_delete,
  .f_copy = (Copy_Operator)&SkipNode_copy,
  .f_comp = (Comp_Operator)&SkipNode_compare,
  .f_print = (Printer)&SkipNode_print,
  .f_size = (Sizer)&SkipNode_getSize
};

typedef struct SkipNode
{
  Object object;
  Object * key;
  Object * item;
  unsigned int level;
  void * forward[SKIPLIST_MAX_LEVEL];
} SkipNode;


PUBLIC SkipNode * SkipNode_new()
{
  SkipNode * this = 0;

  this = (SkipNode*)Object_new(sizeof(SkipNode),&skipNodeClass);
  if (this==0) return 0;

  this->item = 0;
  this->level = 1;
  this->key = END_NODE;

  return this;
}

PUBLIC SkipNode * SkipNode_newFromAllocator(Allocator * allocator)
{
  SkipNode * this = 0;

  this = (SkipNode*)Object_newFromAllocator(&skipNodeClass, allocator);
  if (this == 0) return 0;
  this->item = 0;
  this->level = 1;
  this->key = END_NODE;
  return this;
}

PUBLIC void SkipNode_delete(SkipNode * this)
{
  if (this==0) return;

  //Object_delete(this->item);
  Object_deallocate(&this->object);
}

PUBLIC SkipNode * SkipNode_copy(SkipNode * this)
{
  SkipNode * copy = 0;

  return copy;
}

PUBLIC int SkipNode_compare(SkipNode * this, SkipNode * compared)
{
    return 1;
}

PUBLIC void SkipNode_print(SkipNode * this)
{
  if (this==0) return;



}

PUBLIC unsigned int SkipNode_getSize(SkipNode * this)
{
  return sizeof(SkipNode);
}
#endif /* _SKIPNODE_H_ */
```

4.89 String2.h

```
/* String2.h */

#ifndef _STRING2_H_
#define _STRING2_H_

#include "Types.h"
#include "List.h"

typedef struct String String;

PUBLIC String * String_new(const char * constString);
PUBLIC String * String_newByRef(const char * constString);
PUBLIC void String_delete(String * this);
PUBLIC String * String_copy(String * this);
PUBLIC String * String_getRef(String * this);
PUBLIC unsigned int String_getLength(String * this);
PUBLIC char * String_getBuffer(String * this);
PUBLIC void String_setBuffer(String* this, char* buffer, int isOwned);
PUBLIC unsigned int String_isContained(String * this, String * s2);
PUBLIC unsigned int String_prepend(String * this, const char * prefix);
PUBLIC unsigned int String_append(String* this, const char* postfix);
PUBLIC int String_compare(String * this, String * compared);
PUBLIC String * String_subString(String * this, unsigned int idx, unsigned int length);
PUBLIC unsigned int String_matchWildcard(String * this, const char * wildcard);
PUBLIC int String_toInt(String* this);
PUBLIC List* String_splitToken(String* this, const char* separator);
PUBLIC void String_stealBuffer(String* this, String* s);
PUBLIC unsigned int String_getSize(String* this);
PUBLIC void String_print(String* this);
#endif /* _STRING2_H_ */
```

4.90 String2.h

```
/* String2.h */

#ifndef _STRING2_H_
#define _STRING2_H_

#include "Types.h"
#include "List.h"

typedef struct String String;

PUBLIC String * String_new(const char * constString);
PUBLIC String * String_newByRef(const char * constString);
PUBLIC void String_delete(String * this);
PUBLIC String * String_copy(String * this);
PUBLIC String * String_getRef(String * this);
PUBLIC unsigned int String_getLength(String * this);
PUBLIC char * String_getBuffer(String * this);
PUBLIC void String_setBuffer(String* this, char* buffer, int isOwned);
PUBLIC unsigned int String_isContained(String * this, String * s2);
PUBLIC unsigned int String_prepend(String * this, const char * prefix);
PUBLIC unsigned int String_append(String* this, const char* postfix);
PUBLIC int String_compare(String * this, String * compared);
PUBLIC String * String_subString(String * this, unsigned int idx, unsigned int length);
PUBLIC unsigned int String_matchWildcard(String * this, const char * wildcard);
PUBLIC int String_toInt(String* this);
PUBLIC List* String_splitToken(String* this, const char* separator);
PUBLIC void String_stealBuffer(String* this, String* s);
PUBLIC unsigned int String_getSize(String* this);
PUBLIC void String_print(String* this);
#endif /* _STRING2_H_ */
```

4.91 Times.h

```
/* Time.h */

long double get_wall_time();
long double get_cpu_time();

```

4.92 Times.h

```
/* Time.h */

long double get_wall_time();
long double get_cpu_time();

```

4.93 Types.h

```
/* Types.h */

#ifndef _TYPES_H_
#define _TYPES_H_

#define PUBLIC

#define DECLARE_CLASS(x)

#ifndef UNIT_TEST
  #define PRIVATE static
#else
  #define PRIVATE
#endif

union mem_align
{
  void * a;
  long int b;
  long long c;
};

#define MEM_ALIGN (sizeof(union mem_align))

#include "UserTypes.h"
#endif /* _TYPES_H_ */
```

4.94 Types.h

```
/* Types.h */

#ifndef _TYPES_H_
#define _TYPES_H_

#define PUBLIC

#define DECLARE_CLASS(x)

#ifndef UNIT_TEST
  #define PRIVATE static
#else
  #define PRIVATE
#endif

union mem_align
{
  void * a;
  long int b;
  long long c;
};

#define MEM_ALIGN (sizeof(union mem_align))

#include "UserTypes.h"
#endif /* _TYPES_H_ */
```

4.95 UserTypes.h

```
/* User Types */
#ifndef _USERTYPES_
#define _USERTYPES_
#include "Class.h"
extern Class listClass;
extern Class stringClass;

enum ClassId
{
  ListClass,
  StringClass,
  NB_CLASSES
};

/*Class * userTypes[] =
{
  &listClass,
  &stringClass,
};*/
#endif /* UserTypes.h */
```

4.96 UserTypes.h

```
/* User Types */
#ifndef _USERTYPES_
#define _USERTYPES_
#include "Class.h"
extern Class listClass;
extern Class stringClass;

enum ClassId
{
  ListClass,
  StringClass,
  NB_CLASSES
};

/*Class * userTypes[] =
{
  &listClass,
  &stringClass,
};*/
#endif /* UserTypes.h */
```

4.97 /home/thomas/Projects/SParse-master/SParse/src/main.c File Reference

Contains the main() function.

```
#include ¨OptionMgr.h¨
#include ¨FileMgr.h¨
#include ¨TimeMgr.h¨
#include ¨Error.h¨
#include ¨Debug.h¨
#include ¨SParse.h¨
#include ¨Memory.h¨
#include ¨ObjectMgr.h¨
#include <signal.h>
```

Functions

- PRIVATE void print_usage ()

Prints the application help.
- PRIVATE void sighandler (int signum, siginfo_t ∗info, void ∗ptr)

  Display and exit when signal is received.
- PUBLIC int main (const int argc, const char ∗∗argv)

  Inital entry point for the application.

Variables

- struct sigaction **action**

4.97.1 Detailed Description

Contains the main() function.

This file contains only one function main() which initialises the OptionMgr and FileMgr objects. The function also processes each source file in turn.

4.97.2 Function Documentation

4.97.2.1 main()

```
PUBLIC int main (
             const int argc,
             const char ** argv )
```

Inital entry point for the application.

Parameters

argc	Number of arguments
argv	Array of arguments

The main function: 1) Reads the options from command line or file 2) Starts the application for a DB name and an input file directory.

4.97.2.2 print_usage()

```
PRIVATE void print_usage ( )
```

Prints the application help.

Prints the usage information for the aplication.

4.97.2.3 sighandler()

```
PRIVATE void sighandler (
             int signum,
             siginfo_t * info,
             void * ptr )
```

Display and exit when signal is received.

Parameters

signum	TBC
info	TBC
ptr	TBC

This function displays a signal and exit the application.

4.98 Ast.h

4.99 Declarator.h

```
/* Declarator.h */

#ifndef _DECLARATOR_H_
#define _DECLARATOR_H_

typedef enum
{
  E_DEC_FUNCTION,
  E_DEC_VAR,
  E_DEC_TYPE
} DeclaratorType;

typedef struct Declarator Declarator;

Declarator * Declarator_new(DeclaratorType * t)
{
}

void Declarator_delete(Declarator * this)
{
}

#endif /* #ifndef _DECLARATOR_H_
```

4.100 /home/thomas/Projects/SParse-master/SParse/src/ParseLib/↩ Configuration/Configuration.c File Reference

This file contains the implementation for the class Configuration The class Configuration lists all the SW products to parse including the path to the source files, any dependency.

```
#include "Configuration.h"
#include "Product.h"
#include "TransUnit.h"
#include "Grammar.h"
#include "Object.h"
#include "Memory.h"
#include "Error.h"
#include "Debug.h"
```

Classes

- class Configuration

Macros

- #define **DEBUG** (0)
- #define IS_KEY(C)
- #define **IS_COLON**(P) (Memory_ncmp(P, ":", 1))
- #define **IS_LIST**(P) (Memory_ncmp(P, "- ", 2))
- #define **IS_LOCATION_KEY**(P) (Memory_ncmp(P,"Location:", 9))
- #define **IS_INCLUDES_KEY**(P) (Memory_ncmp(P,"Includes:", 9))
- #define **IS_USES_KEY**(P) (Memory_ncmp(P,"Uses:", 5))
- #define **IS_SOURCES_KEY**(P) (Memory_ncmp(P,"Sources:", 8))
- #define **IS_IGNORED**(C) ((C==' ') || (C=='\n') || (C=='\r'))
- #define IS_STRING(C)
- #define **IS_FORBIDDEN**(C) (C=='\t')
- #define **IS_EOL**(P) (Memory_ncmp(P, "\n", 1))

Functions

- PRIVATE List ∗ **Configuration_readProducts** (Configuration ∗this, String ∗s)
- PRIVATE String ∗ **Configuration_readLocation** (Configuration ∗this, String ∗s, unsigned int ∗idx)
- PRIVATE List ∗ **Configuration_readIncludes** (Configuration ∗this, String ∗s, unsigned int ∗idx)
- PRIVATE List ∗ **Configuration_readUses** (Configuration ∗this, String ∗s, unsigned int ∗idx)
- PRIVATE List ∗ **Configuration_readSources** (Configuration ∗this, String ∗s, unsigned int ∗idx)
- PRIVATE String ∗ **Configuration_readString** (Configuration ∗this, String ∗s, unsigned int ∗idx)
- PRIVATE List ∗ **Configuration_readList** (Configuration ∗this, String ∗s, unsigned int ∗idx)
- PRIVATE void **Configuration_readEndOfLine** (Configuration ∗this, String ∗s, unsigned int ∗idx)
- PRIVATE unsigned int **Configuration_readIndent** (Configuration ∗this, String ∗s, unsigned int ∗idx)

4.100.1 Detailed Description

This file contains the implementation for the class Configuration The class Configuration lists all the SW products to parse including the path to the source files, any dependency.

4.100.2 Macro Definition Documentation

4.100.2.1 IS_KEY

```
#define IS_KEY(
            C )
```

Value:

```
(((C>='A') && (C<='Z')) || ((C>='a') && (C<='z')) \
|| ((C>='0') && (C<='9')) || (C=='_'))
```

4.100.2.2 IS_STRING

```
#define IS_STRING(
            C )
```

Value:

```
(((C>='A') && (C<='Z')) || ((C>='a') && (C<='z')) \
|| ((C>='0') && (C<='9')) || (C=='_') || (C=='/') || (C=='\\') || (C=='-') || (C=='.'))
```

4.101 Configuration.h

```
/* Configuration.h */
#ifndef _CONFIGURATION_H_
#define _CONFIGURATION_H_

#include "Types.h"
#include "String2.h"
#include "List.h"
#include "FileMgr.h"

typedef struct Configuration Configuration;

PUBLIC Configuration * Configuration_new(String * input);
PUBLIC void Configuration_delete(Configuration * this);
PUBLIC void Configuration_print(Configuration * this);
PUBLIC unsigned int Configuration_getSize(Configuration * this);
PUBLIC List * Configuration_getProducts(Configuration* this);
#endif /* _CONFIGURATION_H_ */

```

4.102 Configuration.h

```
/* Configuration.h */
#ifndef _CONFIGURATION_H_
#define _CONFIGURATION_H_

#include "Types.h"
#include "String2.h"
#include "List.h"
#include "FileMgr.h"

typedef struct Configuration Configuration;

PUBLIC Configuration * Configuration_new(String * input);
PUBLIC void Configuration_delete(Configuration * this);
PUBLIC void Configuration_print(Configuration * this);
PUBLIC unsigned int Configuration_getSize(Configuration * this);
PUBLIC List * Configuration_getProducts(Configuration* this);
#endif /* _CONFIGURATION_H_ */

```

4.103 /home/thomas/Projects/SParse-master/SParse/src/ParseLib/↩ Configuration/Product.c File Reference

This file contains the implementation for the class Product.

```
#include "Product.h"
#include "FileMgr.h"
#include "String2.h"
#include "Debug.h"
```

Classes

- class Product

Macros

- #define **DEBUG** (0)

Functions

- PUBLIC Product ∗ **Product_new** (String ∗s)
- PUBLIC void **Product_delete** (Product ∗this)
- PUBLIC void **Product_print** (Product ∗this)
- PUBLIC unsigned int **Product_getSize** (Product ∗this)
- PUBLIC String ∗ **Product_getName** (Product ∗this)
- PUBLIC void **Product_setLocation** (Product ∗this, String ∗s)
- PUBLIC String ∗ **Product_getLocation** (Product ∗this)
- PUBLIC void **Product_setIncludes** (Product ∗this, List ∗l)
- PUBLIC void **Product_setUses** (Product ∗this, List ∗l)
- PUBLIC void **Product_setSources** (Product ∗this, List ∗l)
- PUBLIC FileMgr ∗ **Product_getSourceFiles** (Product ∗this)

4.103.1 Detailed Description

This file contains the implementation for the class Product.

The class Product contains the sources for a given product.

4.104 Product.h

```
/* Product.h */
#ifndef _PRODUCT_H_
#define _PRODUCT_H_

#include "FileMgr.h"
#include "List.h"
#include "String2.h"
#include "Object.h"
#include "Debug.h"

typedef struct Product Product;

PUBLIC Product* Product_new(String * this);
PUBLIC void Product_delete(Product * this);
PUBLIC void Product_print(Product * this);
PUBLIC unsigned int Product_getSize(Product * this);
PUBLIC String* Product_getName(Product* this);
PUBLIC void Product_setLocation(Product * this, String * s);
PUBLIC String* Product_getLocation(Product* this);
PUBLIC void Product_setIncludes(Product * this, List * l);
PUBLIC void Product_setUses(Product * this, List * l);
PUBLIC void Product_setSources(Product * this, List * l);
PUBLIC FileMgr * Product_getSourceFiles(Product * this);
#endif /* _PRODUCT_H_ */
```

4.105 Product.h

```
/* Product.h */
#ifndef _PRODUCT_H_
#define _PRODUCT_H_

#include "FileMgr.h"
#include "List.h"
#include "String2.h"
#include "Object.h"
#include "Debug.h"

typedef struct Product Product;

PUBLIC Product* Product_new(String * this);
PUBLIC void Product_delete(Product * this);
PUBLIC void Product_print(Product * this);
PUBLIC unsigned int Product_getSize(Product * this);
PUBLIC String* Product_getName(Product* this);
PUBLIC void Product_setLocation(Product * this, String * s);
PUBLIC String* Product_getLocation(Product* this);
PUBLIC void Product_setIncludes(Product * this, List * l);
PUBLIC void Product_setUses(Product * this, List * l);
PUBLIC void Product_setSources(Product * this, List * l);
PUBLIC FileMgr * Product_getSourceFiles(Product * this);
#endif /* _PRODUCT_H_ */
```

4.106 /home/thomas/Projects/SParse-master/SParse/src/ParseLib/File↩ Reader/FileReader.c File Reference

This file contains the implementation for the class FileReader.

```
#include "FileReader.h"
#include "Class.h"
#include "Object.h"
#include "String2.h"
#include "FileMgr.h"
#include "FileDesc.h"
#include "OptionMgr.h"
#include "List.h"
#include "Error.h"
#include "Memory.h"
```

Classes

- struct IncludeInfo
- class FileReader

Typedefs

- typedef struct IncludeInfo **IncludeInfo**

Functions

- PRIVATE unsigned int **IncludeDir_getSize** (IncludeInfo *this)

Variables

- PRIVATE Class includeInfoClass

4.106.1 Detailed Description

This file contains the implementation for the class FileReader.

The class FileReader is TBD

4.106.2 Variable Documentation

4.106.2.1 includeInfoClass

```
PRIVATE Class includeInfoClass
```

Initial value:

```
=
{
  .f_new = (Constructor)0,
  .f_delete = (Destructor)0,
  .f_copy = (Copy_Operator)0,
  .f_comp = (Comp_Operator)0,
  .f_print = (Printer)0,
  .f_size = (Sizer)&IncludeDir_getSize
}
```

4.107 FileReader.h

```c
/* FileReader.h */

#ifndef _FILEREADER_H_
#define _FILEREADER_H_

#include "Types.h"
#include "String2.h"
#include "FileMgr.h"

typedef struct FileReader FileReader;

PUBLIC FileReader * FileReader_new(FileDesc * file, FileMgr * fileMgr);
PUBLIC void FileReader_delete(FileReader * this);
PUBLIC FileReader * FileReader_copy(FileReader * this);
PUBLIC void FileReader_print(FileReader * this);
PUBLIC unsigned int FileReader_getSize(FileReader * this);
PUBLIC char * FileReader_getBuffer(FileReader * this);
PUBLIC String * FileReader_getName(FileReader * this);
PUBLIC char * FileReader_addFile(FileReader * this, String * fileName);
#endif /* _FILEREADER_H_ */
```

4.108 FileReader.h

```c
/* FileReader.h */

#ifndef _FILEREADER_H_
#define _FILEREADER_H_

#include "Types.h"
#include "String2.h"
#include "FileMgr.h"

typedef struct FileReader FileReader;

PUBLIC FileReader * FileReader_new(FileDesc * file, FileMgr * fileMgr);
PUBLIC void FileReader_delete(FileReader * this);
PUBLIC FileReader * FileReader_copy(FileReader * this);
PUBLIC void FileReader_print(FileReader * this);
PUBLIC unsigned int FileReader_getSize(FileReader * this);
PUBLIC char * FileReader_getBuffer(FileReader * this);
PUBLIC String * FileReader_getName(FileReader * this);
PUBLIC char * FileReader_addFile(FileReader * this, String * fileName);
#endif /* _FILEREADER_H_ */
```

4.109 Grammar.h

```c
/* Grammar.h */
#ifndef _GRAMMAR_H_
#define _GRAMMAR_H_

#include "Types.h"
#include "Object.h"

typedef struct Grammar Grammar;

struct Grammar
{
  Object object;
  Grammar * (*new)(void);
  void (*delete)(Grammar * this);
  Grammar * (*copy)(Grammar * this);
  void (*print)(Grammar * this);
};

PUBLIC Grammar * Grammar_new();
PUBLIC void Grammar_delete(Grammar * this);
PUBLIC void Grammar_process(Grammar * this);
PUBLIC void Grammar_print(Grammar * this);

#endif /* _GRAMMAR_H_ */
```

4.110 Grammar.h

```c
/* Grammar.h */
```

```c
#ifndef _GRAMMAR_H_
#define _GRAMMAR_H_

#include "Types.h"
#include "Object.h"

typedef struct Grammar Grammar;

struct Grammar
{
  Object object;
  Grammar * (*new)(void);
  void (*delete)(Grammar * this);
  Grammar * (*copy)(Grammar * this);
  void (*print)(Grammar * this);
};

PUBLIC Grammar * Grammar_new();
PUBLIC void Grammar_delete(Grammar * this);
PUBLIC void Grammar_process(Grammar * this);
PUBLIC void Grammar_print(Grammar * this);

#endif /* _GRAMMAR_H_ */
```

4.111 Grammar2.h

```c
/* Grammar2.h */

#include "Types.h"
#include "SdbMgr.h"
#include "FileReader.h"

typedef struct Grammar2 Grammar2;

PUBLIC Grammar2 * Grammar2_new(FileReader * fr, SdbMgr * sdbMgr);
PUBLIC void Grammar2_delete(Grammar2 * this);
PUBLIC Grammar2 * Grammar2_copy(Grammar2 * this);
PUBLIC void Grammar2_print(Grammar2 * this);
PUBLIC unsigned int Grammar2_getSize(Grammar2 * this);
PUBLIC void Grammar2_process(Grammar2 * this);
PUBLIC FileReader * Grammar2_getFileReader(Grammar2 * grammar); // Not used
PUBLIC SdbMgr * Grammar2_getSdbMgr(Grammar2 * grammar); // Not used
PUBLIC void Grammar2_addToBuffer(Grammar2 * grammar, char * text); // Used by lex.c
PUBLIC void Grammar2_addComment(Grammar2 * this); // Used by parse.c
PUBLIC void Grammar2_addCodeNode(Grammar2 * this);
PUBLIC void Grammar2_addIncludeNode(Grammar2 * this, char * name);
PUBLIC char * Grammar2_processNewFile(Grammar2 * this, String * fileName); // Used by lex.c
PUBLIC void Grammar2_returnToFile(Grammar2 * this); // Used by lex.c
```

4.112 Grammar2.h

```c
/* Grammar2.h */

#include "Types.h"
#include "SdbMgr.h"
#include "FileReader.h"

typedef struct Grammar2 Grammar2;

PUBLIC Grammar2 * Grammar2_new(FileReader * fr, SdbMgr * sdbMgr);
PUBLIC void Grammar2_delete(Grammar2 * this);
PUBLIC Grammar2 * Grammar2_copy(Grammar2 * this);
PUBLIC void Grammar2_print(Grammar2 * this);
PUBLIC unsigned int Grammar2_getSize(Grammar2 * this);
PUBLIC void Grammar2_process(Grammar2 * this);
PUBLIC FileReader * Grammar2_getFileReader(Grammar2 * grammar); // Not used
PUBLIC SdbMgr * Grammar2_getSdbMgr(Grammar2 * grammar); // Not used
PUBLIC void Grammar2_addToBuffer(Grammar2 * grammar, char * text); // Used by lex.c
PUBLIC void Grammar2_addComment(Grammar2 * this); // Used by parse.c
PUBLIC void Grammar2_addCodeNode(Grammar2 * this);
PUBLIC void Grammar2_addIncludeNode(Grammar2 * this, char * name);
PUBLIC char * Grammar2_processNewFile(Grammar2 * this, String * fileName); // Used by lex.c
PUBLIC void Grammar2_returnToFile(Grammar2 * this); // Used by lex.c
```

4.113 Grammar2.parse.h

```c
/* A Bison parser, made by GNU Bison 3.8.2.  */

/* Bison interface for Yacc-like parsers in C

   Copyright (C) 1984, 1989-1990, 2000-2015, 2018-2021 Free Software Foundation,
   Inc.

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

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

   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <https://www.gnu.org/licenses/>.  */

/* As a special exception, you may create a larger work that contains
   part or all of the Bison parser skeleton and distribute that work
   under terms of your choice, so long as that work isn't itself a
   parser generator using the skeleton or a modified version thereof
   as a parser skeleton.  Alternatively, if you modify or redistribute
   the parser skeleton itself, you may (at your option) remove this
   special exception, which will cause the skeleton and the resulting
   Bison output files to be licensed under the GNU General Public
   License without this special exception.

   This special exception was added by the Free Software Foundation in
   version 2.2 of Bison.  */

/* DO NOT RELY ON FEATURES THAT ARE NOT DOCUMENTED in the manual,
   especially those whose name start with YY_ or yy_.  They are
   private implementation details that can be changed or removed.  */

#ifndef YY_GRAMMAR2_GRAMMAR2_PARSE_H_INCLUDED
# define YY_GRAMMAR2_GRAMMAR2_PARSE_H_INCLUDED
/* Debug traces.  */
#ifndef YYDEBUG
# define YYDEBUG 0
#endif
#if YYDEBUG
extern int Grammar2_debug;
#endif

/* Token kinds.  */
#ifndef YYTOKENTYPE
# define YYTOKENTYPE
  enum yytokentype
  {
    YYEMPTY = -2,
    YYEOF = 0,                     /* "end of file"  */
    YYerror = 256,                 /* error  */
    YYUNDEF = 257,                 /* "invalid token"  */
    COMMENT = 258,                 /* COMMENT  */
    CODE = 259,                    /* CODE  */
    END_OF_UNIT = 260              /* END_OF_UNIT  */
  };
  typedef enum yytokentype yytoken_kind_t;
#endif
/* Token kinds.  */
#define YYEMPTY -2
#define YYEOF 0
#define YYerror 256
#define YYUNDEF 257
#define COMMENT 258
#define CODE 259
#define END_OF_UNIT 260

/* Value type.  */
#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
union YYSTYPE
{
#line 18 "Grammar2.y"

  String * text;

#line 81 "Grammar2.parse.h"

};
typedef union YYSTYPE YYSTYPE;
# define YYSTYPE_IS_TRIVIAL 1
# define YYSTYPE_IS_DECLARED 1
#endif


```

```


int Grammar2_parse (void * scanner, Grammar2 * grammar);


#endif /* !YY_GRAMMAR2_GRAMMAR2_PARSE_H_INCLUDED  */
```

4.114 GrammarC99.h

```
/* GrammarC99.h */
#ifndef _GRAMMARC99_H_
#define _GRAMMARC99_H_

#include "Types.h"
#include "FileMgr.h"
#include "Grammar.h"

typedef struct GrammarC99 GrammarC99;

PUBLIC Grammar* GrammarC99_new(FileDesc * fileDesc, FileMgr * fm);
PUBLIC void GrammarC99_delete(Grammar* this);
PUBLIC void GrammarC99_print(Grammar* this);
PUBLIC unsigned int GrammarC99_getSize(Grammar* this);
PUBLIC void GrammarC99_process(GrammarC99* this);

#endif /* _GRAMMARC99_H_ */
```

4.115 GrammarC99.h

```
/* GrammarC99.h */
#ifndef _GRAMMARC99_H_
#define _GRAMMARC99_H_

#include "Types.h"
#include "FileMgr.h"
#include "Grammar.h"

typedef struct GrammarC99 GrammarC99;

PUBLIC Grammar* GrammarC99_new(FileDesc * fileDesc, FileMgr * fm);
PUBLIC void GrammarC99_delete(Grammar* this);
PUBLIC void GrammarC99_print(Grammar* this);
PUBLIC unsigned int GrammarC99_getSize(Grammar* this);
PUBLIC void GrammarC99_process(GrammarC99* this);

#endif /* _GRAMMARC99_H_ */
```

4.116 GrammarC99.parse.h

```
/* A Bison parser, made by GNU Bison 3.8.2.  */

/* Bison interface for Yacc-like parsers in C

   Copyright (C) 1984, 1989-1990, 2000-2015, 2018-2021 Free Software Foundation,
   Inc.

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

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

   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <https://www.gnu.org/licenses/>.  */

/* As a special exception, you may create a larger work that contains
   part or all of the Bison parser skeleton and distribute that work
   under terms of your choice, so long as that work isn't itself a
   parser generator using the skeleton or a modified version thereof
   as a parser skeleton.  Alternatively, if you modify or redistribute
   the parser skeleton itself, you may (at your option) remove this
   special exception, which will cause the skeleton and the resulting
   Bison output files to be licensed under the GNU General Public
   License without this special exception.

   This special exception was added by the Free Software Foundation in
   version 2.2 of Bison.  */

/* DO NOT RELY ON FEATURES THAT ARE NOT DOCUMENTED in the manual,
   especially those whose name start with YY_ or yy_.  They are
   private implementation details that can be changed or removed.  */

#ifndef YY_GRAMMARC99_GRAMMARC99_PARSE_H_INCLUDED
# define YY_GRAMMARC99_GRAMMARC99_PARSE_H_INCLUDED
/* Debug traces.  */
#ifndef GRAMMARC99_DEBUG
# if defined YYDEBUG
#if YYDEBUG
#   define GRAMMARC99_DEBUG 1
#  else
#   define GRAMMARC99_DEBUG 0
#  endif
# else /* ! defined YYDEBUG */
#  define GRAMMARC99_DEBUG 0
# endif /* ! defined YYDEBUG */
#endif  /* ! defined GRAMMARC99_DEBUG */
#if GRAMMARC99_DEBUG
extern int GrammarC99_debug;
#endif
/* "%code requires" blocks.  */
#line 19 "GrammarC99.y"
 #include "MyType.h"

#line 60 "GrammarC99.parse.h"

/* Token kinds.  */
#ifndef GRAMMARC99_TOKENTYPE
# define GRAMMARC99_TOKENTYPE
  enum GrammarC99_tokentype
  {
    GRAMMARC99_EMPTY = -2,
    GRAMMARC99_EOF = 0,            /* "end of file"  */
    GRAMMARC99_error = 256,        /* error  */
    GRAMMARC99_UNDEF = 257,        /* "invalid token"  */
    IDENTIFIER = 258,              /* IDENTIFIER  */
    CONSTANT = 259,                /* CONSTANT  */
    STRING_LITERAL = 260,          /* STRING_LITERAL  */
    SIZEOF = 261,                  /* SIZEOF  */
    PTR_OP = 262,                  /* PTR_OP  */
    INC_OP = 263,                  /* INC_OP  */
    DEC_OP = 264,                  /* DEC_OP  */
    LEFT_OP = 265,                 /* LEFT_OP  */
    RIGHT_OP = 266,                /* RIGHT_OP  */
    LE_OP = 267,                   /* LE_OP  */
    GE_OP = 268,                   /* GE_OP  */
    EQ_OP = 269,                   /* EQ_OP  */
    NE_OP = 270,                   /* NE_OP  */
    AND_OP = 271,                  /* AND_OP  */
    OR_OP = 272,                   /* OR_OP  */
    MUL_ASSIGN = 273,              /* MUL_ASSIGN  */
    DIV_ASSIGN = 274,              /* DIV_ASSIGN  */
    MOD_ASSIGN = 275,              /* MOD_ASSIGN  */
    ADD_ASSIGN = 276,              /* ADD_ASSIGN  */
    SUB_ASSIGN = 277,              /* SUB_ASSIGN  */
    LEFT_ASSIGN = 278,             /* LEFT_ASSIGN  */
    RIGHT_ASSIGN = 279,            /* RIGHT_ASSIGN  */
    AND_ASSIGN = 280,              /* AND_ASSIGN  */
    XOR_ASSIGN = 281,              /* XOR_ASSIGN  */
    OR_ASSIGN = 282,               /* OR_ASSIGN  */
    TYPE_NAME = 283,               /* TYPE_NAME  */
    TYPEDEF = 284,                 /* TYPEDEF  */
    EXTERN = 285,                  /* EXTERN  */
    STATIC = 286,                  /* STATIC  */
    AUTO = 287,                    /* AUTO  */
    REGISTER = 288,                /* REGISTER  */
    INLINE = 289,                  /* INLINE  */
    RESTRICT = 290,                /* RESTRICT  */
    CHAR = 291,                    /* CHAR  */
    SHORT = 292,                   /* SHORT  */
    INT = 293,                     /* INT  */
    LONG = 294,                    /* LONG  */
    SIGNED = 295,                  /* SIGNED  */
    UNSIGNED = 296,                /* UNSIGNED  */
    FLOAT = 297,                   /* FLOAT  */
    DOUBLE = 298,                  /* DOUBLE  */
    CONST = 299,                   /* CONST  */
    VOLATILE = 300,                /* VOLATILE  */
    VOID = 301,                    /* VOID  */
    BOOL = 302,                    /* BOOL  */
    COMPLEX = 303,                 /* COMPLEX  */
    IMAGINARY = 304,               /* IMAGINARY  */
    STRUCT = 305,                  /* STRUCT  */
    UNION = 306,                   /* UNION  */
    ENUM = 307,                    /* ENUM  */
    ELLIPSIS = 308,                /* ELLIPSIS  */
    CASE = 309,                    /* CASE  */
    DEFAULT = 310,                 /* DEFAULT  */
    IF = 311,                      /* IF  */
    ELSE = 312,                    /* ELSE  */
    SWITCH = 313,                  /* SWITCH  */
    WHILE = 314,                   /* WHILE  */
    DO = 315,                      /* DO  */
    FOR = 316,                     /* FOR  */
    GOTO = 317,                    /* GOTO  */
    CONTINUE = 318,                /* CONTINUE  */
    BREAK = 319,                   /* BREAK  */
    RETURN = 320                   /* RETURN  */
  };
  typedef enum GrammarC99_tokentype GrammarC99_token_kind_t;
#endif

/* Value type.  */
#if ! defined GRAMMARC99_STYPE && ! defined GRAMMARC99_STYPE_IS_DECLARED
typedef  MyType  GRAMMARC99_STYPE;
# define GRAMMARC99_STYPE_IS_TRIVIAL 1
# define GRAMMARC99_STYPE_IS_DECLARED 1
#endif




int GrammarC99_parse (void * scanner, GrammarC99 * grammar);


#endif /* !YY_GRAMMARC99_GRAMMARC99_PARSE_H_INCLUDED  */
```

4.117 MyType.h

```
typedef struct MyType MyType;

struct MyType {
  char * sval;
};
```

4.118 HTTPRequest.h

```
/* HTTPRequest.h */
#ifndef _HTTPREQUEST_H_
#define _HTTPREQUEST_H_

#include "Object.h"
#include "Types.h"
#include "Class.h"
#include "String2.h"
#include "Map.h"
#include "Memory.h"
#include "Debug.h"

#include <stdio.h>
#include <stdlib.h>

typedef struct HTTPRequest HTTPRequest;

PRIVATE HTTPRequest * HTTPRequest_new(char * buffer);
PRIVATE void HTTPRequest_delete(HTTPRequest * this);
PRIVATE HTTPRequest * HTTPRequest_copy(HTTPRequest * this);
PRIVATE int HTTPRequest_compare(HTTPRequest* this, HTTPRequest* compared);
PRIVATE void HTTPRequest_print(HTTPRequest * this);
PRIVATE unsigned int HTTPRequest_getSize(HTTPRequest* this);
PRIVATE String* HTTPRequest_getPath(HTTPRequest* this);
PRIVATE enum Method HTTPRequest_getMethod(HTTPRequest* this);
PRIVATE int HTTPRequest_isValid(HTTPRequest* this);
PRIVATE int HTTPRequest_parseBuffer(HTTPRequest* this, char* buffer);

enum Method
{
  METHOD_GET=0,
  METHOD_POST,
  METHOD_PUT,
  METHOD_PATCH,
  METHOD_DELETE,
  METHOD_INVALID
};

static char* methods_text[] = { "GET", "POST", "PUT", "PATCH", "DELETE" };
/**********************************************/
struct HTTPRequest
{
  Object object;
  enum Method method;
  String * path;
  int majorVersion;
  int minorVersion;
  Map* headers;
  String * body;
  int isValid;
};

/**********************************************/
PRIVATE Class httpRequestClass =
{
  .f_new = 0,
  .f_delete = (Destructor)&HTTPRequest_delete,
  .f_copy = (Copy_Operator)&HTTPRequest_copy,
  .f_comp = (Comp_Operator)&HTTPRequest_compare,
  .f_print = (Printer)&HTTPRequest_print,
  .f_size = (Sizer)&HTTPRequest_getSize
};

/**********************************************/
PRIVATE HTTPRequest * HTTPRequest_new(char * buffer)
{
  HTTPRequest* this = 0;

  this = (HTTPRequest*)Object_new(sizeof(HTTPRequest), &httpRequestClass);

  if (this == 0) return 0;

  this->method = METHOD_INVALID;
  this->path = 0;
  this->majorVersion = 0;
  this->minorVersion = 0;
  this->headers = Map_new();
  this->body = 0;
  this->isValid = HTTPRequest_parseBuffer(this, buffer);

  return this;
}

/**********************************************/
PRIVATE void HTTPRequest_delete(HTTPRequest * this)
{
  if (!Object_isValid((Object*)this)) return;

  String_delete(this->path);
  Map_delete(this->headers);
}

/**********************************************/
PRIVATE HTTPRequest * HTTPRequest_copy(HTTPRequest * this)
{
  return 0;
}

/**********************************************/
PRIVATE int HTTPRequest_compare(HTTPRequest * this, HTTPRequest * compared)
{
  return 0;
}

/**********************************************/
PRIVATE void HTTPRequest_print(HTTPRequest * this)
{
  if (this->method == METHOD_INVALID) PRINT(("Method: INVALID\n"));
  if (this->method == METHOD_GET) PRINT(("Method: GET\n"));
  if (this->method == METHOD_POST) PRINT(("Method: POST\n"));
  PRINT(("Path: %s\n", String_getBuffer(this->path)));
  PRINT(("Version: %d.%d\n", this->majorVersion, this->minorVersion));
  //PRINT(("Host: %s\n", host));
  //PRINT(("User-Agent: %s\n", userAgent));
}

/**********************************************/
PRIVATE unsigned int HTTPRequest_getSize(HTTPRequest * this)
```

```
{
  return sizeof(HTTPRequest);
}

PRIVATE String * HTTPRequest_getPath(HTTPRequest* this)
{
  return this->path;
}

PRIVATE enum Method HTTPRequest_getMethod(HTTPRequest* this)
{
  return this->method;
}

PRIVATE int HTTPRequest_isValid(HTTPRequest* this)
{
  return this->isValid;
}

PRIVATE int HTTPRequest_parseBuffer(HTTPRequest* this, char* buffer)
{
  int isValid = 0;
  char * path_start = buffer;
  int path_length = 0;

  /* Read method*/
  for (enum Method i = METHOD_GET; i < METHOD_INVALID; i++)
  {
    if (Memory_ncmp(buffer, methods_text[i], sizeof(methods_text[i]) - 1))
    {
      this->method = i;
      path_start = buffer + sizeof(methods_text[i]);
      isValid = 1;
      break;
    }
  }

  /* Read path */
  while ((path_length < (int)Memory_len(buffer)) && (*(path_start + path_length) != ' '))
  {
    path_length++;
  }

  char* path_buffer = Memory_alloc(path_length + 1);
  Memory_copy(path_buffer, path_start, path_length + 1);
  path_buffer[path_length + 1] = 0;

  this->path = String_newByRef(path_buffer);
  char * version_start = path_start + path_length + 1;

  /* Read version */
  if (Memory_ncmp(version_start, "HTTP/1.1", sizeof("HTTP/1.1") - 1))
  {
    this->majorVersion = 1;
    this->minorVersion = 1;
    isValid = isValid && 1;
  }

  return isValid;
}
#endif /* _HTTPREQUEST_H_ */
```

4.119 HTTPResponse.h

```
/* HTTPResponse.h */
#ifndef _HTTPRESPONSE_H_
#define _HTTPRESPONSE_H_

#include "Object.h"
#include "Types.h"
#include "Class.h"
#include "String2.h"
#include "Map.h"
#include <stdio.h>

typedef struct HTTPResponse HTTPResponse;

enum Reason
{
  REASON_OK,
  REASON_INVALID
};

```

```
PRIVATE HTTPResponse * HTTPResponse_new();
PRIVATE void HTTPResponse_delete(HTTPResponse * this);
PRIVATE HTTPResponse* HTTPResponse_copy(HTTPResponse* this);
PRIVATE int HTTPResponse_compare(HTTPResponse* this, HTTPResponse* compared);
PRIVATE void HTTPResponse_print(HTTPResponse* this);
PRIVATE unsigned int HTTPResponse_getSize(HTTPResponse* this);
PRIVATE void HTTPResponse_setVersion(HTTPResponse* this, int majorVersion, int minorVersion);
PRIVATE void HTTPResponse_setStatusCode(HTTPResponse* this, int code);
PRIVATE void HTTPResponse_setReason(HTTPResponse* this, enum Reason);
PRIVATE void HTTPResponse_addHeader(HTTPResponse* this, char* key, char* value);
PRIVATE void HTTPResponse_setBody(HTTPResponse* this, char* body);
PRIVATE int HTTPResponse_generate(HTTPResponse* this, char* buffer, int size);

/**********************************************/
struct HTTPResponse
{
  Object object;
  int statusCode;
  enum Reason reason;
  int majorVersion;
  int minorVersion;
  Map* headers;
  String* body;
  int isValid;
};

/**********************************************/
PRIVATE Class httpResponseClass =
{
  .f_new = 0,
  .f_delete = (Destructor)&HTTPResponse_delete,
  .f_copy = (Copy_Operator)&HTTPResponse_copy,
  .f_comp = (Comp_Operator)&HTTPResponse_compare,
  .f_print = (Printer)&HTTPResponse_print,
  .f_size = (Sizer)&HTTPResponse_getSize
};

/**********************************************/
PRIVATE HTTPResponse* HTTPResponse_new()
{
  HTTPResponse* this = 0;

  this = (HTTPResponse*)Object_new(sizeof(HTTPResponse), &httpResponseClass);

  if (this == 0) return 0;

  this->statusCode = REASON_INVALID;
  this->majorVersion = 0;
  this->minorVersion = 0;
  this->headers = Map_new();
  this->body = 0;
  this->isValid = 0;

  return this;
}

/**********************************************/
PRIVATE void HTTPResponse_delete(HTTPResponse* this)
{
  if (!Object_isValid((Object*)this)) return;

  String_delete(this->body);
  Map_delete(this->headers);
}

/**********************************************/
PRIVATE HTTPResponse* HTTPResponse_copy(HTTPResponse* this)
{
  return 0;
}

/**********************************************/
PRIVATE int HTTPResponse_compare(HTTPResponse* this, HTTPResponse* compared)
{
  return 0;
}

/**********************************************/
PRIVATE void HTTPResponse_print(HTTPResponse* this)
{

}

/**********************************************/
PRIVATE unsigned int HTTPResponse_getSize(HTTPResponse* this)
{
  return sizeof(HTTPResponse);
}

PRIVATE void HTTPResponse_setReason(HTTPResponse* this, enum Reason reason)
{
  this-> reason = REASON_OK;
}

PRIVATE void HTTPResponse_setStatusCode(HTTPResponse* this, int statusCode)
{
  this->statusCode = statusCode;
}

PRIVATE void HTTPResponse_setVersion(HTTPResponse* this, int majorVersion, int minorVersion)
{
  this->majorVersion = majorVersion;
  this->minorVersion = minorVersion;
}

PRIVATE void HTTPResponse_addHeader(HTTPResponse* this, char* key, char* value)
{

}

PRIVATE void HTTPResponse_setBody(HTTPResponse* this, char* body)
{
   this->body = String_newByRef(body);
}

PRIVATE int HTTPResponse_generate(HTTPResponse* this, char * buffer, int size)
{
  char test_response[] = "HTTP/1.1 200 OK\r\n"
    "Content-Type: text/html; charset=UTF-8\r\n\r\n"
    "<doctype !html><html><head><title>Hello World</title></head>"
    "<body><h1>Hello world!</h1></body></html>\r\n";

  int nbCharToWrite = snprintf(buffer, size,"HTTP/%d.%d %d OK\r\nContent-Type: text/html;
charset=UTF-8\r\n\r\n%s", this->majorVersion, this->minorVersion, this->statusCode,
String_getBuffer(this->body));

  return nbCharToWrite;
}
#endif /* _HTTPRESPONSE_H_ */
```

4.120 /home/thomas/Projects/SParse-master/SParse/src/ParseLib/↩ HTTPServer/HTTPServer.c File Reference

A HTTP Server class. This class provides server function to create, start HTML pages.

```
#include "HTTPServer.h"
#include "HTTPRequest.h"
#include "HTTPResponse.h"
#include "TaskMgr.h"
#include "Task.h"
#include "Object.h"
#include "Memory.h"
#include "Debug.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include <errno.h>
```

Classes

- class HTTPServer
- struct ConnectionParam

Macros

- #define **REQUEST_BUFFER_SIZE** (4096)
- #define **RESPONSE_BUFFER_SIZE** (4096)

Functions

- int **msleep** (long msec)
- PRIVATE void ∗ **HTTPServer_listenTaskBody** (void ∗params)
- PUBLIC void **HTTPServer_delete** (HTTPServer ∗this)
- PUBLIC HTTPServer ∗ **HTTPServer_copy** (HTTPServer ∗this)
- PUBLIC int **HTTPServer_compare** (HTTPServer ∗this, HTTPServer ∗compared)
- PUBLIC void **HTTPServer_print** (HTTPServer ∗this)
- PUBLIC unsigned int **HTTPServer_getSize** (HTTPServer ∗this)
- PUBLIC void **HTTPServer_start** (HTTPServer ∗this)

4.120.1 Detailed Description

A HTTP Server class. This class provides server function to create, start HTML pages.

4.121 HTTPServer.h

```
/* HTTPServer.h */
#ifndef _HTTPSERVER_H_
#define _HTTPSERVER_H_

#include "Types.h"

typedef struct HTTPServer HTTPServer;

PUBLIC HTTPServer * HTTPServer_new();
PUBLIC void HTTPServer_delete(HTTPServer * this);
PUBLIC HTTPServer* HTTPServer_copy(HTTPServer* this);
PUBLIC int HTTPServer_compare(HTTPServer* this, HTTPServer* compared);
PUBLIC void HTTPServer_print(HTTPServer* this);
PUBLIC void HTTPServer_start(HTTPServer* this);
PUBLIC unsigned int HTTPServer_getSize(HTTPServer* this);
//PUBLIC void HTTPServer_start(HTTPServer* this);
#endif /* _HTTPSERVER_H_ */
```

4.122 HTTPServer.h

```
/* HTTPServer.h */
#ifndef _HTTPSERVER_H_
#define _HTTPSERVER_H_

#include "Types.h"

typedef struct HTTPServer HTTPServer;

PUBLIC HTTPServer * HTTPServer_new();
PUBLIC void HTTPServer_delete(HTTPServer * this);
PUBLIC HTTPServer* HTTPServer_copy(HTTPServer* this);
PUBLIC int HTTPServer_compare(HTTPServer* this, HTTPServer* compared);
PUBLIC void HTTPServer_print(HTTPServer* this);
PUBLIC void HTTPServer_start(HTTPServer* this);
PUBLIC unsigned int HTTPServer_getSize(HTTPServer* this);
//PUBLIC void HTTPServer_start(HTTPServer* this);
#endif /* _HTTPSERVER_H_ */
```

4.123 /home/thomas/Projects/SParse-master/SParse/src/ParseLib/↩ SParse/SParse.c File Reference

This file contains the implementation for the class SParse.

```
#include "SParse.h"
#include "Class.h"
#include "Object.h"
#include "FileReader.h"
#include "TransUnit.h"
#include "Configuration.h"
#include "Product.h"
#include "SdbMgr.h"
#include "Error.h"
#include "Grammar2.h"
#include "FileMgr.h"
#include "FileDesc.h"
#include "List.h"
#include "OptionMgr.h"
#include "Debug.h"
```

Classes

- class SParse

Functions

- PRIVATE unsigned int **SParse_parseFile** (SParse ∗this, FileDesc ∗fileDesc, FileMgr ∗fileMgr)
- PUBLIC unsigned int **SParse_getSize** (SParse ∗this)

4.123.1 Detailed Description

This file contains the implementation for the class SParse.

The class SParse parses all files with extension .X and stores the result of the parsing in the SQLite DB name.

4.124 SParse.h

```
/* SParse.h */

#ifndef _SPARSE_H_
#define _SPARSE_H_

#include "Types.h"

typedef struct SParse SParse;

PUBLIC SParse *SParse_new(/* Sdb name */);
PUBLIC void SParse_delete(SParse * this);
PUBLIC SParse * SParse_copy(SParse * this);
PUBLIC void SParse_print(SParse * this);
PUBLIC unsigned int SParse_getSize(SParse * this);
PUBLIC unsigned int SParse_parse(SParse * this, const char * extension);

#endif /* _SPARSE_H_ */
```

4.125 SParse.h

```
/* SParse.h */

#ifndef _SPARSE_H_
#define _SPARSE_H_

#include "Types.h"

typedef struct SParse SParse;

PUBLIC SParse *SParse_new(/* Sdb name */);
PUBLIC void SParse_delete(SParse * this);
PUBLIC SParse * SParse_copy(SParse * this);
PUBLIC void SParse_print(SParse * this);
PUBLIC unsigned int SParse_getSize(SParse * this);
PUBLIC unsigned int SParse_parse(SParse * this, const char * extension);

#endif /* _SPARSE_H_ */
```

4.126 Buffer.h

```
/* Buffer.h */
#ifndef _BUFFER_H_
#define _BUFFER_H_

#include "Types.h"
#include "Class.h"
#include "Object.h"
#include "Debug.h"

typedef struct Buffer Buffer;

PRIVATE Buffer * Buffer_new();
PRIVATE void Buffer_delete(Buffer * this);
PRIVATE void Buffer_print(Buffer * this);
PRIVATE unsigned int Buffer_getSize(Buffer * this);

struct Buffer
{
  Object object;
  String* string;
  char* currentPtr;
  char* startPtr;
  int nbCharRead;
};

PRIVATE Class bufferClass =
{
  .f_new = (Constructor)0,
  .f_delete = (Destructor)&Buffer_delete,
  .f_copy = (Copy_Operator)0,
  .f_comp = (Comp_Operator)0,
  .f_print = (Printer)&Buffer_print,
  .f_size = (Sizer)&Buffer_getSize
};

PRIVATE Buffer * Buffer_new(String * content)
{
  Buffer * this = 0;

  this = (Buffer*)Object_new(sizeof(Buffer), &bufferClass);

  this->string = content;
  this->startPtr = String_getBuffer(this->string);
  this->currentPtr = this->startPtr;
  this->nbCharRead = 0;

  return this;
}

PRIVATE void Buffer_delete(Buffer * this)
{
  if (this == 0) return;

  /* De-allocate the specific members */
  String_delete(this->string);
  this->startPtr = 0;
  this->currentPtr = 0;
  this->nbCharRead = 0;
  /* De-allocate the base object */
  Object_deallocate(&this->object);
}
```

```

PRIVATE void Buffer_print(Buffer * this)
{

}

PRIVATE unsigned int Buffer_getSize(Buffer * this)
{
  return sizeof(this);
}
#endif /* _BUFFER_H_ */
```

4.127 MacroDefinition.h

```
/* MacroDefinition.h */
#ifndef _MACRODEFINITION_H_
#define _MACRODEFINITION_H_

#include "Types.h"
#include "Class.h"
#include "Object.h"

typedef struct MacroDefinition MacroDefinition;

PRIVATE MacroDefinition * MacroDefinition_new(List * parameters, String * body);
PRIVATE void MacroDefinition_delete(MacroDefinition * this);
PRIVATE void MacroDefinition_print(MacroDefinition * this);
PRIVATE unsigned int MacroDefinition_getSize(MacroDefinition * this);

struct MacroDefinition
{
  Object object;
  String* body;
  List* parameters;
};

PRIVATE Class macroDefinitionClass =
{
  .f_new = (Constructor)0,
  .f_delete = (Destructor)&MacroDefinition_delete,
  .f_copy = (Copy_Operator)0,
  .f_comp = (Comp_Operator)0,
  .f_print = (Printer)&MacroDefinition_print,
  .f_size = (Sizer)&MacroDefinition_getSize
};

PRIVATE MacroDefinition* MacroDefinition_new(List * parameters, String * body)
{

  MacroDefinition * this = (MacroDefinition*)Object_new(sizeof(MacroDefinition),
&macroDefinitionClass);

  this->parameters = parameters;
  this->body = body;

  return this;
}

PRIVATE void MacroDefinition_delete(MacroDefinition* this)
{
  if (this == 0) return;
  /* De-allocate the specific members */
  List_delete(this->parameters);
  String_delete(this->body);

  /* De-allocate the base object */
  Object_deallocate(&this->object);
}

PRIVATE void MacroDefinition_print(MacroDefinition* this)
{

}

PRIVATE unsigned int MacroDefinition_getSize(MacroDefinition* this)
{
  return sizeof(MacroDefinition);
}
#endif /* _MACRODEFINITION_H_ */
```

4.128 MacroStore.h

```
/* MacroStore.h */
#ifndef _MACROSTORE_H_
#define _MACROSTORE_H_

#include "Types.h"
#include "Class.h"
#include "Object.h"
#include "String2.h"
#include "Memory.h"
#include "MacroDefinition.h"
#include "Debug.h"

#define MAX_CHILDREN (28)

typedef struct MacroStore MacroStore;


PRIVATE char convert[256];
PRIVATE char convertBack[MAX_CHILDREN];

enum MacroEvalName
{
  E_NOT_MACRO,
  E_POSSIBLE_MACRO,
  E_DEFINED_MACRO
};

struct MacroStoreNode
{
  int isLeaf;
  MacroDefinition * def;
  void * children[MAX_CHILDREN];
};

struct MacroStore
{
  Object object;
  struct MacroStoreNode * root;
};

PRIVATE MacroStore * MacroStore_new();
PRIVATE void MacroStore_delete(MacroStore * this);
PRIVATE void MacroStore_print(MacroStore * this);
PRIVATE unsigned int MacroStore_getSize(MacroStore * this);
PRIVATE int MacroStore_insertName(MacroStore* this, String* name, MacroDefinition* body);
PRIVATE int MacroStore_isDefName(MacroStore* this, String* name);
PRIVATE int MacroStore_removeName(MacroStore* this, String* name);
PRIVATE enum MacroEvalName MacroStore_evalName(MacroStore* this, char* ptr, int length);
PRIVATE void MacroStore_printChildrenNodes(MacroStore* this, struct MacroStoreNode* node, char* name,
int l);
PRIVATE void MacroStore_deleteChildrenNodes(MacroStore* this, struct MacroStoreNode* node);
PRIVATE String * MacroStore_expandMacro(MacroStore * this, String * inStr);

PRIVATE Class macroStoreClass =
{
  .f_new = (Constructor)0,
  .f_delete = (Destructor)&MacroStore_delete,
  .f_copy = (Copy_Operator)0,
  .f_comp = (Comp_Operator)0,
  .f_print = (Printer)&MacroStore_print,
  .f_size = (Sizer)&MacroStore_getSize
};



PRIVATE MacroStore* MacroStore_new()
{
  MacroStore * this = (MacroStore*)Object_new(sizeof(MacroStore), &macroStoreClass);

  this->root = (struct MacroStoreNode * )Memory_alloc(sizeof(struct MacroStoreNode));

  for (int i = 0; i < MAX_CHILDREN; i++)
  {
    this->root->children[i] = 0;
    this->root->isLeaf = 1;
    this->root->def = 0;
  }

  for (int c = 0; c < 255; c++)
  {
    if ((c >= 'A') && (c <= 'Z'))
    {
      convert[c] = c - 'A' + 2;
      convertBack[c - 'A' + 2] = c;
    }
    else if (c == '_')
    {
      convert[c] = 1;
      convertBack[1] = c;
    }
    else
    {
      convert[c] = 0;
      convertBack[0] = 0;
    }
  }
  return this;
}

PRIVATE void MacroStore_delete(MacroStore* this)
{
  if (this == 0) return;
  /* De-allocate the specific members */
  for (int i = 0; i < MAX_CHILDREN; i++)
  {
    if (this->root->children[i])
    {
      MacroStore_deleteChildrenNodes(this, this->root->children[i]);
      Memory_free(this->root->children[i], sizeof(struct MacroStoreNode*));
      this->root->children[i] = 0;
    }
  }
  Memory_free(this->root, sizeof(struct MacroStoreNode*));

  /* De-allocate the base object */
  Object_deallocate(&this->object);
}

PRIVATE void MacroStore_print(MacroStore* this)
{
  struct MacroStoreNode* currentNode = this->root;
  char * name = Memory_alloc(256); // MAX Macro name length
  int l = 0;
  for (int i = 0; i < MAX_CHILDREN; i++)
  {
    if (currentNode->children[i])
    {
      name[l] = convertBack[i];
      if (!currentNode->isLeaf)
        MacroStore_printChildrenNodes(this, currentNode->children[i], name, l + 1);
      else
      {
        name[l] = 0;
      }
      //PRINT(("%s\n", name));
    }
  }
  Memory_free(name, 256);
}

PRIVATE unsigned int MacroStore_getSize(MacroStore* this)
{
  return sizeof(MacroStore);
}

PRIVATE int MacroStore_insertName(MacroStore* this, String * name, MacroDefinition* body)
{
  char* buffer = String_getBuffer(name);
  int length = String_getLength(name);
  struct MacroStoreNode* currentNode = this->root;

  int c = 0;
  for (int i = 0; i < length; i++)
  {
    c = convert[buffer[i]];
    if (currentNode->isLeaf)
    {
      currentNode->isLeaf = 0;
      currentNode->children[c] = Memory_alloc(sizeof(struct MacroStoreNode));
      currentNode = currentNode->children[c];
      for (int c = 0; c < MAX_CHILDREN; c++)
        currentNode->children[c] = 0;
      currentNode->isLeaf = 1;
      currentNode->def = 0;
    }
    else if (currentNode->children[c])
      currentNode = currentNode->children[c];
    else
    {
      currentNode->children[c] = Memory_alloc(sizeof(struct MacroStoreNode));
      currentNode = currentNode->children[c];
      for (int c = 0; c < MAX_CHILDREN; c++)
```

```
        currentNode->children[c] = 0;
      currentNode->isLeaf = 1;
      currentNode->def = 0;
    }
  }
  currentNode->isLeaf = 1;
  currentNode->def = body;

  return 0;
}

PRIVATE int MacroStore_isDefName(MacroStore* this, String* name)
{
  char* buffer = String_getBuffer(name);
  int length = String_getLength(name);
  struct MacroStoreNode* currentNode = this->root;

  for (int i = 0; i < length; i++)
  {
    int c = convert[buffer[i]];
    if (currentNode->children[c]) currentNode = currentNode->children[c];
    else
      return 0; // Not found
  }
  if (currentNode->def) return 1; // Found
  return 0;
}
PRIVATE int MacroStore_removeName(MacroStore* this, String* name)
{
  char* buffer = String_getBuffer(name);
  int length = String_getLength(name);
  struct MacroStoreNode* currentNode = this->root;

  if (currentNode->isLeaf) return 0;

  for (int i = 0; i < length; i++)
  {
   int  c = convert[buffer[i]];
  }
  return 0;
}

PRIVATE enum MacroEvalName MacroStore_evalName(MacroStore* this, char* buffer, int length)
{
  if (this == 0) return E_NOT_MACRO;
  if (length <= 0) return E_NOT_MACRO;
  struct MacroStoreNode* currentNode = this->root;

  int c = 0;
  int i;
  for (i = 0; i < length; i++)
  {
    c = convert[buffer[i]];
    if (currentNode->children[c])
      currentNode = currentNode->children[c];
    else
      return E_NOT_MACRO;
  }
  if (currentNode->def != 0) return E_DEFINED_MACRO;

  return E_POSSIBLE_MACRO;
}

PRIVATE void MacroStore_printChildrenNodes(MacroStore* this, struct MacroStoreNode* node, char* name,
      int l)
{
  if (node == 0) return;

  if (node->isLeaf)
  {
    name[l] = 0;
    PRINT(("%s\n", name));
    return;
  }
  else
  {
    if (node->def)
    {
      name[l] = 0;
      PRINT(("%s\n", name));
    }
    for (int i = 0; i < MAX_CHILDREN; i++)
    {
      name[l] = convertBack[i];
      if (node->children[i]) MacroStore_printChildrenNodes(this, node->children[i], name, l + 1);
    }
  }
```

```
}
PRIVATE void MacroStore_deleteChildrenNodes(MacroStore* this, struct MacroStoreNode* node)
{
  if (node == 0) return;
  MacroDefinition_delete(node->def);
  if (node->isLeaf) return;
  for (int i = 0; i < MAX_CHILDREN; i++)
  {
    if (node->children[i])
    {
      MacroStore_deleteChildrenNodes(this, node->children[i]);
      Memory_free(node->children[i], sizeof(struct MacroStoreNode*));
      node->children[i] = 0;
    }
  }
}
PRIVATE String* MacroStore_expandMacro(MacroStore* this, String* inStr)
{
  int length = 1;
  enum MacroEvalName status;
  status = MacroStore_evalName(this, String_getBuffer(inStr), length);
  if (status == E_NOT_MACRO) return 0;
  while ((status == E_POSSIBLE_MACRO) && (length<String_getLength(inStr)))
  {
    length++;
    status = MacroStore_evalName(this, String_getBuffer(inStr), length);
  }
  if (status == E_POSSIBLE_MACRO) return 0;
  return 0;
}
#endif /* _MACROSTORE_H_ */
```

4.129 test.h

```
void header();
```

4.130 /home/thomas/Projects/SParse-master/SParse/src/ParseLib/↩ TransUnit/TransUnit.c File Reference

This file implements a class that extract C code.

```
#include "TransUnit.h"
#include "MacroDefinition.h"
#include "MacroStore.h"
#include "Buffer.h"
#include "List.h"
#include "Map.h"
#include "String2.h"
#include "Memory.h"
#include "Error.h"
#include "Object.h"
#include "Debug.h"
```

Classes

- class TransUnit

Macros

- #define **DEBUG** (0)
- #define **IS_MACRO_LETTER**(C) ((((C)>='A') && ((C)<='Z')) || (((C)>='a') && ((C)<='z')) || ((C)=='_'))
- #define **OUTPUT_BUFFER_SIZE** (20000)

Functions

- PRIVATE void **TransUnit_consumeLineComment** (TransUnit ∗this)
- PRIVATE void **TransUnit_consumeMultilineComment** (TransUnit ∗this)
- PRIVATE void **TransUnit_consumeInclude** (TransUnit ∗this)
- PRIVATE void **TransUnit_readMacroDefinition** (TransUnit ∗this)
- PRIVATE void **TransUnit_checkMacro** (TransUnit ∗this, int checkForTrue)
- PRIVATE int **TransUnit_pushNewBuffer** (TransUnit ∗this, String ∗content)
- PRIVATE int **TransUnit_popBuffer** (TransUnit ∗this)
- PRIVATE int **TransUnit_expandMacro** (TransUnit ∗this)

4.130.1 Detailed Description

This file implements a class that extract C code.

It removes the comments, expands the macros, parses the include files

4.131 TransUnit.h

```
/* TransUnit.h */
#ifndef _TRANSUNIT_H_
#define _TRANSUNIT_H_

#include "Types.h"
#include "FileMgr.h"

typedef struct TransUnit TransUnit;

PUBLIC TransUnit * TransUnit_new(FileDesc * file, FileMgr * fileMgr);
PUBLIC void TransUnit_delete(TransUnit * this);
PUBLIC void TransUnit_print(TransUnit* this);
PUBLIC unsigned int TransUnit_getSize(TransUnit* this);
PUBLIC char* TransUnit_getName(TransUnit* this);
PUBLIC String * TransUnit_getNextBuffer(TransUnit* this);

#endif /* _CONFIGURATION_H_ */
```

4.132 TransUnit.h

```
/* TransUnit.h */
#ifndef _TRANSUNIT_H_
#define _TRANSUNIT_H_

#include "Types.h"
#include "FileMgr.h"

typedef struct TransUnit TransUnit;

PUBLIC TransUnit * TransUnit_new(FileDesc * file, FileMgr * fileMgr);
PUBLIC void TransUnit_delete(TransUnit * this);
PUBLIC void TransUnit_print(TransUnit* this);
PUBLIC unsigned int TransUnit_getSize(TransUnit* this);
PUBLIC char* TransUnit_getName(TransUnit* this);
PUBLIC String * TransUnit_getNextBuffer(TransUnit* this);

#endif /* _CONFIGURATION_H_ */
```

Index