Obelisk

1.0.2

Generated by Doxygen 1.9.1

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Namespace Documentation

4.1 obelisk Namespace Reference

The obelisk namespace contains everything needed to compile obelisk code.

Classes

- class CallExpressionAST

  The call AST expression node used to call functions.
- class ExpressionAST

  A generic AST expression which other expression will inherit from.
- class FunctionAST

  A Funcion AST node.
- class NumberExpressionAST

  A number expression AST node.
- class PrototypeAST

  The prototype AST node.
- class VariableExpressionAST

  The variable expression AST node.
- class Lexer

  The Lexer reads and identifies tokens in the obelisk source code.
- class LexerException

  Lexer exception class.
- class KnowledgeBase

  The KnowledgeBase class represents a collection of facts, rules, actions, and related language connectors.
- class KnowledgeBaseException

  Exception thrown by the KnowledgeBase.
- class Action

  The Action model represents an action to take when a fact is true or false.
- class Entity

  The Entity model represents either a left or right side entity, typically used in facts and rules.
- class Fact

  The Fact model represents truth in the releationship between two entities separated by a verb.
- class Rule

  The Rule model represents a truth relation between 2 Facts.
- class SuggestAction

  The SuggestAction model representas the actions to take depending on if the Fact is true or false.
- class Verb

  The Verb model represents a verb which is used to connnect entities.
- class Obelisk

  The obelisk library provides everything needed to consult the KnowledgeBase.
- class DatabaseException

  Exception thrown by database models.
- class DatabaseSizeException

  Exception thrown if the string or blob size exceeds sqlite's limits.
- class DatabaseRangeException

  Exception thrown if the index used it out of range.

- class DatabaseMemoryException

  Exception thrown if there is not enough memory to perform the operation.
- class DatabaseBusyException

  Exception thrown if the database was busy.
- class DatabaseMisuseException

  Exception thrown if there is a misuse of the databse.
- class DatabaseConstraintException

  Exception thrown if a constraint was violated.
- class Parser

  The Parser is responsible for analyzing the language's key words and taking action based on its analysis.
- class ParserException

  The exceptions thrown by the Parser.

Functions

- std::unique_ptr< ExpressionAST > LogError (const char *str)

  Log an AST expression error.
- llvm::Value * LogErrorV (const char *str)

  Log an AST value error.
- static void showUsage ()

  Prints out the usage of obelisk to the stdin.
- int mainLoop (const std::vector< std::string > &sourceFiles, const std::string &kbFile)

  This is the main loop for obelisk.

Variables

- static std::unique_ptr< llvm::LLVMContext > TheContext

  The LLVM context.
- static std::unique_ptr< llvm::Module > TheModule

  The LLVM module.
- static std::unique_ptr< llvm::IRBuilder<> > Builder

  The LLVM IR builder.
- static std::map< std::string, llvm::Value * > NamedValues

  The LLVM named values.
- std::string usageMessage

  The usage messsage displayed during help or incorrect usage.
- static struct option long_options [ ]

  The command line arguments that obelisk accepts.

4.1.1 Detailed Description

The obelisk namespace contains everything needed to compile obelisk code.

4.1.2 Function Documentation

4.1.2.1 LogError()

```
std::unique_ptr< obelisk::ExpressionAST > obelisk::LogError (
            const char * str )
```

Log an AST expression error.

Parameters

in	*str*	The error message.

Returns

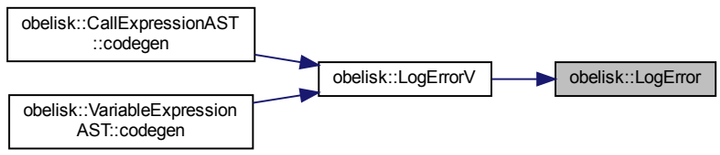
std::unique_ptr<ExpressionAST> Returns the AST expression that caused the error.

Definition at line 3 of file error.cpp.

```
00004 {
00005     fprintf(stderr, "Error: %s\n", Str);
00006     return nullptr;
00007 }
```

Referenced by LogErrorV().

Here is the caller graph for this function:



4.1.2.2 LogErrorV()

```
llvm::Value * obelisk::LogErrorV (
            const char * str )
```

Log an AST value error.

Parameters

in	*str*	The error message.

Returns

llvm::Value* Returns the AST value that caused the error.

Definition at line 9 of file error.cpp.

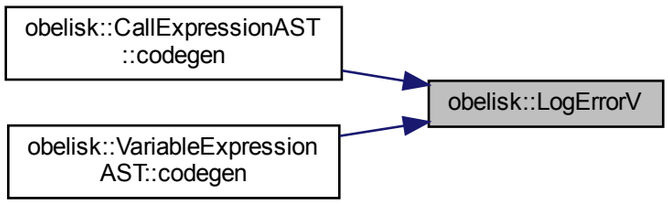
```
00010 {
00011     LogError(Str);
00012     return nullptr;
00013 }
```

References LogError().

Referenced by obelisk::CallExpressionAST::codegen(), and obelisk::VariableExpressionAST::codegen().

Here is the call graph for this function:

Here is the caller graph for this function:



4.1.2.3 mainLoop()

```
int obelisk::mainLoop (
            const std::vector< std::string > & sourceFiles,
            const std::string & kbFile )
```

This is the main loop for obelisk.

This loop handles lexing and parsing of obelisk source code.

Returns

int Returns EXIT_SUCCESS or EXIT_FAILURE.

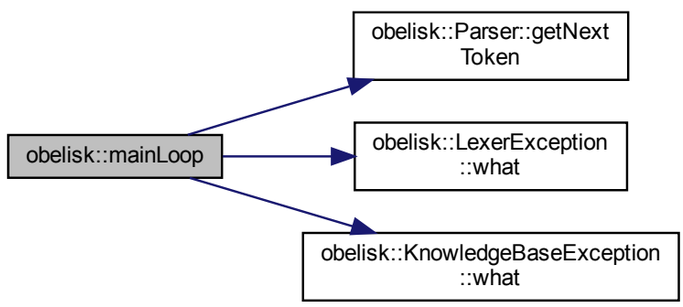
Definition at line 13 of file main.cpp.

```
00015 {
00016     std::unique_ptr<obelisk::KnowledgeBase> kb;
00017
00018     try
00019     {
00020         kb = std::unique_ptr<obelisk::KnowledgeBase> {
00021             new obelisk::KnowledgeBase(kbFile.c_str())};
00022     }
00023     catch (obelisk::KnowledgeBaseException& exception)
00024     {
00025         std::cout « exception.what() « std::endl;
00026         return EXIT_FAILURE;
00027     }
00028
00029     size_t file = 0;
00030     std::shared_ptr<obelisk::Lexer> lexer;
00031     try
00032     {
00033         lexer = std::shared_ptr<obelisk::Lexer> {
00034             new obelisk::Lexer(sourceFiles[file++])};
00035     }
00036     catch (obelisk::LexerException& exception)
00037     {
00038         std::cout « exception.what() « std::endl;
00039         return EXIT_FAILURE;
00040     }
00041     auto parser = std::unique_ptr<obelisk::Parser> {new obelisk::Parser(lexer)};
00042
00043     // prime the first token
00044     try
00045     {
00046         parser->getNextToken();
00047     }
00048     catch (obelisk::LexerException& exception)
00049     {
00050         std::cout « "Error: " « exception.what() « std::endl;
00051         return EXIT_FAILURE;
00052     }
00053
00054     while (true)
00055     {
00056         switch (parser->getCurrentToken())
00057         {
00058             case obelisk::Lexer::kTokenEof :
00059                 // end of source file found, create a new lexer and pass it to
00060                 // the parser to use
00061                 if (file >= sourceFiles.size())
00062                 {
00063                     return EXIT_SUCCESS;
00064                 }
00065                 try
00066                 {
```

```
00067                     lexer = std::shared_ptr<obelisk::Lexer> {
00068                         new obelisk::Lexer(sourceFiles[file++])};
00069                     parser->setLexer(lexer);
00070                     // prime the first token in the parser
00071                     parser->getNextToken();
00072                 }
00073                 catch (obelisk::LexerException& exception)
00074                 {
00075                     std::cout << exception.what() << std::endl;
00076                     return EXIT_FAILURE;
00077                 }
00078                 break;
00079             case ';' :
00080                 // semicolon found, the end of a statement
00081                 try
00082                 {
00083                     parser->getNextToken();
00084                 }
00085                 catch (obelisk::LexerException& exception)
00086                 {
00087                     std::cout << "Error: " << exception.what() << std::endl;
00088                     return EXIT_FAILURE;
00089                 }
00090                 break;
00091             case obelisk::Lexer::kTokenFact :
00092                 try
00093                 {
00094                     parser->handleFact(kb);
00095                 }
00096                 catch (obelisk::ParserException& exception)
00097                 {
00098                     std::cout << "Error: " << exception.what() << std::endl;
00099                     return EXIT_FAILURE;
00100                 }
00101                 break;
00102             case obelisk::Lexer::kTokenRule :
00103                 try
00104                 {
00105                     parser->handleRule(kb);
00106                 }
00107                 catch (obelisk::ParserException& exception)
00108                 {
00109                     std::cout << "Error: " << exception.what() << std::endl;
00110                     return EXIT_FAILURE;
00111                 }
00112                 break;
00113             case obelisk::Lexer::kTokenAction :
00114                 try
00115                 {
00116                     parser->handleAction(kb);
00117                 }
00118                 catch (obelisk::ParserException& exception)
00119                 {
00120                     std::cout << "Error: " << exception.what() << std::endl;
00121                     return EXIT_FAILURE;
00122                 }
00123                 break;
00124             default :
00125                 parser->getNextToken();
00126                 break;
00127         }
00128     }
00129
00130     return EXIT_SUCCESS;
00131 }
```

References obelisk::Parser::getNextToken(), obelisk::Lexer::kTokenAction, obelisk::Lexer::kTokenEof, obelisk::Lexer::kTokenFact, obelisk::Lexer::kTokenRule, obelisk::LexerException::what(), and obelisk::KnowledgeBaseException::what().

Here is the call graph for this function:



4.1.3 Variable Documentation

4.1.3.1 long_options

```
struct option obelisk::long_options[]  [static]
```

Initial value:

```
= {
        {"help",    no_argument,       0, 'h'},
        {"kb",      required_argument, 0, 'k'},
        {"version", no_argument,       0, 'v'},
        {0,         0,                 0, 0  }
    }
```

The command line arguments that obelisk accepts.

Definition at line 17 of file main.h.

4.1.3.2 usageMessage

```
std::string obelisk::usageMessage
```

Initial value:

```
= R"(Usage: obelisk [OPTION]... [FILE]...
Compile the obelisk source FILE(s) into knowledge base and library.

Options:
  -h, --help            shows this help/usage message
  -k, --kb=FILENAME     output knowldege base filename
  -v, --version         shows the version of obelisk)"
```

The usage messsage displayed during help or incorrect usage.

Definition at line 17 of file main.h.

Referenced by showUsage().

Chapter 5

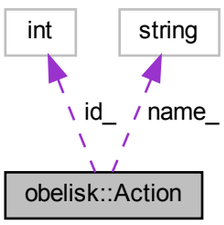
Class Documentation

5.1 obelisk::Action Class Reference

The Action model represents an action to take when a fact is true or false.

`#include <action.h>`

Collaboration diagram for obelisk::Action:



Public Member Functions

- Action ()
  Construct a new Action object.
- Action (int id)
  Construct a new Action object.
- Action (std::string name)
  Construct a new Action object.
- Action (int id, std::string name)
  Construct a new Action object.
- int & getId ()
  Get the ID of the Action.
- void setId (int id)
  Set the ID of the Action.
- std::string & getName ()
  Get the name of the Action.
- void setName (std::string name)
  Set the name of the Action.
- void selectByName (sqlite3 *dbConnection)
  Select an Action from the datbase based on the object name.
- void insert (sqlite3 *dbConnection)
  Insert an Action into the KnowledgeBase based on the object's fields.

Static Public Member Functions

- static const char * createTable ()

  Create the Action table in the KnowledgeBase.

Private Attributes

- int id_

  The ID of the Action in the KnowledgeBase.
- std::string name_

  The name of the Action.

5.1.1 Detailed Description

The Action model represents an action to take when a fact is true or false.

Definition at line 15 of file action.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Action() [1/3]

```
obelisk::Action::Action (
            int id )  [inline]
```

Construct a new Action object.

Parameters

in	*id*	The ID of the Action.

Definition at line 46 of file action.h.

```
00046                           :
00047                 id_(id),
00048                 name_("")
00049             {
00050             }
```

5.1.2.2 Action() [2/3]

```
obelisk::Action::Action (
            std::string name )  [inline]
```

Construct a new Action object.

Parameters

in	*name*	The name of the Action.

Definition at line 57 of file action.h.

```
00057                                 :
00058                 id_(0),
00059                 name_(name)
00060             {
00061             }
```

5.1.2.3 Action() [3/3]

```
obelisk::Action::Action (
            int id,
            std::string name )  [inline]
```

Construct a new Action object.

Parameters

in	*id*	The ID of the Action.
in	*name*	The name of the Action.

Definition at line 69 of file action.h.

```
00069                                                   :
00070                 id_(id),
00071                 name_(name)
00072             {
00073             }
```

5.1.3 Member Function Documentation

5.1.3.1 createTable()

```
const char * obelisk::Action::createTable ( )  [static]
```

Create the Action table in the KnowledgeBase.

Returns

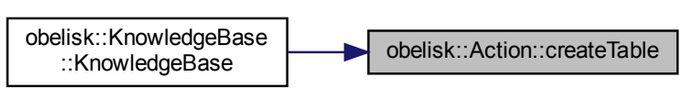
const char* Returns the query used to create the table.

Definition at line 4 of file action.cpp.

```
00005 {
00006     return R"(
00007         CREATE TABLE "action" (
00008             "id"   INTEGER NOT NULL UNIQUE,
00009             "name" TEXT NOT NULL CHECK(trim(name) != ”) UNIQUE,
00010             PRIMARY KEY("id" AUTOINCREMENT)
00011         );
00012     )";
00013 }
```

Referenced by obelisk::KnowledgeBase::KnowledgeBase().

Here is the caller graph for this function:



5.1.3.2 getId()

```
int & obelisk::Action::getId ( )
```

Get the ID of the Action.

Returns

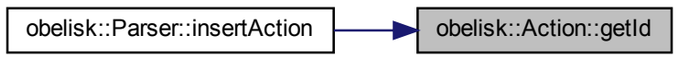
int& Returns the ID.

Definition at line 150 of file action.cpp.

```
00151 {
00152     return id_;
00153 }
```

Referenced by obelisk::Parser::insertAction().

Here is the caller graph for this function:



5.1.3.3 getName()

```
std::string & obelisk::Action::getName ( )
```

Get the name of the Action.

Returns

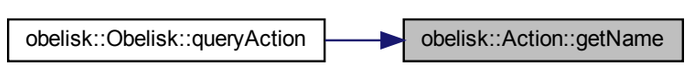
std::string& The Action name.

Definition at line 160 of file action.cpp.

```
00161 {
00162     return name_;
00163 }
```

Referenced by obelisk::Obelisk::queryAction().

Here is the caller graph for this function:



5.1.3.4 insert()

```
void obelisk::Action::insert (
            sqlite3 * dbConnection )
```

Insert an Action into the KnowledgeBase based on the object's fields.

Parameters

in	*dbConnection*	The database connection to use.

Definition at line 83 of file action.cpp.

```
00084 {
00085     if (dbConnection == nullptr)
00086     {
00087         throw obelisk::DatabaseException("database isn't open");
00088     }
00089
00090     sqlite3_stmt* ppStmt = nullptr;
00091
00092     auto result = sqlite3_prepare_v2(dbConnection,
00093         "INSERT INTO action (name) VALUES (?)",
00094         -1,
00095         &ppStmt,
00096         nullptr);
00097     if (result != SQLITE_OK)
00098     {
00099         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00100     }
00101
00102     result
00103         = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_TRANSIENT);
00104     switch (result)
00105     {
00106         case SQLITE_OK :
00107             break;
00108         case SQLITE_TOOBIG :
00109             throw obelisk::DatabaseSizeException();
00110             break;
00111         case SQLITE_RANGE :
00112             throw obelisk::DatabaseRangeException();
00113             break;
00114         case SQLITE_NOMEM :
00115             throw obelisk::DatabaseMemoryException();
00116             break;
00117         default :
00118             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00119             break;
00120     }
00121
00122     result = sqlite3_step(ppStmt);
00123     switch (result)
00124     {
00125         case SQLITE_DONE :
00126             setId((int) sqlite3_last_insert_rowid(dbConnection));
00127             sqlite3_set_last_insert_rowid(dbConnection, 0);
00128             break;
00129         case SQLITE_CONSTRAINT :
00130             throw obelisk::DatabaseConstraintException(
00131                 sqlite3_errmsg(dbConnection));
00132         case SQLITE_BUSY :
00133             throw obelisk::DatabaseBusyException();
00134             break;
00135         case SQLITE_MISUSE :
00136             throw obelisk::DatabaseMisuseException();
00137             break;
00138         default :
00139             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00140             break;
00141     }
00142
00143     result = sqlite3_finalize(ppStmt);
00144     if (result != SQLITE_OK)
00145     {
00146         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00147     }
00148 }
```

5.1.3.5 selectByName()

```
void obelisk::Action::selectByName (
            sqlite3 * dbConnection )
```

Select an Action from the datbase based on the object name.

Parameters

in	*dbConnection*	The database connection to use.

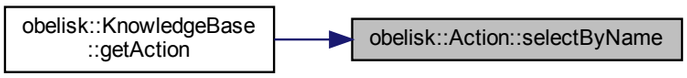
Definition at line 15 of file action.cpp.

```
00016 {
00017     if (dbConnection == nullptr)
00018     {
00019         throw obelisk::DatabaseException("database isn't open");
00020     }
00021
00022     sqlite3_stmt* ppStmt = nullptr;
```

```
00023
00024     auto result = sqlite3_prepare_v2(dbConnection,
00025         "SELECT id, name FROM action WHERE name=?",
00026         -1,
00027         &ppStmt,
00028         nullptr);
00029
00030     if (result != SQLITE_OK)
00031     {
00032         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00033     }
00034
00035     result = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_STATIC);
00036     switch (result)
00037     {
00038         case SQLITE_OK :
00039             break;
00040         case SQLITE_TOOBIG :
00041             throw obelisk::DatabaseSizeException();
00042             break;
00043         case SQLITE_RANGE :
00044             throw obelisk::DatabaseRangeException();
00045             break;
00046         case SQLITE_NOMEM :
00047             throw obelisk::DatabaseMemoryException();
00048             break;
00049         default :
00050             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00051             break;
00052     }
00053
00054     result = sqlite3_step(ppStmt);
00055     switch (result)
00056     {
00057         case SQLITE_DONE :
00058             // no rows in the database
00059             break;
00060         case SQLITE_ROW :
00061             setId(sqlite3_column_int(ppStmt, 0));
00062             setName((char*) sqlite3_column_text(ppStmt, 1));
00063             break;
00064         case SQLITE_BUSY :
00065             throw obelisk::DatabaseBusyException();
00066             break;
00067         case SQLITE_MISUSE :
00068             throw obelisk::DatabaseMisuseException();
00069             break;
00070         default :
00071             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00072             break;
00073     }
00074
00075     result = sqlite3_finalize(ppStmt);
00076
00077     if (result != SQLITE_OK)
00078     {
00079         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00080     }
00081 }
```

Referenced by obelisk::KnowledgeBase::getAction().

Here is the caller graph for this function:



5.1.3.6 setId()

```
void obelisk::Action::setId (
            int id )
```

Set the ID of the Action.

Parameters

in	*id*	Set the ID of the Action.

Definition at line 155 of file action.cpp.

```
00156 {
00157     id_ = id;
00158 }
```

5.1.3.7 setName()

```
void obelisk::Action::setName (
            std::string name )
```

Set the name of the Action.

Parameters

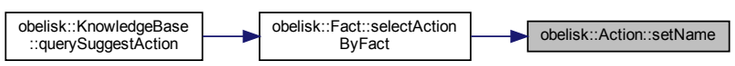
in	*name*	The name of the Action.

Definition at line 165 of file action.cpp.

```
00166 {
00167     name_ = name;
00168 }
```

Referenced by obelisk::Fact::selectActionByFact().

Here is the caller graph for this function:



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

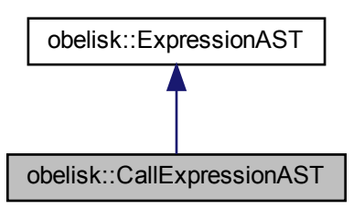
- src/lib/include/models/action.h
- src/lib/models/action.cpp

5.2 obelisk::CallExpressionAST Class Reference

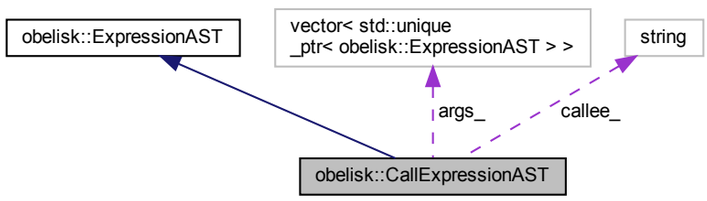
The call AST expression node used to call functions.

`#include <call_expression_ast.h>`

Inheritance diagram for obelisk::CallExpressionAST:



Collaboration diagram for obelisk::CallExpressionAST:



Public Member Functions

- CallExpressionAST (const std::string &callee, std::vector< std::unique_ptr< ExpressionAST >> args)
  Construct a new CallExpressionAST object.
- llvm::Value * codegen () override
  Generate the calle IR code.

Private Member Functions

- std::string getCallee ()
  Get the callee.
- void setCallee (std::string callee)
  Set the callee.
- std::vector< std::unique_ptr< ExpressionAST > > getArgs ()
  Get the arguments being used by the function.
- void setArgs (std::vector< std::unique_ptr< ExpressionAST >> args)
  Set the arguments to be used by the function.

Private Attributes

- std::string callee_
  The function being called.
- std::vector< std::unique_ptr< ExpressionAST > > args_
  The arguments passed to the function.

5.2.1 Detailed Description

The call AST expression node used to call functions.

Definition at line 16 of file call_expression_ast.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 CallExpressionAST()

```
obelisk::CallExpressionAST::CallExpressionAST (
            const std::string & callee,
            std::vector< std::unique_ptr< ExpressionAST >> args )  [inline]
```

Construct a new CallExpressionAST object.

Parameters

in	*callee*	The function to call.
in	*args*	The args to pass into the function.

Definition at line 68 of file call_expression_ast.h.

```
00069                                                                    :
00070                 callee_(callee),
00071                 args_(std::move(args))
00072             {
00073             }
```

5.2.3 Member Function Documentation

5.2.3.1 codegen()

`llvm::Value * obelisk::CallExpressionAST::codegen ( )  [override], [virtual]`

Generate the calle IR code.

Returns

llvm::Value*

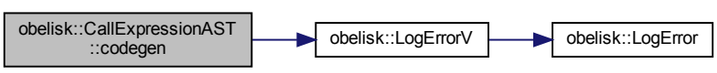
Implements obelisk::ExpressionAST.

Definition at line 5 of file call_expression_ast.cpp.

```
00006 {
00007     // Look up the name in the global module table.
00008     llvm::Function *calleeF = TheModule->getFunction(callee_);
00009     if (!calleeF)
00010     {
00011         return LogErrorV("Unknown function referenced");
00012     }
00013
00014     // If argument mismatch error.
00015     if (calleeF->arg_size() != args_.size())
00016     {
00017         return LogErrorV("Incorrect # arguments passed");
00018     }
00019
00020     std::vector<llvm::Value *> argsV;
00021     for (unsigned i = 0, e = args_.size(); i != e; ++i)
00022     {
00023         argsV.push_back(args_[i]->codegen());
00024         if (!argsV.back())
00025         {
00026             return nullptr;
00027         }
00028     }
00029
00030     return Builder->CreateCall(calleeF, argsV, "calltmp");
00031 }
```

References args_, obelisk::Builder, callee_, obelisk::LogErrorV(), and obelisk::TheModule.

Here is the call graph for this function:



5.2.3.2 getArgs()

```
std::vector<std::unique_ptr<ExpressionAST> > obelisk::CallExpressionAST::getArgs ( )  [private]
```

Get the arguments being used by the function.

Returns

std::vector<std::unique_ptr<ExpressionAST>> Returns an AST expression containing the args.

5.2.3.3 getCallee()

```
std::string obelisk::CallExpressionAST::getCallee ( )  [private]
```

Get the callee.

Returns

std::string Returns the name of the function being called.

5.2.3.4 setArgs()

```
void obelisk::CallExpressionAST::setArgs (
            std::vector< std::unique_ptr< ExpressionAST >> args )  [private]
```

Set the arguments to be used by the function.

Parameters

in	*args*	The args to set.

5.2.3.5 setCallee()

```
void obelisk::CallExpressionAST::setCallee (
            std::string callee )  [private]
```

Set the callee.

Parameters

in	*callee*	The name of the function.

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

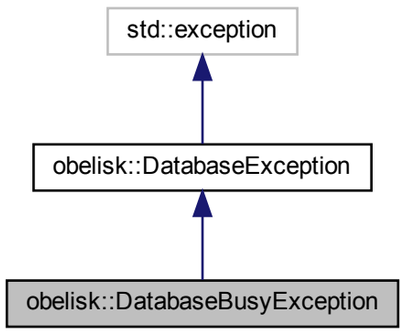
- src/ast/call_expression_ast.h
- src/ast/call_expression_ast.cpp

5.3 obelisk::DatabaseBusyException Class Reference

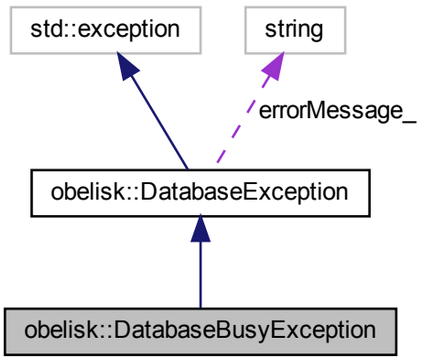
Exception thrown if the database was busy.

```
#include <error.h>
```

Inheritance diagram for obelisk::DatabaseBusyException:



Collaboration diagram for obelisk::DatabaseBusyException:



Public Member Functions

- DatabaseBusyException ()

  Construct a new DatabaseBusyException object.
- virtual const char * what () const noexcept

  Retreive the exception message as a C type string.
- virtual void setErrorMessage (const std::string errorMessage)

  Set the error message.

Protected Attributes

- std::string errorMessage_

  The error message describing the exception.

5.3.1 Detailed Description

Exception thrown if the database was busy.

Definition at line 132 of file error.h.

5.3.2 Member Function Documentation

5.3.2.1 setErrorMessage()

```
virtual void obelisk::DatabaseException::setErrorMessage (
            const std::string errorMessage )  [inline], [virtual], [inherited]
```

Set the error message.

Parameters

in	*errorMessage*	The error message.

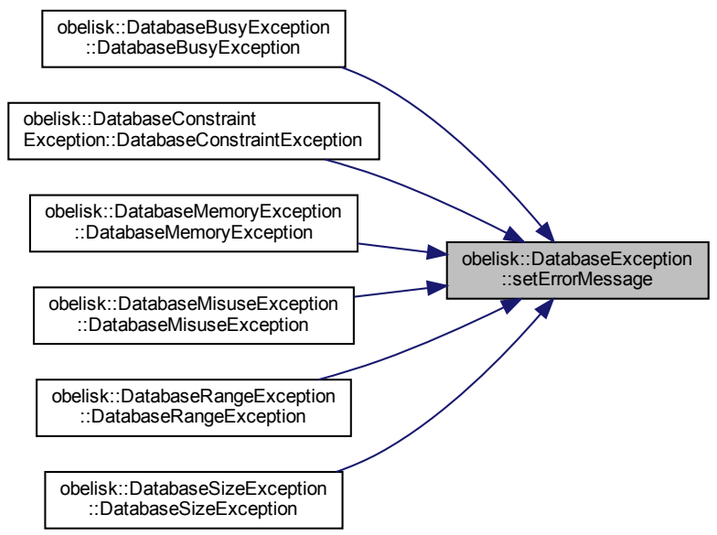
Definition at line 69 of file error.h.

```
00070             {
00071                 errorMessage_ = errorMessage;
00072             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by DatabaseBusyException(), obelisk::DatabaseConstraintException::DatabaseConstraintException(), obelisk::DatabaseMemoryException::DatabaseMemoryException(), obelisk::DatabaseMisuseException::DatabaseMisuseException(), obelisk::DatabaseRangeException::DatabaseRangeException(), and obelisk::DatabaseSizeException::DatabaseSizeException().

Here is the caller graph for this function:



5.3.2.2 what()

```
virtual const char* obelisk::DatabaseException::what ( ) const  [inline], [virtual], [noexcept],
[inherited]
```

Retreive the exception message as a C type string.

Returns

const char* The error message.

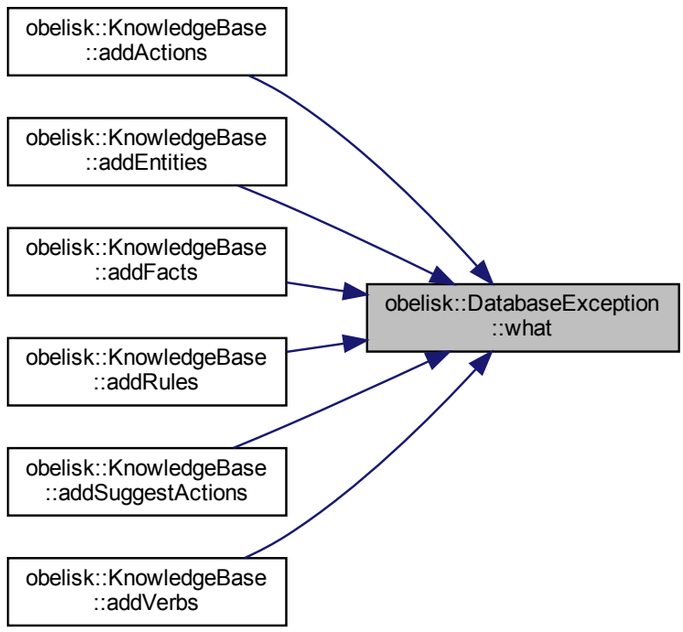
Definition at line 59 of file error.h.

```
00060             {
00061                 return errorMessage_.c_str();
00062             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by obelisk::KnowledgeBase::addActions(), obelisk::KnowledgeBase::addEntities(), obelisk::KnowledgeBase::addFacts(), obelisk::KnowledgeBase::addRules(), obelisk::KnowledgeBase::addSuggestActions(), and obelisk::KnowledgeBase::addVerbs().

Here is the caller graph for this function:



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

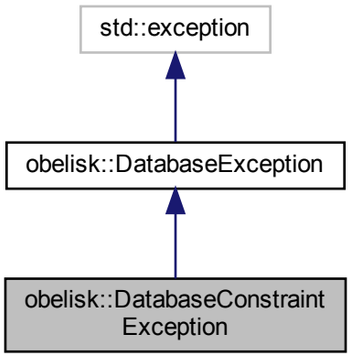
- src/lib/models/error.h

5.4 obelisk::DatabaseConstraintException Class Reference

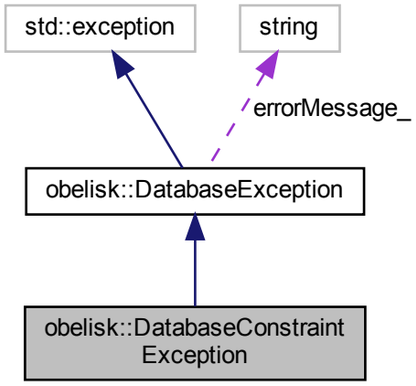
Exception thrown if a constraint was violated.

`#include <error.h>`

Inheritance diagram for obelisk::DatabaseConstraintException:



Collaboration diagram for obelisk::DatabaseConstraintException:



Public Member Functions

- DatabaseConstraintException ()
  Construct a new DatabaseConstraintException object.
- DatabaseConstraintException (const std::string &errorMessage)
  Construct a new DatabaseConstraintException object.
- virtual const char * what () const noexcept
  Retreive the exception message as a C type string.
- virtual void setErrorMessage (const std::string errorMessage)
  Set the error message.

Protected Attributes

- std::string errorMessage_
  The error message describing the exception.

5.4.1 Detailed Description

Exception thrown if a constraint was violated.

Definition at line 168 of file error.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 DatabaseConstraintException()

```
obelisk::DatabaseConstraintException::DatabaseConstraintException (
            const std::string & errorMessage )  [inline]
```

Construct a new DatabaseConstraintException object.

Parameters

in	*errorMessage*	The error message to send when the constraint is violated.

Definition at line 186 of file error.h.

```
00187             {
00188                 setErrorMessage(errorMessage);
00189             }
```

References obelisk::DatabaseException::setErrorMessage().

Here is the call graph for this function:

5.4.3 Member Function Documentation

5.4.3.1 setErrorMessage()

```
virtual void obelisk::DatabaseException::setErrorMessage (
            const std::string errorMessage )  [inline], [virtual], [inherited]
```

Set the error message.

Parameters

in	*errorMessage*	The error message.

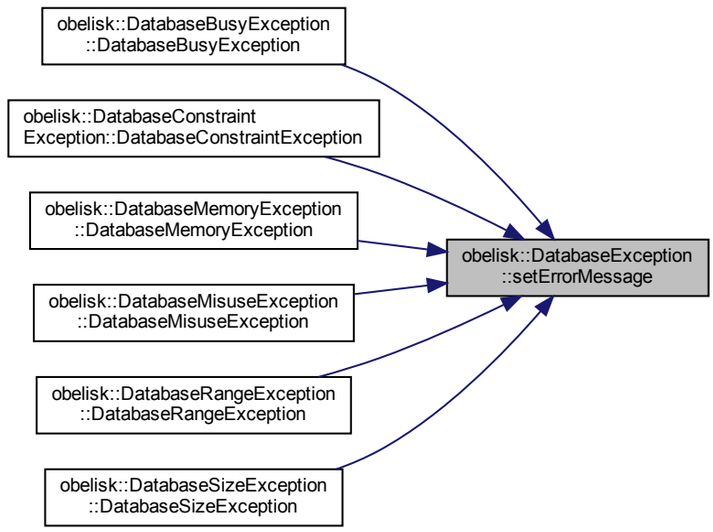
Definition at line 69 of file error.h.

```
00070             {
00071                 errorMessage_ = errorMessage;
00072             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by obelisk::DatabaseBusyException::DatabaseBusyException(), DatabaseConstraintException(), obelisk::DatabaseMemoryException::DatabaseMemoryException(), obelisk::DatabaseMisuseException::DatabaseMisuseException(), obelisk::DatabaseRangeException::DatabaseRangeException(), and obelisk::DatabaseSizeException::DatabaseSizeException().

Here is the caller graph for this function:



5.4.3.2 what()

```
virtual const char* obelisk::DatabaseException::what ( ) const  [inline], [virtual], [noexcept],
[inherited]
```

Retreive the exception message as a C type string.

Returns

const char* The error message.

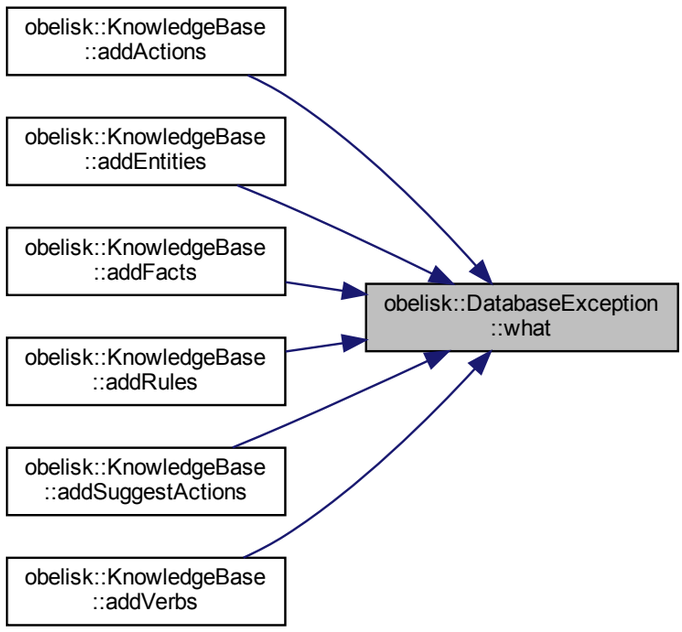
Definition at line 59 of file error.h.

```
00060             {
00061                 return errorMessage_.c_str();
00062             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by obelisk::KnowledgeBase::addActions(), obelisk::KnowledgeBase::addEntities(), obelisk::KnowledgeBase::addFacts(), obelisk::KnowledgeBase::addRules(), obelisk::KnowledgeBase::addSuggestActions(), and obelisk::KnowledgeBase::addVerbs().

Here is the caller graph for this function:



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

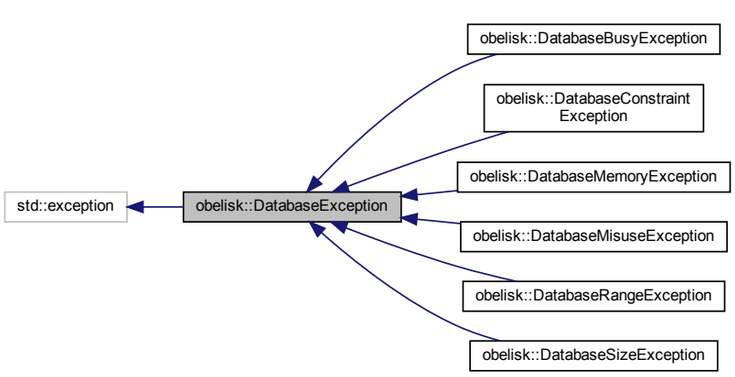
- src/lib/models/error.h

5.5 obelisk::DatabaseException Class Reference

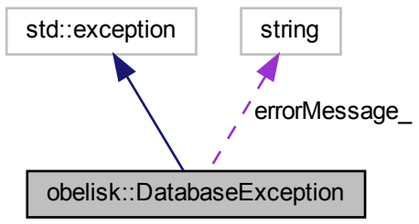
Exception thrown by database models.

`#include <error.h>`

Inheritance diagram for obelisk::DatabaseException:



Collaboration diagram for obelisk::DatabaseException:



Public Member Functions

- DatabaseException ()
  Construct a new DatabaseException object.
- DatabaseException (const int errorCode)
  Construct a new DatabaseException object.
- DatabaseException (const std::string &errorMessage)
  Construct a new DatabaseException object.
- virtual const char * what () const noexcept
  Retreive the exception message as a C type string.
- virtual void setErrorMessage (const std::string errorMessage)
  Set the error message.

Protected Attributes

- std::string errorMessage_
  The error message describing the exception.

5.5.1 Detailed Description

Exception thrown by database models.

Definition at line 13 of file error.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 DatabaseException() [1/2]

```
obelisk::DatabaseException::DatabaseException (
            const int errorCode )  [inline]
```

Construct a new DatabaseException object.

Parameters

in	*errorCode*	The error code that came from sqlite.

Definition at line 37 of file error.h.

```
00037                                                 :
00038                 errorMessage_(
```

```
00039                     "database error " + std::to_string(errorCode) + " ocurred")
00040             {
00041             }
```

5.5.2.2 DatabaseException() [2/2]

```
obelisk::DatabaseException::DatabaseException (
            const std::string & errorMessage )  [inline]
```

Construct a new DatabaseException object.

Parameters

in	*errorMessage*	The error message to describe the exception.

Definition at line 49 of file error.h.

```
00049                                                                    :
00050                 errorMessage_(errorMessage)
00051             {
00052             }
```

5.5.3 Member Function Documentation

5.5.3.1 setErrorMessage()

```
virtual void obelisk::DatabaseException::setErrorMessage (
            const std::string errorMessage )  [inline], [virtual]
```

Set the error message.

Parameters

in	*errorMessage*	The error message.

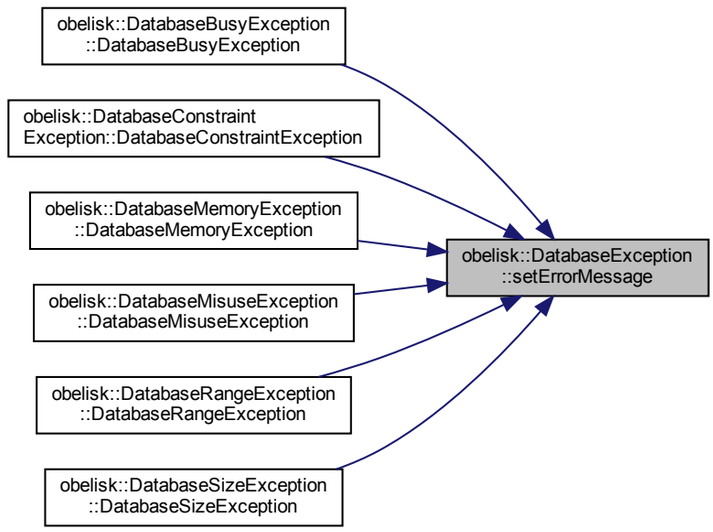
Definition at line 69 of file error.h.

```
00070             {
00071                 errorMessage_ = errorMessage;
00072             }
```

References errorMessage_.

Referenced by obelisk::DatabaseBusyException::DatabaseBusyException(), obelisk::DatabaseConstraintException::DatabaseConstraintE
obelisk::DatabaseMemoryException::DatabaseMemoryException(), obelisk::DatabaseMisuseException::DatabaseMisuseException(),
obelisk::DatabaseRangeException::DatabaseRangeException(), and obelisk::DatabaseSizeException::DatabaseSizeException().

Here is the caller graph for this function:



5.5.3.2 what()

```
virtual const char* obelisk::DatabaseException::what ( ) const  [inline], [virtual], [noexcept]
```

Retreive the exception message as a C type string.

Returns

const char* The error message.

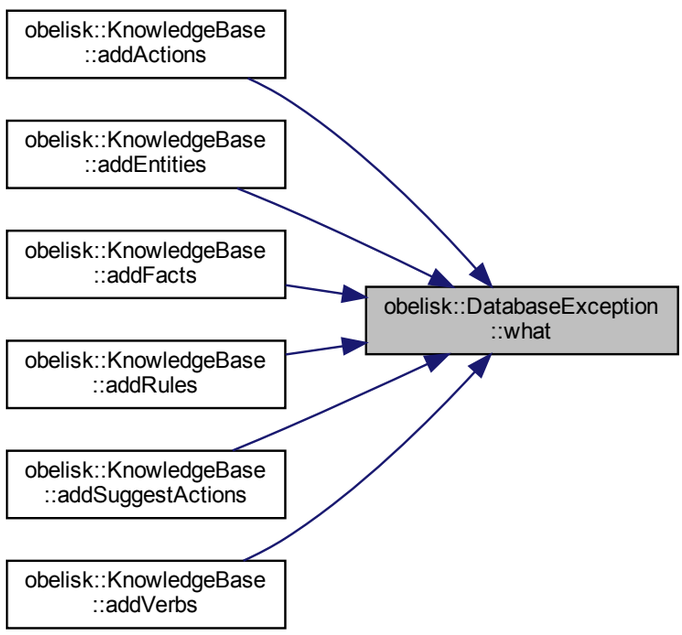
Definition at line 59 of file error.h.

```
00060             {
00061                 return errorMessage_.c_str();
00062             }
```

References errorMessage_.

Referenced by obelisk::KnowledgeBase::addActions(), obelisk::KnowledgeBase::addEntities(), obelisk::KnowledgeBase::addFacts(), obelisk::KnowledgeBase::addRules(), obelisk::KnowledgeBase::addSuggestActions(), and obelisk::KnowledgeBase::addVerbs().

Here is the caller graph for this function:



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

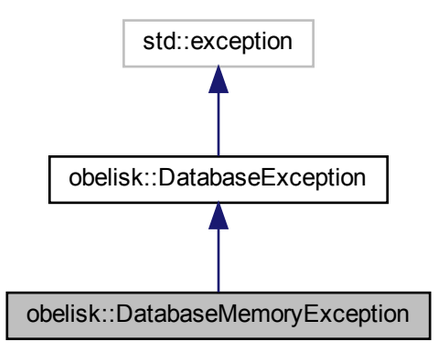
- src/lib/models/error.h

5.6 obelisk::DatabaseMemoryException Class Reference

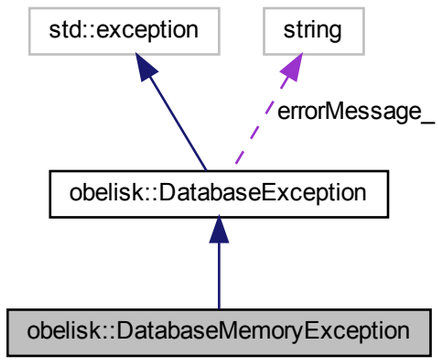
Exception thrown if there is not enough memory to perform the operation.

`#include <error.h>`

Inheritance diagram for obelisk::DatabaseMemoryException:



Collaboration diagram for obelisk::DatabaseMemoryException:



Public Member Functions

- DatabaseMemoryException ()

  Construct a new DatabaseMemoryException object.
- virtual const char * what () const noexcept

  Retreive the exception message as a C type string.
- virtual void setErrorMessage (const std::string errorMessage)

  Set the error message.

Protected Attributes

- std::string errorMessage_

  The error message describing the exception.

5.6.1 Detailed Description

Exception thrown if there is not enough memory to perform the operation.

Definition at line 115 of file error.h.

5.6.2 Member Function Documentation

5.6.2.1 setErrorMessage()

```
virtual void obelisk::DatabaseException::setErrorMessage (
            const std::string errorMessage )  [inline], [virtual], [inherited]
```

Set the error message.

Parameters

in	*errorMessage*	The error message.

Definition at line 69 of file error.h.

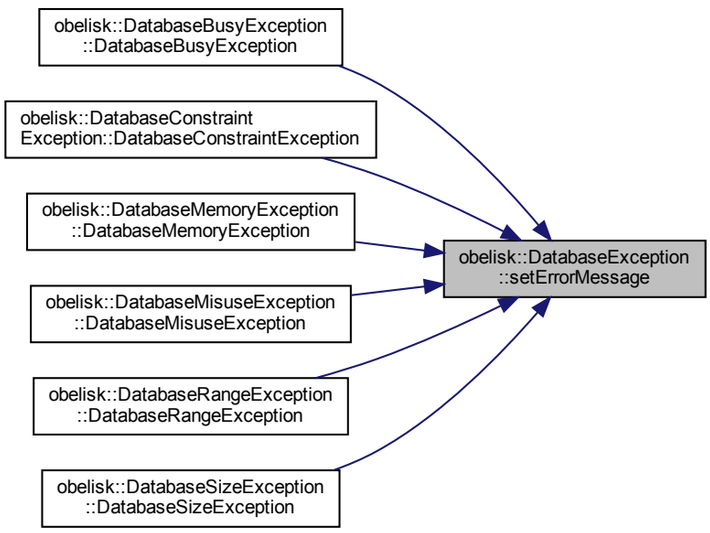
```
00070             {
00071                 errorMessage_ = errorMessage;
```

```
00072             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by obelisk::DatabaseBusyException::DatabaseBusyException(), obelisk::DatabaseConstraintException::DatabaseConstraintE
DatabaseMemoryException(), obelisk::DatabaseMisuseException::DatabaseMisuseException(), obelisk::DatabaseRangeException::Data
and obelisk::DatabaseSizeException::DatabaseSizeException().

Here is the caller graph for this function:



5.6.2.2 what()

```
virtual const char* obelisk::DatabaseException::what ( ) const  [inline], [virtual], [noexcept],
[inherited]
```

Retreive the exception message as a C type string.

Returns

const char* The error message.

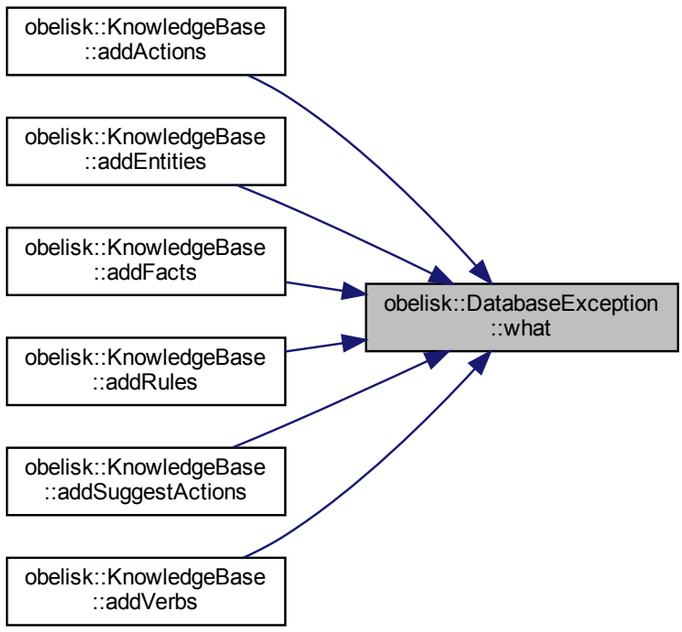
Definition at line 59 of file error.h.

```
00060             {
00061                 return errorMessage_.c_str();
00062             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by obelisk::KnowledgeBase::addActions(), obelisk::KnowledgeBase::addEntities(), obelisk::KnowledgeBase::addFacts(),
obelisk::KnowledgeBase::addRules(), obelisk::KnowledgeBase::addSuggestActions(), and obelisk::KnowledgeBase::addVerbs().

Here is the caller graph for this function:



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

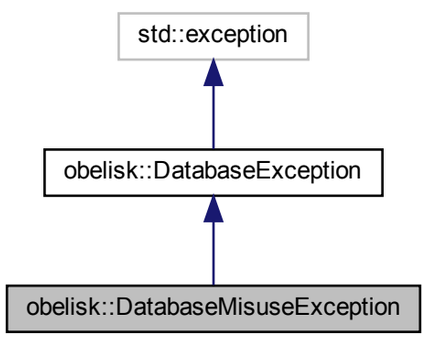
- src/lib/models/error.h

5.7 obelisk::DatabaseMisuseException Class Reference

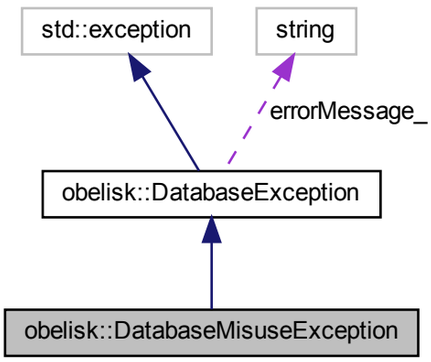
Exception thrown if there is a misuse of the databse.

`#include <error.h>`

Inheritance diagram for obelisk::DatabaseMisuseException:



Collaboration diagram for obelisk::DatabaseMisuseException:



Public Member Functions

- DatabaseMisuseException ()

  Construct a new DatabaseMisuseException object.
- virtual const char * what () const noexcept

  Retreive the exception message as a C type string.
- virtual void setErrorMessage (const std::string errorMessage)

  Set the error message.

Protected Attributes

- std::string errorMessage_

  The error message describing the exception.

5.7.1 Detailed Description

Exception thrown if there is a misuse of the databse.

Definition at line 150 of file error.h.

5.7.2 Member Function Documentation

5.7.2.1 setErrorMessage()

```
virtual void obelisk::DatabaseException::setErrorMessage (
            const std::string errorMessage )  [inline], [virtual], [inherited]
```

Set the error message.

Parameters

in	*errorMessage*	The error message.

Definition at line 69 of file error.h.

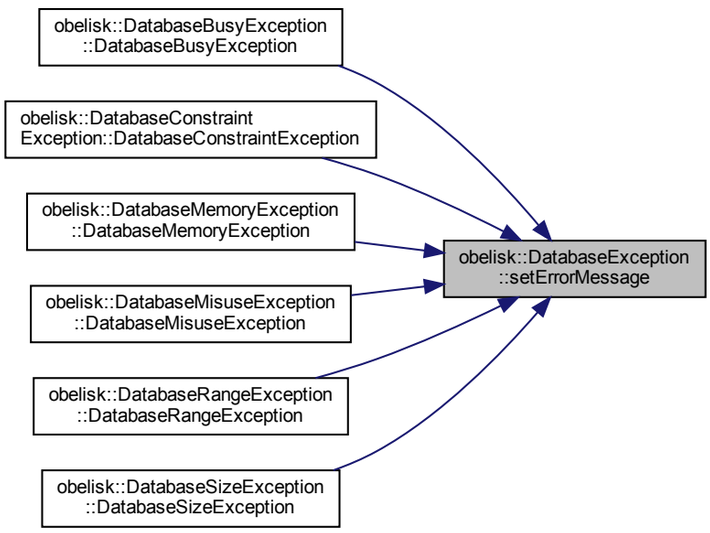
```
00070             {
00071                 errorMessage_ = errorMessage;
```

```
00072             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by obelisk::DatabaseBusyException::DatabaseBusyException(), obelisk::DatabaseConstraintException::DatabaseConstraintE
obelisk::DatabaseMemoryException::DatabaseMemoryException(), DatabaseMisuseException(), obelisk::DatabaseRangeException::Dat
and obelisk::DatabaseSizeException::DatabaseSizeException().

Here is the caller graph for this function:



5.7.2.2 what()

```
virtual const char* obelisk::DatabaseException::what ( ) const  [inline], [virtual], [noexcept],
[inherited]
```

Retreive the exception message as a C type string.

Returns

const char* The error message.

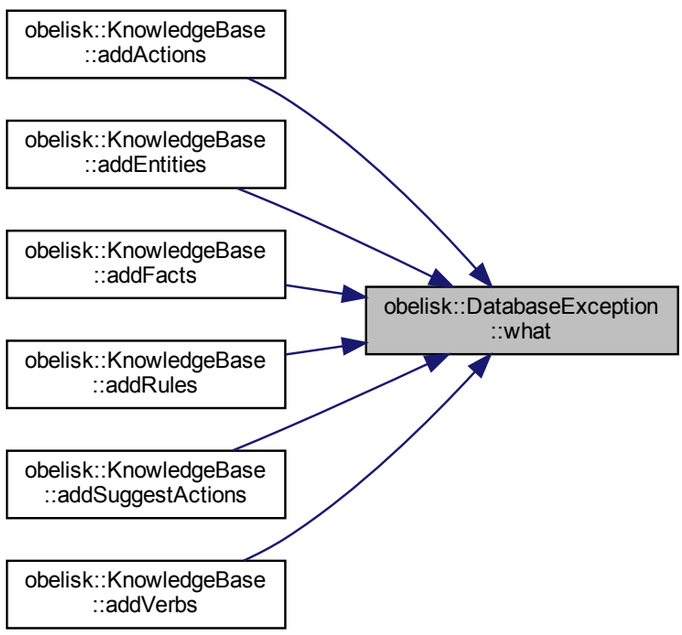
Definition at line 59 of file error.h.

```
00060             {
00061                 return errorMessage_.c_str();
00062             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by obelisk::KnowledgeBase::addActions(), obelisk::KnowledgeBase::addEntities(), obelisk::KnowledgeBase::addFacts(),
obelisk::KnowledgeBase::addRules(), obelisk::KnowledgeBase::addSuggestActions(), and obelisk::KnowledgeBase::addVerbs().

Here is the caller graph for this function:



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

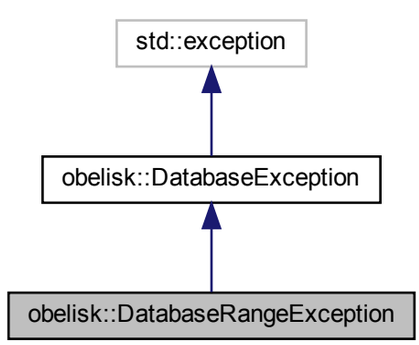
- src/lib/models/error.h

5.8 obelisk::DatabaseRangeException Class Reference

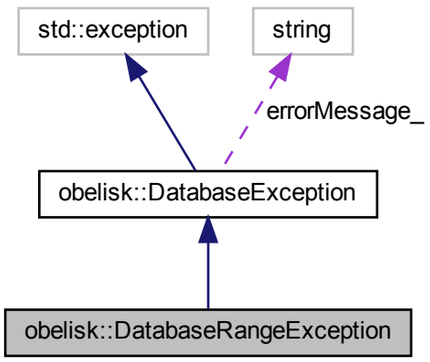
Exception thrown if the index used it out of range.

`#include <error.h>`

Inheritance diagram for obelisk::DatabaseRangeException:



Collaboration diagram for obelisk::DatabaseRangeException:



Public Member Functions

- DatabaseRangeException ()

  Construct a new DatabaseRangeException object.
- virtual const char * what () const noexcept

  Retreive the exception message as a C type string.
- virtual void setErrorMessage (const std::string errorMessage)

  Set the error message.

Protected Attributes

- std::string errorMessage_

  The error message describing the exception.

5.8.1 Detailed Description

Exception thrown if the index used it out of range.

Definition at line 97 of file error.h.

5.8.2 Member Function Documentation

5.8.2.1 setErrorMessage()

```
virtual void obelisk::DatabaseException::setErrorMessage (
            const std::string errorMessage )  [inline], [virtual], [inherited]
```

Set the error message.

Parameters

in	*errorMessage*	The error message.

Definition at line 69 of file error.h.

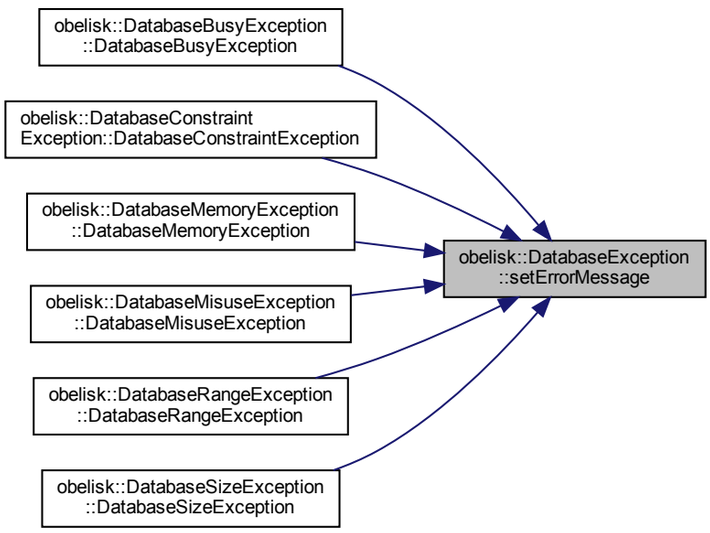
```
00070             {
00071                 errorMessage_ = errorMessage;
```

```
00072             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by obelisk::DatabaseBusyException::DatabaseBusyException(), obelisk::DatabaseConstraintException::DatabaseConstraintE
obelisk::DatabaseMemoryException::DatabaseMemoryException(), obelisk::DatabaseMisuseException::DatabaseMisuseException(),
DatabaseRangeException(), and obelisk::DatabaseSizeException::DatabaseSizeException().

Here is the caller graph for this function:



5.8.2.2 what()

```
virtual const char* obelisk::DatabaseException::what ( ) const  [inline], [virtual], [noexcept],
[inherited]
```

Retreive the exception message as a C type string.

Returns

const char* The error message.

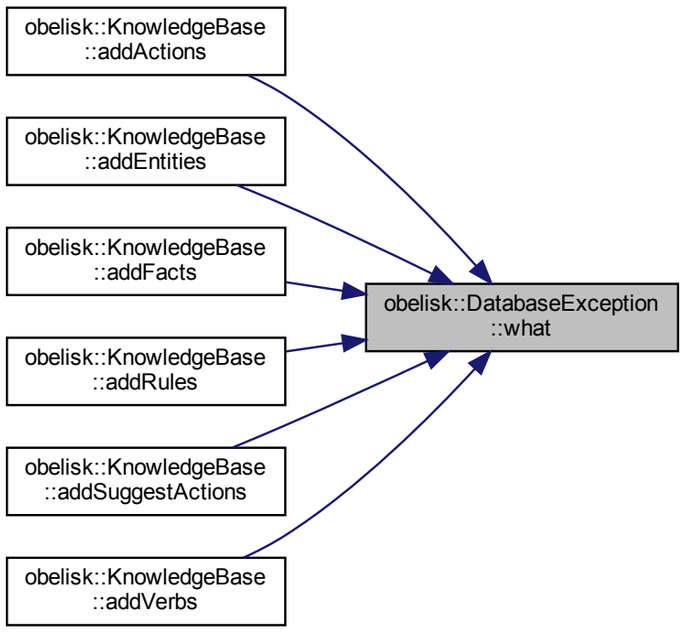
Definition at line 59 of file error.h.

```
00060             {
00061                 return errorMessage_.c_str();
00062             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by obelisk::KnowledgeBase::addActions(), obelisk::KnowledgeBase::addEntities(), obelisk::KnowledgeBase::addFacts(),
obelisk::KnowledgeBase::addRules(), obelisk::KnowledgeBase::addSuggestActions(), and obelisk::KnowledgeBase::addVerbs().

Here is the caller graph for this function:



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

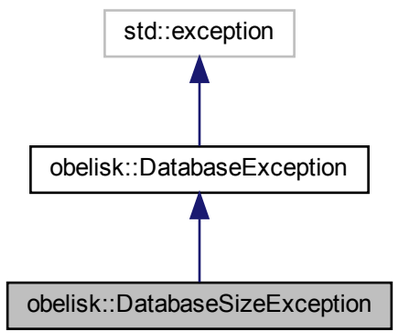
- src/lib/models/error.h

5.9 obelisk::DatabaseSizeException Class Reference

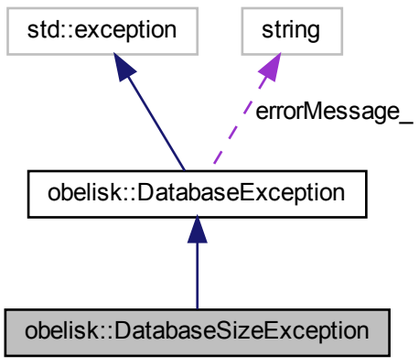
Exception thrown if the string or blob size exceeds sqlite's limits.

`#include <error.h>`

Inheritance diagram for obelisk::DatabaseSizeException:



Collaboration diagram for obelisk::DatabaseSizeException:



Public Member Functions

- DatabaseSizeException ()

  Construct a new DatabaseSizeException object.
- virtual const char * what () const noexcept

  Retreive the exception message as a C type string.
- virtual void setErrorMessage (const std::string errorMessage)

  Set the error message.

Protected Attributes

- std::string errorMessage_

  The error message describing the exception.

5.9.1 Detailed Description

Exception thrown if the string or blob size exceeds sqlite's limits.

Definition at line 80 of file error.h.

5.9.2 Member Function Documentation

5.9.2.1 setErrorMessage()

```
virtual void obelisk::DatabaseException::setErrorMessage (
            const std::string errorMessage )  [inline], [virtual], [inherited]
```

Set the error message.

Parameters

in	*errorMessage*	The error message.

Definition at line 69 of file error.h.

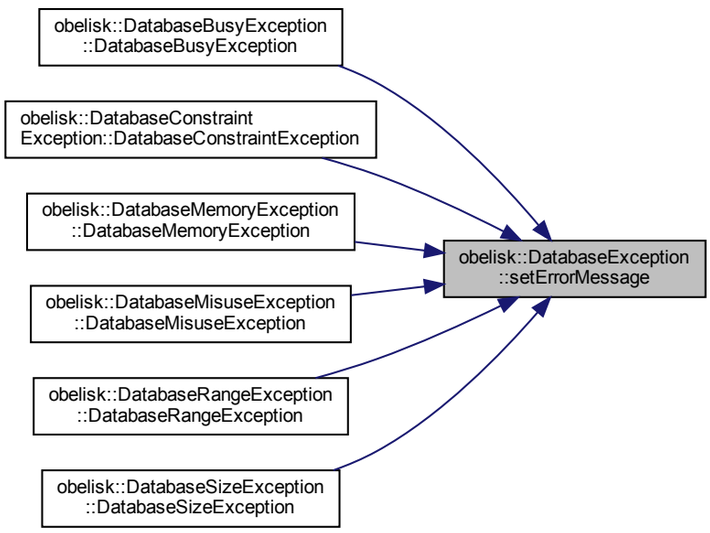
```
00070             {
00071                 errorMessage_ = errorMessage;
```

```
00072             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by obelisk::DatabaseBusyException::DatabaseBusyException(), obelisk::DatabaseConstraintException::DatabaseConstraintE
obelisk::DatabaseMemoryException::DatabaseMemoryException(), obelisk::DatabaseMisuseException::DatabaseMisuseException(),
obelisk::DatabaseRangeException::DatabaseRangeException(), and DatabaseSizeException().

Here is the caller graph for this function:



5.9.2.2 what()

```
virtual const char* obelisk::DatabaseException::what ( ) const  [inline], [virtual], [noexcept],
[inherited]
```

Retreive the exception message as a C type string.

Returns

const char* The error message.

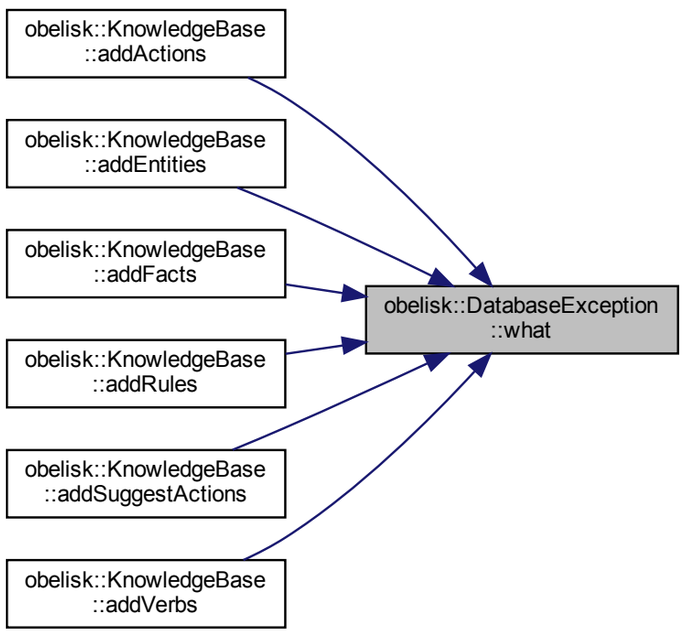
Definition at line 59 of file error.h.

```
00060             {
00061                 return errorMessage_.c_str();
00062             }
```

References obelisk::DatabaseException::errorMessage_.

Referenced by obelisk::KnowledgeBase::addActions(), obelisk::KnowledgeBase::addEntities(), obelisk::KnowledgeBase::addFacts(),
obelisk::KnowledgeBase::addRules(), obelisk::KnowledgeBase::addSuggestActions(), and obelisk::KnowledgeBase::addVerbs().

Here is the caller graph for this function:



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

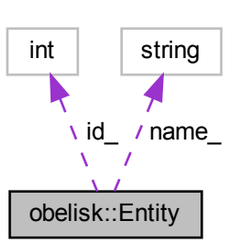
- src/lib/models/error.h

5.10 obelisk::Entity Class Reference

The Entity model represents either a left or right side entity, typically used in facts and rules.

`#include <entity.h>`

Collaboration diagram for obelisk::Entity:



Public Member Functions

- Entity ()

  Construct a new Entity object.
- Entity (int id)

  Construct a new Entity object.
- Entity (std::string name)

Construct a new Entity object.

- Entity (int id, std::string name)

  Construct a new Entity object.
- int & getId ()

  Get the ID of the Entity.
- void setId (int id)

  Set the ID of the Entity.
- std::string & getName ()

  Get the name of the Entity.
- void setName (std::string name)

  Set the name of the Entity.
- void selectByName (sqlite3 *dbConnection)

  Select an Entity from the KnowledgeBase based on the object's name.
- void insert (sqlite3 *dbConnection)

  Insert an Entity into the KnowledgeBase based on the object's fields.

Static Public Member Functions

- static const char * createTable ()

  Create the table in the KnowledgeBase.

Private Attributes

- int id_

  The ID of the Entity in the KnowledgeBase.
- std::string name_

  The name of the Entity.

5.10.1 Detailed Description

The Entity model represents either a left or right side entity, typically used in facts and rules.

Definition at line 15 of file entity.h.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Entity() [1/3]

```
obelisk::Entity::Entity (
            int id )  [inline]
```

Construct a new Entity object.

Parameters

in	*id*	The ID of the Entity.

Definition at line 46 of file entity.h.

```
00046                                 :
00047                 id_(id),
00048                 name_("")
00049             {
00050             }
```

5.10.2.2 Entity() [2/3]

```
obelisk::Entity::Entity (
            std::string name )  [inline]
```

Construct a new Entity object.

Parameters

in	*name*	The name of the Entity.

Definition at line 57 of file entity.h.

```
00057                                       :
00058                 id_(0),
00059                 name_(name)
00060             {
00061             }
```

5.10.2.3 Entity() [3/3]

```
obelisk::Entity::Entity (
            int id,
            std::string name )  [inline]
```

Construct a new Entity object.

Parameters

in	*id*	The ID of the Entity.
in	*name*	The name of the Entity.

Definition at line 69 of file entity.h.

```
00069                                                :
00070                 id_(id),
00071                 name_(name)
00072             {
00073             }
```

5.10.3 Member Function Documentation

5.10.3.1 createTable()

```
const char * obelisk::Entity::createTable ( )  [static]
```

Create the table in the KnowledgeBase.

Returns

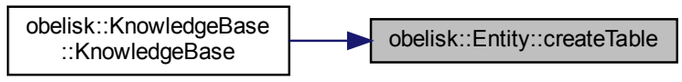
const char* Returns the query used to create the table.

Definition at line 4 of file entity.cpp.

```
00005 {
00006     return R"(
00007         CREATE TABLE "entity" (
00008             "id"   INTEGER NOT NULL UNIQUE,
00009             "name" TEXT NOT NULL CHECK(trim(name) != ”) UNIQUE,
00010             PRIMARY KEY("id" AUTOINCREMENT)
00011         );
00012     )";
00013 }
```

Referenced by obelisk::KnowledgeBase::KnowledgeBase().

Here is the caller graph for this function:



5.10.3.2 getId()

```
int & obelisk::Entity::getId ( )
```

Get the ID of the Entity.

Returns

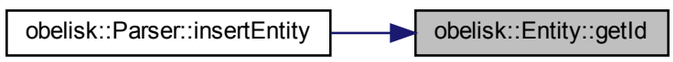
int& Returns the ID.

Definition at line 150 of file entity.cpp.

```
00151 {
00152     return id_;
00153 }
```

Referenced by obelisk::Parser::insertEntity().

Here is the caller graph for this function:



5.10.3.3 getName()

```
std::string & obelisk::Entity::getName ( )
```

Get the name of the Entity.

Returns

std::string& The name of the Entity.

Definition at line 160 of file entity.cpp.

```
00161 {
00162     return name_;
00163 }
```

5.10.3.4 insert()

```
void obelisk::Entity::insert (
            sqlite3 * dbConnection )
```

Insert an Entity into the KnowledgeBase based on the object's fields.

Parameters

in	*dbConnection*	The database connection to use.

Definition at line 83 of file entity.cpp.

```
00084 {
00085     if (dbConnection == nullptr)
00086     {
00087         throw obelisk::DatabaseException("database isn't open");
00088     }
00089
00090     sqlite3_stmt* ppStmt = nullptr;
00091
00092     auto result = sqlite3_prepare_v2(dbConnection,
00093         "INSERT INTO entity (name) VALUES (?)",
00094         -1,
00095         &ppStmt,
00096         nullptr);
00097     if (result != SQLITE_OK)
00098     {
00099         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00100     }
00101
00102     result
00103         = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_TRANSIENT);
00104     switch (result)
00105     {
00106         case SQLITE_OK :
00107             break;
00108         case SQLITE_TOOBIG :
00109             throw obelisk::DatabaseSizeException();
00110             break;
00111         case SQLITE_RANGE :
00112             throw obelisk::DatabaseRangeException();
00113             break;
00114         case SQLITE_NOMEM :
00115             throw obelisk::DatabaseMemoryException();
00116             break;
00117         default :
00118             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00119             break;
00120     }
00121
00122     result = sqlite3_step(ppStmt);
00123     switch (result)
00124     {
00125         case SQLITE_DONE :
00126             setId((int) sqlite3_last_insert_rowid(dbConnection));
00127             sqlite3_set_last_insert_rowid(dbConnection, 0);
00128             break;
00129         case SQLITE_CONSTRAINT :
00130             throw obelisk::DatabaseConstraintException(
00131                 sqlite3_errmsg(dbConnection));
00132         case SQLITE_BUSY :
00133             throw obelisk::DatabaseBusyException();
00134             break;
00135         case SQLITE_MISUSE :
00136             throw obelisk::DatabaseMisuseException();
00137             break;
00138         default :
00139             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00140             break;
00141     }
00142
00143     result = sqlite3_finalize(ppStmt);
00144     if (result != SQLITE_OK)
00145     {
00146         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00147     }
00148 }
```

5.10.3.5 selectByName()

```
void obelisk::Entity::selectByName (
            sqlite3 * dbConnection )
```

Select an Entity from the KnowledgeBase based on the object's name.

Parameters

in	*dbConnection*	The database connection to use.

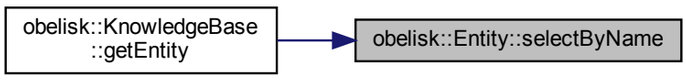
Definition at line 15 of file entity.cpp.

```
00016 {
00017     if (dbConnection == nullptr)
00018     {
00019         throw obelisk::DatabaseException("database isn't open");
00020     }
00021
00022     sqlite3_stmt* ppStmt = nullptr;
```

```
00023
00024     auto result = sqlite3_prepare_v2(dbConnection,
00025         "SELECT id, name FROM entity WHERE name=?",
00026         -1,
00027         &ppStmt,
00028         nullptr);
00029
00030     if (result != SQLITE_OK)
00031     {
00032         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00033     }
00034
00035     result = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_STATIC);
00036     switch (result)
00037     {
00038         case SQLITE_OK :
00039             break;
00040         case SQLITE_TOOBIG :
00041             throw obelisk::DatabaseSizeException();
00042             break;
00043         case SQLITE_RANGE :
00044             throw obelisk::DatabaseRangeException();
00045             break;
00046         case SQLITE_NOMEM :
00047             throw obelisk::DatabaseMemoryException();
00048             break;
00049         default :
00050             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00051             break;
00052     }
00053
00054     result = sqlite3_step(ppStmt);
00055     switch (result)
00056     {
00057         case SQLITE_DONE :
00058             // no rows in the database
00059             break;
00060         case SQLITE_ROW :
00061             setId(sqlite3_column_int(ppStmt, 0));
00062             setName((char*) sqlite3_column_text(ppStmt, 1));
00063             break;
00064         case SQLITE_BUSY :
00065             throw obelisk::DatabaseBusyException();
00066             break;
00067         case SQLITE_MISUSE :
00068             throw obelisk::DatabaseMisuseException();
00069             break;
00070         default :
00071             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00072             break;
00073     }
00074
00075     result = sqlite3_finalize(ppStmt);
00076
00077     if (result != SQLITE_OK)
00078     {
00079         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00080     }
00081 }
```

Referenced by obelisk::KnowledgeBase::getEntity().

Here is the caller graph for this function:



5.10.3.6 setId()

```
void obelisk::Entity::setId (
            int id )
```

Set the ID of the Entity.

Parameters

in	*id*	The ID of the Entity.

Definition at line 155 of file entity.cpp.

```
00156 {
00157     id_ = id;
00158 }
```

5.10.3.7 setName()

```
void obelisk::Entity::setName (
            std::string name )
```

Set the name of the Entity.

Parameters

in	*name*	The name of the Entity.

Definition at line 165 of file entity.cpp.

```
00166 {
00167     name_ = name;
00168 }
```

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

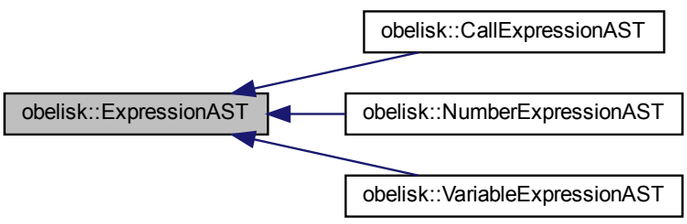
- src/lib/include/models/entity.h
- src/lib/models/entity.cpp

5.11 obelisk::ExpressionAST Class Reference

A generic AST expression which other expression will inherit from.

```
#include <expression_ast.h>
```

Inheritance diagram for obelisk::ExpressionAST:



Public Member Functions

- virtual ~ExpressionAST ()=default

  Destroy the ExpressionAST object.
- virtual llvm::Value * codegen ()=0

  Generate LLVM IR code based on the AST expression.

5.11.1 Detailed Description

A generic AST expression which other expression will inherit from.

Definition at line 12 of file expression_ast.h.

5.11.2 Member Function Documentation

5.11.2.1 codegen()

```
virtual llvm::Value* obelisk::ExpressionAST::codegen ( )  [pure virtual]
```

Generate LLVM IR code based on the AST expression.

Returns

llvm::Value* Returns the LLVM code value from the expression.

Implemented in obelisk::VariableExpressionAST, obelisk::NumberExpressionAST, and obelisk::CallExpressionAST.

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

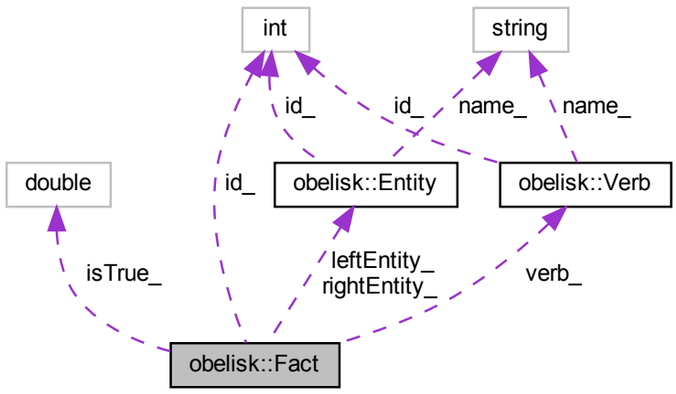
- src/ast/expression_ast.h

5.12 obelisk::Fact Class Reference

The Fact model represents truth in the releationship between two entities separated by a verb.

```
#include <fact.h>
```

Collaboration diagram for obelisk::Fact:



Public Member Functions

- Fact ()
  Construct a new Fact object.
- Fact (int id)
  Construct a new Fact object.
- Fact (obelisk::Entity leftEntity, obelisk::Entity rightEntity, obelisk::Verb verb, double isTrue=0)
  Construct a new Fact object.
- Fact (int id, obelisk::Entity leftEntity, obelisk::Entity rightEntity, obelisk::Verb verb, double isTrue=0)
  Construct a new Fact object.
- int & getId ()
  Get the ID of the Fact.
- void setId (int id)
  Set the ID of the Fact.

- Entity & getLeftEntity ()
  Get the left Entity object.
- void setLeftEntity (obelisk::Entity leftEntity)
  Set the left Entity object.
- Entity & getRightEntity ()
  Get the right Entity object.
- void setRightEntity (obelisk::Entity rightEntity)
  Set the right Entity object.
- Verb & getVerb ()
  Get the Verb object.
- void setVerb (obelisk::Verb verb)
  Set the Verb object.
- double & getIsTrue ()
  Gets the isTrue value.
- void setIsTrue (double isTrue)
  Set the Fact as true or false.
- void selectById (sqlite3 *dbConnection)
  Select the Fact from the KnowledgeBase by IDs of the sub-objects.
- void selectByName (sqlite3 *dbConnection)
  Select the Fact from the KnowledgeBase by the name's of the entities and verb.
- void selectActionByFact (sqlite3 *dbConnection, obelisk::Action &action)
  Select an Action from the KnowledgeBase using the provided Fact.
- void insert (sqlite3 *dbConnection)
  Insert the Fact into the KnowledgeBase.
- void updateIsTrue (sqlite3 *dbConnection)
  Update whether or not the fact is true in the KnowledgeBase.

Static Public Member Functions

- static const char * createTable ()
  Create the Fact table in the KnowledgeBase.

Private Attributes

- int id_
  The ID of the Fact in the KnowledgeBase.
- obelisk::Entity leftEntity_
  The Entity from the left side of the expression.
- obelisk::Entity rightEntity_
  The Entity from the right side of the expression.
- obelisk::Verb verb_
  The Verb that represents the relationship in the expression.
- double isTrue_
  Whether or not the fact is considered true or not.

5.12.1 Detailed Description

The Fact model represents truth in the releationship between two entities separated by a verb.

Definition at line 18 of file fact.h.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Fact() [1/3]

```
obelisk::Fact::Fact (
            int id ) [inline]
```

Construct a new Fact object.

Parameters

in	*id*	The ID of the Fact in the KnowledgeBase.

Definition at line 71 of file fact.h.

```
00071                                :
00072                 id_(id),
00073                 leftEntity_(),
00074                 rightEntity_(),
00075                 verb_(),
00076                 isTrue_(0)
00077             {
00078             }
```

5.12.2.2 Fact() [2/3]

```
obelisk::Fact::Fact (
            obelisk::Entity leftEntity,
            obelisk::Entity rightEntity,
            obelisk::Verb verb,
            double isTrue = 0 )  [inline]
```

Construct a new Fact object.

Parameters

in	*leftEntity*	The Entity on the left side of the expression.
in	*rightEntity*	The Entity on the right side of the expression.
in	*verb*	The Verb separating the entities.
in	*isTrue*	Whether or not the fact is true.

Definition at line 90 of file fact.h.

```
00093                                          :
00094                 id_(0),
00095                 leftEntity_(leftEntity),
00096                 rightEntity_(rightEntity),
00097                 verb_(verb),
00098                 isTrue_(isTrue)
00099             {
00100             }
```

5.12.2.3 Fact() [3/3]

```
obelisk::Fact::Fact (
            int id,
            obelisk::Entity leftEntity,
            obelisk::Entity rightEntity,
            obelisk::Verb verb,
            double isTrue = 0 )  [inline]
```

Construct a new Fact object.

Parameters

in	*id*	The ID of the Fact in the KnowledgeBase.
in	*leftEntity*	The Entity on the left side of the expression.
in	*rightEntity*	The Entity on the right side of the expression.
in	*verb*	The Verb separating the entities.
in	*isTrue*	Whether or not the fact is true.

Definition at line 113 of file fact.h.

```
00117                                          :
00118                 id_(id),
00119                 leftEntity_(leftEntity),
00120                 rightEntity_(rightEntity),
```

```
00121                 verb_(verb),
00122                 isTrue_(isTrue)
00123             {
00124             }
```

5.12.3 Member Function Documentation

5.12.3.1 createTable()

```
const char * obelisk::Fact::createTable ( )  [static]
```

Create the Fact table in the KnowledgeBase.

Returns

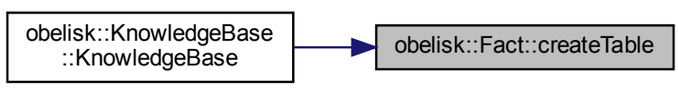
const char* Returns the query used to create the table.

Definition at line 4 of file fact.cpp.

```
00005 {
00006     return R"(
00007         CREATE TABLE "fact" (
00008             "id"           INTEGER NOT NULL UNIQUE,
00009             "left_entity"  INTEGER NOT NULL,
00010             "verb"         INTEGER NOT NULL,
00011             "right_entity" INTEGER NOT NULL,
00012             "is_true"      INTEGER NOT NULL DEFAULT 0,
00013             PRIMARY KEY("id" AUTOINCREMENT),
00014             UNIQUE("left_entity", "right_entity", "verb")
00015             FOREIGN KEY("verb") REFERENCES "verb"("id") ON DELETE RESTRICT,
00016             FOREIGN KEY("right_entity") REFERENCES "entity"("id") ON DELETE RESTRICT,
00017             FOREIGN KEY("left_entity") REFERENCES "entity"("id") ON DELETE RESTRICT
00018         );
00019     )";
00020 }
```

Referenced by obelisk::KnowledgeBase::KnowledgeBase().

Here is the caller graph for this function:



5.12.3.2 getId()

```
int & obelisk::Fact::getId ( )
```

Get the ID of the Fact.

Returns

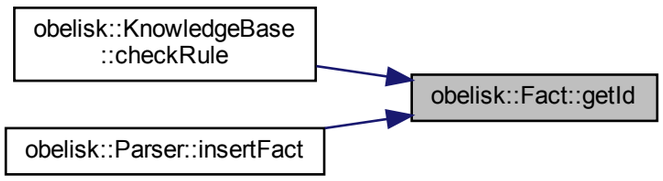
int& Returns the ID.

Definition at line 552 of file fact.cpp.

```
00553 {
00554     return id_;
00555 }
```

Referenced by obelisk::KnowledgeBase::checkRule(), and obelisk::Parser::insertFact().

Here is the caller graph for this function:



5.12.3.3 getIsTrue()

```
double & obelisk::Fact::getIsTrue ( )
```

Gets the isTrue value.

Returns

true If the Fact is considered true.

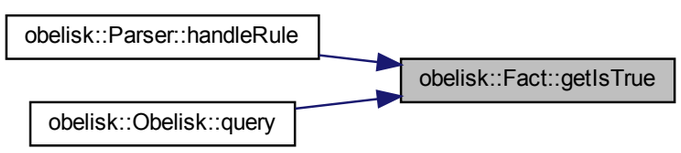
false If the Fact is considered false.

Definition at line 592 of file fact.cpp.

```
00593 {
00594     return isTrue_;
00595 }
```

Referenced by obelisk::Parser::handleRule(), and obelisk::Obelisk::query().

Here is the caller graph for this function:



5.12.3.4 getLeftEntity()

```
obelisk::Entity & obelisk::Fact::getLeftEntity ( )
```

Get the left Entity object.

Returns

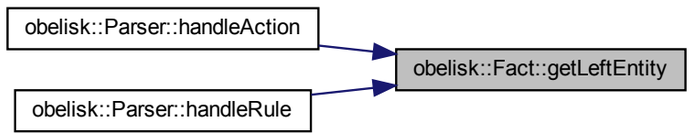
Entity& The left Entity.

Definition at line 562 of file fact.cpp.

```
00563 {
00564     return leftEntity_;
00565 }
```

Referenced by obelisk::Parser::handleAction(), and obelisk::Parser::handleRule().

Here is the caller graph for this function:



5.12.3.5 getRightEntity()

```
obelisk::Entity & obelisk::Fact::getRightEntity ( )
```

Get the right Entity object.

Returns

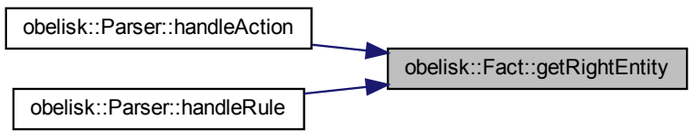
Entity& The right Entity.

Definition at line 572 of file fact.cpp.

```
00573 {
00574     return rightEntity_;
00575 }
```

Referenced by obelisk::Parser::handleAction(), and obelisk::Parser::handleRule().

Here is the caller graph for this function:



5.12.3.6 getVerb()

```
obelisk::Verb & obelisk::Fact::getVerb ( )
```

Get the Verb object.

Returns

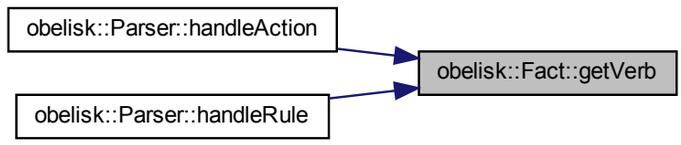
Verb& The Verb.

Definition at line 582 of file fact.cpp.

```
00583 {
00584     return verb_;
00585 }
```

Referenced by obelisk::Parser::handleAction(), and obelisk::Parser::handleRule().

Here is the caller graph for this function:



5.12.3.7 insert()

```
void obelisk::Fact::insert (
            sqlite3 * dbConnection )
```

Insert the Fact into the KnowledgeBase.

Parameters

in	*dbConnection*	The database connection to use.

Definition at line 345 of file fact.cpp.

```
00346 {
00347     if (dbConnection == nullptr)
00348     {
00349         throw obelisk::DatabaseException("database isn't open");
00350     }
00351
00352     sqlite3_stmt* ppStmt = nullptr;
00353
00354     auto result = sqlite3_prepare_v2(dbConnection,
00355         "INSERT INTO fact (left_entity, right_entity, verb, is_true) VALUES (?, ?, ?, ?)",
00356         -1,
00357         &ppStmt,
00358         nullptr);
00359     if (result != SQLITE_OK)
00360     {
00361         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00362     }
00363
00364     result = sqlite3_bind_int(ppStmt, 1, getLeftEntity().getId());
00365     switch (result)
00366     {
00367         case SQLITE_OK :
00368             break;
00369         case SQLITE_TOOBIG :
00370             throw obelisk::DatabaseSizeException();
00371             break;
00372         case SQLITE_RANGE :
00373             throw obelisk::DatabaseRangeException();
00374             break;
00375         case SQLITE_NOMEM :
00376             throw obelisk::DatabaseMemoryException();
00377             break;
```

```
00378         default :
00379             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00380             break;
00381     }
00382
00383     result = sqlite3_bind_int(ppStmt, 2, getRightEntity().getId());
00384     switch (result)
00385     {
00386         case SQLITE_OK :
00387             break;
00388         case SQLITE_TOOBIG :
00389             throw obelisk::DatabaseSizeException();
00390             break;
00391         case SQLITE_RANGE :
00392             throw obelisk::DatabaseRangeException();
00393             break;
00394         case SQLITE_NOMEM :
00395             throw obelisk::DatabaseMemoryException();
00396             break;
00397         default :
00398             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00399             break;
00400     }
00401
00402     result = sqlite3_bind_int(ppStmt, 3, getVerb().getId());
00403     switch (result)
00404     {
00405         case SQLITE_OK :
00406             break;
00407         case SQLITE_TOOBIG :
00408             throw obelisk::DatabaseSizeException();
00409             break;
00410         case SQLITE_RANGE :
00411             throw obelisk::DatabaseRangeException();
00412             break;
00413         case SQLITE_NOMEM :
00414             throw obelisk::DatabaseMemoryException();
00415             break;
00416         default :
00417             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00418             break;
00419     }
00420
00421     result = sqlite3_bind_int(ppStmt, 4, getIsTrue());
00422     switch (result)
00423     {
00424         case SQLITE_OK :
00425             break;
00426         case SQLITE_TOOBIG :
00427             throw obelisk::DatabaseSizeException();
00428             break;
00429         case SQLITE_RANGE :
00430             throw obelisk::DatabaseRangeException();
00431             break;
00432         case SQLITE_NOMEM :
00433             throw obelisk::DatabaseMemoryException();
00434             break;
00435         default :
00436             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00437             break;
00438     }
00439
00440     result = sqlite3_step(ppStmt);
00441     switch (result)
00442     {
00443         case SQLITE_DONE :
00444             setId((int) sqlite3_last_insert_rowid(dbConnection));
00445             sqlite3_set_last_insert_rowid(dbConnection, 0);
00446             break;
00447         case SQLITE_CONSTRAINT :
00448             throw obelisk::DatabaseConstraintException(
00449                 sqlite3_errmsg(dbConnection));
00450         case SQLITE_BUSY :
00451             throw obelisk::DatabaseBusyException();
00452             break;
00453         case SQLITE_MISUSE :
00454             throw obelisk::DatabaseMisuseException();
00455             break;
00456         default :
00457             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00458             break;
00459     }
00460
00461     result = sqlite3_finalize(ppStmt);
00462     if (result != SQLITE_OK)
00463     {
00464         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00465     }
00466 }
```

5.12.3.8 selectActionByFact()

```
void obelisk::Fact::selectActionByFact (
            sqlite3 * dbConnection,
            obelisk::Action & action )
```

Select an Action from the KnowledgeBase using the provided Fact.

Parameters

in	*dbConnection*	The database connection to use.
out	*action*	The Action to take based on the provided fact.

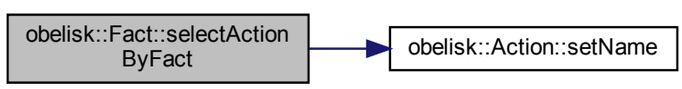
Definition at line 279 of file fact.cpp.

```
00281 {
00282     if (dbConnection == nullptr)
00283     {
00284         throw obelisk::DatabaseException("database isn't open");
00285     }
00286
00287     sqlite3_stmt* ppStmt = nullptr;
00288
00289     auto result = sqlite3_prepare_v2(dbConnection,
00290         "SELECT CASE f.is_true WHEN 0 THEN (SELECT name FROM action WHERE id = fa.id) WHEN 1 THEN (SELECT
      name from action WHERE id = ta.id) END action FROM suggest_action LEFT JOIN action ta ON ta.id =
      suggest_action.true_action LEFT JOIN action fa ON fa.id = suggest_action.false_action LEFT JOIN fact f ON
      f.id = suggest_action.fact WHERE (f.id = ?)",
00291         -1,
00292         &ppStmt,
00293         nullptr);
00294     if (result != SQLITE_OK)
00295     {
00296         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00297     }
00298
00299     result = sqlite3_bind_int(ppStmt, 1, getId());
00300     switch (result)
00301     {
00302         case SQLITE_OK :
00303             break;
00304         case SQLITE_TOOBIG :
00305             throw obelisk::DatabaseSizeException();
00306             break;
00307         case SQLITE_RANGE :
00308             throw obelisk::DatabaseRangeException();
00309             break;
00310         case SQLITE_NOMEM :
00311             throw obelisk::DatabaseMemoryException();
00312             break;
00313         default :
00314             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00315             break;
00316     }
00317
00318     result = sqlite3_step(ppStmt);
00319     switch (result)
00320     {
00321         case SQLITE_DONE :
00322             // no rows in the database
00323             break;
00324         case SQLITE_ROW :
00325             action.setName((char*) sqlite3_column_text(ppStmt, 0));
00326             break;
00327         case SQLITE_BUSY :
00328             throw obelisk::DatabaseBusyException();
00329             break;
00330         case SQLITE_MISUSE :
00331             throw obelisk::DatabaseMisuseException();
00332             break;
00333         default :
00334             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00335             break;
00336     }
00337
00338     result = sqlite3_finalize(ppStmt);
00339     if (result != SQLITE_OK)
00340     {
00341         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00342     }
00343 }
```

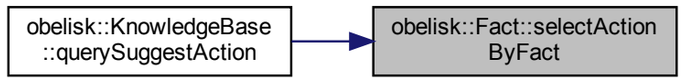
References obelisk::Action::setName().

Referenced by obelisk::KnowledgeBase::querySuggestAction().

Here is the call graph for this function:



Here is the caller graph for this function:



5.12.3.9 selectById()

```
void obelisk::Fact::selectById (
            sqlite3 * dbConnection )
```

Select the Fact from the KnowledgeBase by IDs of the sub-objects.

Parameters

in	*dbConnection*	The database connection to use.

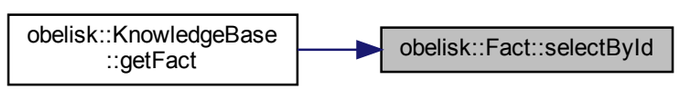
Definition at line 22 of file fact.cpp.

```
00023 {
00024     if (dbConnection == nullptr)
00025     {
00026         throw obelisk::DatabaseException("database isn't open");
00027     }
00028
00029     sqlite3_stmt* ppStmt = nullptr;
00030
00031     const char* query;
00032     if (getId() == 0)
00033     {
00034         query
00035             = "SELECT id, left_entity, right_entity, verb, is_true FROM fact WHERE (left_entity=? AND
       right_entity=? AND verb=?)";
00036     }
00037     else
00038     {
00039         query
00040             = "SELECT id, left_entity, right_entity, verb, is_true FROM fact WHERE (id=?)";
00041     }
00042     auto result = sqlite3_prepare_v2(dbConnection, query, -1, &ppStmt, nullptr);
00043     if (result != SQLITE_OK)
00044     {
00045         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00046     }
00047
00048     if (getId() == 0)
00049     {
00050         result = sqlite3_bind_int(ppStmt, 1, getLeftEntity().getId());
00051         switch (result)
00052         {
00053             case SQLITE_OK :
00054                 break;
00055             case SQLITE_TOOBIG :
00056                 throw obelisk::DatabaseSizeException();
00057                 break;
00058             case SQLITE_RANGE :
00059                 throw obelisk::DatabaseRangeException();
00060                 break;
00061             case SQLITE_NOMEM :
00062                 throw obelisk::DatabaseMemoryException();
00063                 break;
00064             default :
00065                 throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00066                 break;
00067         }
00068
00069         result = sqlite3_bind_int(ppStmt, 2, getRightEntity().getId());
00070         switch (result)
00071         {
00072             case SQLITE_OK :
00073                 break;
00074             case SQLITE_TOOBIG :
00075                 throw obelisk::DatabaseSizeException();
00076                 break;
00077             case SQLITE_RANGE :
00078                 throw obelisk::DatabaseRangeException();
00079                 break;
00080             case SQLITE_NOMEM :
00081                 throw obelisk::DatabaseMemoryException();
00082                 break;
00083             default :
```

```
00084                 throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00085                 break;
00086         }
00087
00088         result = sqlite3_bind_int(ppStmt, 3, getVerb().getId());
00089         switch (result)
00090         {
00091             case SQLITE_OK :
00092                 break;
00093             case SQLITE_TOOBIG :
00094                 throw obelisk::DatabaseSizeException();
00095                 break;
00096             case SQLITE_RANGE :
00097                 throw obelisk::DatabaseRangeException();
00098                 break;
00099             case SQLITE_NOMEM :
00100                 throw obelisk::DatabaseMemoryException();
00101                 break;
00102             default :
00103                 throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00104                 break;
00105         }
00106     }
00107     else
00108     {
00109         result = sqlite3_bind_int(ppStmt, 1, getId());
00110         switch (result)
00111         {
00112             case SQLITE_OK :
00113                 break;
00114             case SQLITE_TOOBIG :
00115                 throw obelisk::DatabaseSizeException();
00116                 break;
00117             case SQLITE_RANGE :
00118                 throw obelisk::DatabaseRangeException();
00119                 break;
00120             case SQLITE_NOMEM :
00121                 throw obelisk::DatabaseMemoryException();
00122                 break;
00123             default :
00124                 throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00125                 break;
00126         }
00127     }
00128
00129     result = sqlite3_step(ppStmt);
00130     switch (result)
00131     {
00132         case SQLITE_DONE :
00133             // no rows in the database
00134             break;
00135         case SQLITE_ROW :
00136             setId(sqlite3_column_int(ppStmt, 0));
00137             getLeftEntity().setId(sqlite3_column_int(ppStmt, 1));
00138             getRightEntity().setId(sqlite3_column_int(ppStmt, 2));
00139             getVerb().setId(sqlite3_column_int(ppStmt, 3));
00140             setIsTrue(sqlite3_column_int(ppStmt, 4));
00141             break;
00142         case SQLITE_BUSY :
00143             throw obelisk::DatabaseBusyException();
00144             break;
00145         case SQLITE_MISUSE :
00146             throw obelisk::DatabaseMisuseException();
00147             break;
00148         default :
00149             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00150             break;
00151     }
00152
00153     result = sqlite3_finalize(ppStmt);
00154     if (result != SQLITE_OK)
00155     {
00156         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00157     }
00158 }
```

Referenced by obelisk::KnowledgeBase::getFact().

Here is the caller graph for this function:



5.12.3.10 selectByName()

```
void obelisk::Fact::selectByName (
             sqlite3 * dbConnection )
```

Select the Fact from the KnowledgeBase by the name's of the entities and verb.

Parameters

in	*dbConnection*	The database connection to use.

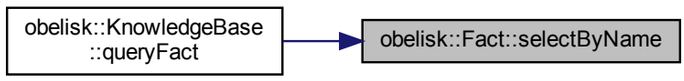
Definition at line 160 of file fact.cpp.

```
00161 {
00162     if (dbConnection == nullptr)
00163     {
00164         throw obelisk::DatabaseException("database isn't open");
00165     }
00166
00167     sqlite3_stmt* ppStmt = nullptr;
00168
00169     auto result = sqlite3_prepare_v2(dbConnection,
00170         "SELECT fact.id, fact.left_entity, fact.right_entity, fact.verb, fact.is_true FROM fact LEFT JOIN
      entity le ON le.id = fact.left_entity LEFT JOIN entity re ON re.id = fact.right_entity LEFT JOIN verb v
      ON fact.verb = v.id WHERE (le.name=? AND v.name=? AND re.name=?)",
00171         -1,
00172         &ppStmt,
00173         nullptr);
00174     if (result != SQLITE_OK)
00175     {
00176         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00177     }
00178
00179     result = sqlite3_bind_text(ppStmt,
00180         1,
00181         getLeftEntity().getName().c_str(),
00182         -1,
00183         SQLITE_STATIC);
00184     switch (result)
00185     {
00186         case SQLITE_OK :
00187             break;
00188         case SQLITE_TOOBIG :
00189             throw obelisk::DatabaseSizeException();
00190             break;
00191         case SQLITE_RANGE :
00192             throw obelisk::DatabaseRangeException();
00193             break;
00194         case SQLITE_NOMEM :
00195             throw obelisk::DatabaseMemoryException();
00196             break;
00197         default :
00198             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00199             break;
00200     }
00201
00202     result = sqlite3_bind_text(ppStmt,
00203         2,
00204         getVerb().getName().c_str(),
00205         -1,
00206         SQLITE_STATIC);
00207     switch (result)
00208     {
00209         case SQLITE_OK :
00210             break;
00211         case SQLITE_TOOBIG :
00212             throw obelisk::DatabaseSizeException();
00213             break;
00214         case SQLITE_RANGE :
00215             throw obelisk::DatabaseRangeException();
00216             break;
00217         case SQLITE_NOMEM :
00218             throw obelisk::DatabaseMemoryException();
00219             break;
00220         default :
00221             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00222             break;
00223     }
00224
00225     result = sqlite3_bind_text(ppStmt,
00226         3,
00227         getRightEntity().getName().c_str(),
00228         -1,
00229         SQLITE_STATIC);
00230     switch (result)
00231     {
00232         case SQLITE_OK :
00233             break;
00234         case SQLITE_TOOBIG :
00235             throw obelisk::DatabaseSizeException();
00236             break;
00237         case SQLITE_RANGE :
00238             throw obelisk::DatabaseRangeException();
00239             break;
00240         case SQLITE_NOMEM :
00241             throw obelisk::DatabaseMemoryException();
00242             break;
00243         default :
00244             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00245             break;
00246     }
00247
00248     result = sqlite3_step(ppStmt);
00249     switch (result)
00250     {
00251         case SQLITE_DONE :
00252             // no rows in the database
00253             break;
00254         case SQLITE_ROW :
```

```
00255             setId(sqlite3_column_int(ppStmt, 0));
00256             getLeftEntity().setId(sqlite3_column_int(ppStmt, 1));
00257             getRightEntity().setId(sqlite3_column_int(ppStmt, 2));
00258             getVerb().setId(sqlite3_column_int(ppStmt, 3));
00259             setIsTrue(sqlite3_column_int(ppStmt, 4));
00260             break;
00261         case SQLITE_BUSY :
00262             throw obelisk::DatabaseBusyException();
00263             break;
00264         case SQLITE_MISUSE :
00265             throw obelisk::DatabaseMisuseException();
00266             break;
00267         default :
00268             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00269             break;
00270     }
00271
00272     result = sqlite3_finalize(ppStmt);
00273     if (result != SQLITE_OK)
00274     {
00275         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00276     }
00277 }
```

Referenced by obelisk::KnowledgeBase::queryFact().

Here is the caller graph for this function:



5.12.3.11 setId()

```
void obelisk::Fact::setId (
            int id )
```

Set the ID of the Fact.

Parameters

in	*id*	Set the ID of the Fact.

Definition at line 557 of file fact.cpp.

```
00558 {
00559     id_ = id;
00560 }
```

5.12.3.12 setIsTrue()

```
void obelisk::Fact::setIsTrue (
            double isTrue )
```

Set the Fact as true or false.

Parameters

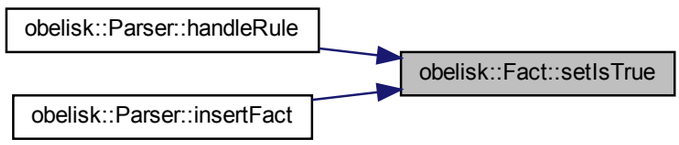
in	*isTrue*	Whether or not the Fact is true.

Definition at line 597 of file fact.cpp.

```
00598 {
00599     isTrue_ = isTrue;
00600 }
```

Referenced by obelisk::Parser::handleRule(), and obelisk::Parser::insertFact().

Here is the caller graph for this function:



5.12.3.13 setLeftEntity()

```
void obelisk::Fact::setLeftEntity (
            obelisk::Entity leftEntity )
```

Set the left Entity object.

Parameters

in	*leftEntity*	The left Entity to set.

Definition at line 567 of file fact.cpp.

```
00568 {
00569     leftEntity_ = leftEntity;
00570 }
```

5.12.3.14 setRightEntity()

```
void obelisk::Fact::setRightEntity (
            obelisk::Entity rightEntity )
```

Set the right Entity object.

Parameters

in	*rightEntity*	The right Entity to set.

Definition at line 577 of file fact.cpp.

```
00578 {
00579     rightEntity_ = rightEntity;
00580 }
```

5.12.3.15 setVerb()

```
void obelisk::Fact::setVerb (
            obelisk::Verb verb )
```

Set the Verb object.

Parameters

in	*verb*	The Verb.

Definition at line 587 of file fact.cpp.

```
00588 {
00589     verb_ = verb;
00590 }
```

5.12.3.16 updateIsTrue()

```
void obelisk::Fact::updateIsTrue (
            sqlite3 * dbConnection )
```

Update whether or not the fact is true in the KnowledgeBase.

Parameters

in	*dbConnection*	The database connection.

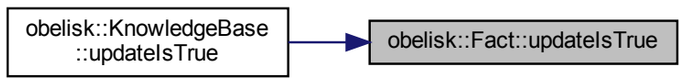
Definition at line 468 of file fact.cpp.

```
00469 {
00470     if (dbConnection == nullptr)
00471     {
00472         throw obelisk::DatabaseException("database isn't open");
00473     }
00474
00475     sqlite3_stmt* ppStmt = nullptr;
00476
00477     auto result = sqlite3_prepare_v2(dbConnection,
00478         "UPDATE fact SET is_true=? WHERE id=?",
00479         -1,
00480         &ppStmt,
00481         nullptr);
00482     if (result != SQLITE_OK)
00483     {
00484         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00485     }
00486
00487     result = sqlite3_bind_int(ppStmt, 1, getIsTrue());
00488     switch (result)
00489     {
00490         case SQLITE_OK :
00491             break;
00492         case SQLITE_TOOBIG :
00493             throw obelisk::DatabaseSizeException();
00494             break;
00495         case SQLITE_RANGE :
00496             throw obelisk::DatabaseRangeException();
00497             break;
00498         case SQLITE_NOMEM :
00499             throw obelisk::DatabaseMemoryException();
00500             break;
00501         default :
00502             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00503             break;
00504     }
00505
00506     result = sqlite3_bind_int(ppStmt, 2, getId());
00507     switch (result)
00508     {
00509         case SQLITE_OK :
00510             break;
00511         case SQLITE_TOOBIG :
00512             throw obelisk::DatabaseSizeException();
00513             break;
00514         case SQLITE_RANGE :
00515             throw obelisk::DatabaseRangeException();
00516             break;
00517         case SQLITE_NOMEM :
00518             throw obelisk::DatabaseMemoryException();
00519             break;
00520         default :
00521             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00522             break;
00523     }
00524
00525     result = sqlite3_step(ppStmt);
00526     switch (result)
00527     {
00528         case SQLITE_DONE :
00529             // Row updated
00530             break;
00531         case SQLITE_CONSTRAINT :
00532             throw obelisk::DatabaseConstraintException(
00533                 sqlite3_errmsg(dbConnection));
00534         case SQLITE_BUSY :
00535             throw obelisk::DatabaseBusyException();
00536             break;
00537         case SQLITE_MISUSE :
00538             throw obelisk::DatabaseMisuseException();
00539             break;
00540         default :
00541             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00542             break;
00543     }
00544
00545     result = sqlite3_finalize(ppStmt);
```

```
00546     if (result != SQLITE_OK)
00547     {
00548         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00549     }
00550 }
```

Referenced by obelisk::KnowledgeBase::updateIsTrue().

Here is the caller graph for this function:



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

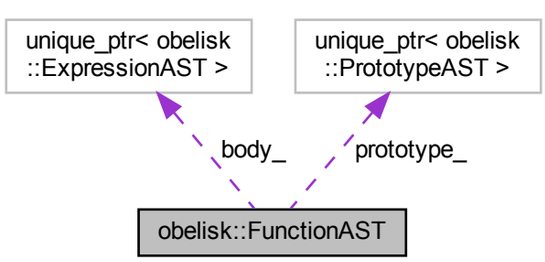
- src/lib/include/models/fact.h
- src/lib/models/fact.cpp

5.13 obelisk::FunctionAST Class Reference

A Funcion AST node.

`#include <function_ast.h>`

Collaboration diagram for obelisk::FunctionAST:



Public Member Functions

- FunctionAST (std::unique_ptr< PrototypeAST > prototype, std::unique_ptr< ExpressionAST > body)
  Construct a new FunctionAST object.
- llvm::Function * codegen ()
  Generate LLVM IR code.

Private Member Functions

- std::unique_ptr< PrototypeAST > getPrototype ()
  Get the prototype.
- void setPrototype (std::unique_ptr< PrototypeAST > prototype)
  Set the prototype.

Private Attributes

- std::unique_ptr< PrototypeAST > prototype_

  The prototype of the function.
- std::unique_ptr< ExpressionAST > body_

  The body of the function.

5.13.1 Detailed Description

A Funcion AST node.

Definition at line 15 of file function_ast.h.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 FunctionAST()

```
obelisk::FunctionAST::FunctionAST (
            std::unique_ptr< PrototypeAST > prototype,
            std::unique_ptr< ExpressionAST > body )  [inline]
```

Construct a new FunctionAST object.

Parameters

in	*prototype*	The prototype of the function.
in	*body*	The body of the function.

Definition at line 51 of file function_ast.h.

```
00052                                                   :
00053                 prototype_(std::move(prototype)),
00054                 body_(std::move(body))
00055             {
00056             }
```

5.13.3 Member Function Documentation

5.13.3.1 codegen()

```
llvm::Function * obelisk::FunctionAST::codegen ( )
```

Generate LLVM IR code.

Returns

llvm::Function* Returns the LLVM IR function code.

Definition at line 6 of file function_ast.cpp.

```
00007 {
00008     llvm::Function *theFunction = TheModule->getFunction(prototype_->getName());
00009
00010     if (!theFunction)
00011     {
00012         theFunction = prototype_->codegen();
00013     }
00014
00015     if (!theFunction)
00016     {
00017         return nullptr;
00018     }
00019
00020     llvm::BasicBlock *bB
```

```
00021         = llvm::BasicBlock::Create(*TheContext, "entry", theFunction);
00022     Builder->SetInsertPoint(bB);
00023
00024     NamedValues.clear();
00025     for (auto &arg : theFunction->args())
00026     {
00027         NamedValues[std::string(arg.getName())] = &arg;
00028     }
00029
00030     if (llvm::Value *retVal = body_->codegen())
00031     {
00032         Builder->CreateRet(retVal);
00033         llvm::verifyFunction(*theFunction);
00034         return theFunction;
00035     }
00036
00037     theFunction->eraseFromParent();
00038     return nullptr;
00039 }
```

References body_, obelisk::Builder, obelisk::NamedValues, prototype_, obelisk::TheContext, and obelisk::TheModule.

5.13.3.2 getPrototype()

```
std::unique_ptr<PrototypeAST> obelisk::FunctionAST::getPrototype ( )  [private]
```

Get the prototype.

Returns

std::unique_ptr<PrototypeAST> Returns the prototype AST.

5.13.3.3 setPrototype()

```
void obelisk::FunctionAST::setPrototype (
            std::unique_ptr< PrototypeAST > prototype )  [private]
```

Set the prototype.

Parameters

in	*prototype*	Set the prototype.

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

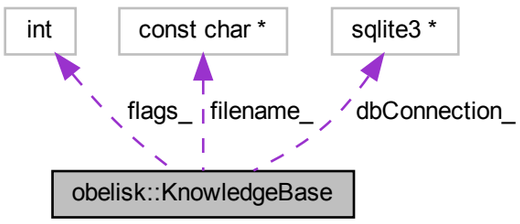
- src/ast/function_ast.h
- src/ast/function_ast.cpp

5.14 obelisk::KnowledgeBase Class Reference

The KnowledgeBase class represents a collection of facts, rules, actions, and related language connectors.

```
#include <knowledge_base.h>
```

Collaboration diagram for obelisk::KnowledgeBase:



Public Member Functions

- KnowledgeBase (const char *filename, int flags)

  Construct a new KnowledgeBase object.
- KnowledgeBase (const char *filename)

  Construct a new KnowledgeBase object.
- ~KnowledgeBase ()

  Destroy the KnowledgeBase object.
- void addEntities (std::vector< obelisk::Entity > &entities)

  Add entities to the KnowledgeBase.
- void addVerbs (std::vector< obelisk::Verb > &verbs)

  Add verbs to the KnowledgeBase.
- void addActions (std::vector< obelisk::Action > &actions)

  Add actions to the KnowledgeBase.
- void addFacts (std::vector< obelisk::Fact > &facts)

  Add facts to the KnowledgeBase.
- void addSuggestActions (std::vector< obelisk::SuggestAction > &suggestActions)

  Add suggested actions to the KnowledgeBase.
- void addRules (std::vector< obelisk::Rule > &rules)

  Add rules to the KnowledgeBase.
- void getEntity (obelisk::Entity &entity)

  Get an Entity object based on the ID it contains.
- void getVerb (obelisk::Verb &verb)

  Get a Verb object based on the ID it contains.
- void getAction (obelisk::Action &action)

  Get an Action based on the ID it contains.
- void getFact (obelisk::Fact &fact)

  Get a Fact object based on the ID it contains.
- void getSuggestAction (obelisk::SuggestAction &suggestAction)

  Get a SuggestAction based on the ID it contains.
- void getRule (obelisk::Rule &rule)

  Get a Rule based on the ID it contains.
- void checkRule (obelisk::Fact &fact)

  Check if a rule looks for this Fact, if so update its truth.
- void updateIsTrue (obelisk::Fact &fact)

  Update the is true field in the KnowledgeBase.
- void queryFact (obelisk::Fact &fact)

  Query the KnowledgeBase to see if a Fact is true or false.
- void querySuggestAction (obelisk::Fact &fact, obelisk::Action &action)

  Query the KnowledgeBase to get a suggested action based on a Fact. If a SuggestAction doesn't exist, it will return an empty Action.
- void getFloat (float &result1, float &result2, double var)

  Take a float and divide it into 2 floats.
- void getDouble (double &result, float var1, float var2)

  Combines 2 separated floats back into a double.

Private Member Functions

- void enableForeignKeys ()

  Enable foreign key functionality in the open database.
- void createTable (std::function< const char *()> function)

  Create the tables in the database.

Private Attributes

- const char * filename_

  The filename of the opened KnowledgeBase.
- sqlite3 * dbConnection_ = nullptr

  The SQLite connection handle.
- int flags_

  The user passed flags to use when opening the database.

5.14.1 Detailed Description

The KnowledgeBase class represents a collection of facts, rules, actions, and related language connectors.

Definition at line 25 of file knowledge_base.h.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 KnowledgeBase() [1/2]

```
obelisk::KnowledgeBase::KnowledgeBase (
            const char * filename,
            int flags )
```

Construct a new KnowledgeBase object.

Parameters

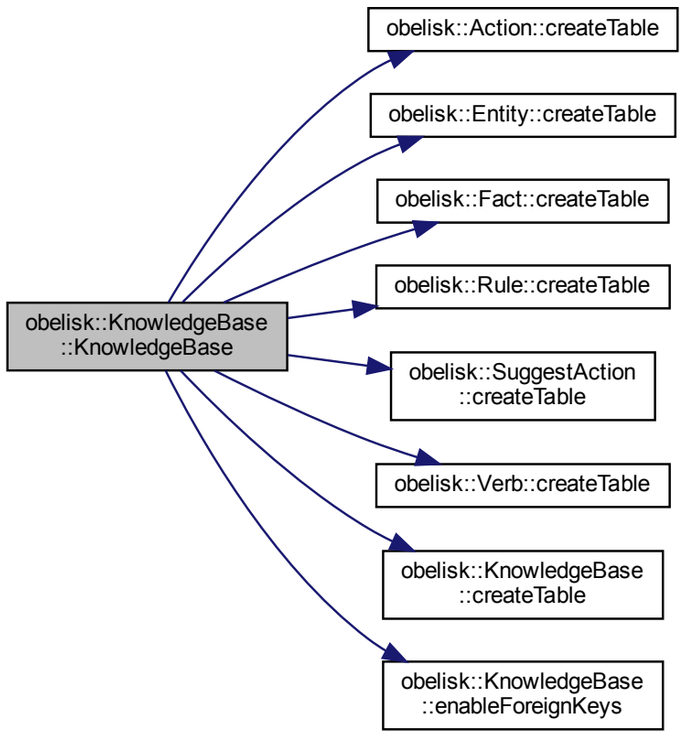
in	*filename*	The name of the file to save the knowledge base as.
in	*flags*	The flags to open the KnowledgeBase with.

Definition at line 9 of file knowledge_base.cpp.

```
00010 {
00011     filename_ = std::move(filename);
00012     flags_    = std::move(flags);
00013
00014     std::filesystem::path path {filename};
00015     auto dbExists = std::filesystem::exists(path);
00016
00017     auto result = sqlite3_open_v2(filename, &dbConnection_, flags, NULL);
00018     if (result != SQLITE_OK)
00019     {
00020         throw new KnowledgeBaseException("database could not be opened");
00021     }
00022
00023     enableForeignKeys();
00024
00025     if (!dbExists)
00026     {
00027         createTable(obelisk::Action::createTable);
00028         createTable(obelisk::Entity::createTable);
00029         createTable(obelisk::Verb::createTable);
00030         createTable(obelisk::Fact::createTable);
00031         createTable(obelisk::Rule::createTable);
00032         createTable(obelisk::SuggestAction::createTable);
00033     }
00034 }
```

References obelisk::Action::createTable(), obelisk::Entity::createTable(), obelisk::Fact::createTable(), obelisk::Rule::createTable(), obelisk::SuggestAction::createTable(), obelisk::Verb::createTable(), createTable(), dbConnection_, enableForeignKeys(), filename_, and flags_.

Here is the call graph for this function:



5.14.2.2 KnowledgeBase() [2/2]

```
obelisk::KnowledgeBase::KnowledgeBase (
            const char * filename )  [inline]
```

Construct a new KnowledgeBase object.

Parameters

in	*filename*	The name of the file to save the knowledge base as.

Definition at line 77 of file knowledge_base.h.

```
00077                                       :
00078                 KnowledgeBase(filename,
00079                     SQLITE_OPEN_READWRITE | SQLITE_OPEN_CREATE)
00080             {
00081             }
```

5.14.2.3 ~KnowledgeBase()

```
obelisk::KnowledgeBase::~KnowledgeBase ( )
```

Destroy the KnowledgeBase object.

This will close the opened KnowledgeBase before destroying it.

Definition at line 36 of file knowledge_base.cpp.

```
00037 {
00038     if (dbConnection_)
00039     {
00040         sqlite3_close_v2(dbConnection_);
00041     }
00042 }
```

5.14.3 Member Function Documentation

5.14.3.1 addActions()

```
void obelisk::KnowledgeBase::addActions (
            std::vector< obelisk::Action > & actions )
```

Add actions to the KnowledgeBase.

Parameters

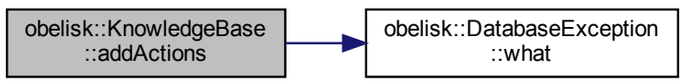
in,out	actions	The actions to add. If the insert is successful it will have a row ID, if nto the ID will be 0.

Definition at line 124 of file knowledge_base.cpp.

```
00125 {
00126     for (auto& action : actions)
00127     {
00128         try
00129         {
00130             action.insert(dbConnection_);
00131         }
00132         catch (obelisk::DatabaseConstraintException& exception)
00133         {
00134             // ignore unique constraint error
00135             if (std::strcmp(exception.what(),
00136                     "UNIQUE constraint failed: action.name")
00137                 != 0)
00138             {
00139                 throw;
00140             }
00141         }
00142     }
00143 }
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:



5.14.3.2 addEntities()

```
void obelisk::KnowledgeBase::addEntities (
            std::vector< obelisk::Entity > & entities )
```

Add entities to the KnowledgeBase.

Parameters

in,out	entities	The entities to add. If the insert is successful it will have a row ID, if not the ID will be 0.

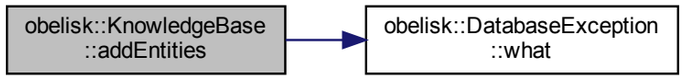
Definition at line 82 of file knowledge_base.cpp.

```
00083 {
00084     for (auto& entity : entities)
00085     {
00086         try
00087         {
00088             entity.insert(dbConnection_);
00089         }
00090         catch (obelisk::DatabaseConstraintException& exception)
```

```
00091         {
00092             // ignore unique constraint error
00093             if (std::strcmp(exception.what(),
00094                     "UNIQUE constraint failed: entity.name")
00095                 != 0)
00096             {
00097                 throw;
00098             }
00099         }
00100     }
00101 }
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:



5.14.3.3 addFacts()

```
void obelisk::KnowledgeBase::addFacts (
            std::vector< obelisk::Fact > & facts )
```

Add facts to the KnowledgeBase.

Parameters

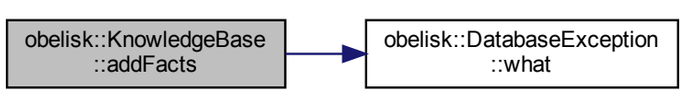
in,out	*facts*	The facts to add. If the insert is successful it will have a row ID, if not the ID will be 0.

Definition at line 145 of file knowledge_base.cpp.

```
00146 {
00147     for (auto& fact : facts)
00148     {
00149         try
00150         {
00151             fact.insert(dbConnection_);
00152         }
00153         catch (obelisk::DatabaseConstraintException& exception)
00154         {
00155             // ignore unique constraint error
00156             if (std::strcmp(exception.what(),
00157                     "UNIQUE constraint failed: fact.left_entity, fact.right_entity, fact.verb")
00158                 != 0)
00159             {
00160                 throw;
00161             }
00162         }
00163     }
00164 }
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:



5.14.3.4 addRules()

```
void obelisk::KnowledgeBase::addRules (
            std::vector< obelisk::Rule > & rules )
```

Add rules to the KnowledgeBase.

Parameters

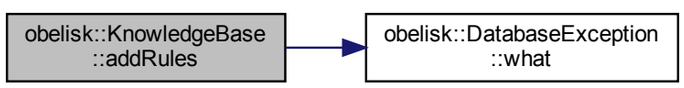
in,out	rules	The rules to add. If the insert is successful it will have a row ID, if not the ID will be 0.

Definition at line 188 of file knowledge_base.cpp.

```
00189 {
00190     for (auto& rule : rules)
00191     {
00192         try
00193         {
00194             rule.insert(dbConnection_);
00195         }
00196         catch (obelisk::DatabaseConstraintException& exception)
00197         {
00198             // ignore unique constraint error
00199             if (std::strcmp(exception.what(),
00200                     "UNIQUE constraint failed: rule.fact, rule.reason")
00201                 != 0)
00202             {
00203                 throw;
00204             }
00205         }
00206     }
00207 }
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:



5.14.3.5 addSuggestActions()

```
void obelisk::KnowledgeBase::addSuggestActions (
            std::vector< obelisk::SuggestAction > & suggestActions )
```

Add suggested actions to the KnowledgeBase.

Parameters

in,out	suggestActions	The suggested actions to add. If the insert is successful it will have a row ID, if not the ID will be 0.

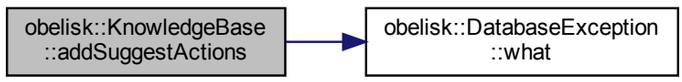
Definition at line 166 of file knowledge_base.cpp.

```
00168 {
00169     for (auto& suggestAction : suggestActions)
00170     {
00171         try
00172         {
00173             suggestAction.insert(dbConnection_);
00174         }
00175         catch (obelisk::DatabaseConstraintException& exception)
00176         {
00177             // ignore unique constraint error
00178             if (std::strcmp(exception.what(),
00179                     "UNIQUE constraint failed: suggest_action.fact, suggest_action.true_action,
      suggest_action.false_action")
00180                 != 0)
00181             {
00182                 throw;
```

```
00183             }
00184         }
00185     }
00186 }
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:



5.14.3.6 addVerbs()

```
void obelisk::KnowledgeBase::addVerbs (
            std::vector< obelisk::Verb > & verbs )
```

Add verbs to the KnowledgeBase.

Parameters

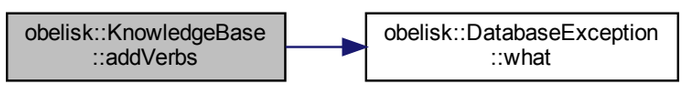
in,out	*verbs*	The verbs to add. If the insert is successful it will have a row ID, if not the ID will be 0.

Definition at line 103 of file knowledge_base.cpp.

```
00104 {
00105     for (auto& verb : verbs)
00106     {
00107         try
00108         {
00109             verb.insert(dbConnection_);
00110         }
00111         catch (obelisk::DatabaseConstraintException& exception)
00112         {
00113             // ignore unique constraint error
00114             if (std::strcmp(exception.what(),
00115                     "UNIQUE constraint failed: verb.name")
00116                 != 0)
00117             {
00118                 throw;
00119             }
00120         }
00121     }
00122 }
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:



5.14.3.7 checkRule()

```
void obelisk::KnowledgeBase::checkRule (
            obelisk::Fact & fact )
```

Check if a rule looks for this Fact, if so update its truth.

Parameters

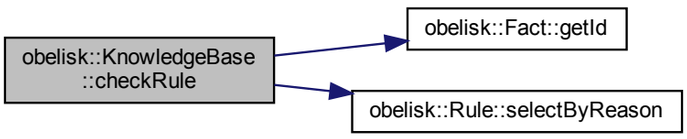
in,out	*fact*	The Fact to check for existing rules.

Definition at line 240 of file knowledge_base.cpp.

```
00241 {
00242     std::vector<obelisk::Rule> rules;
00243     obelisk::Rule::selectByReason(dbConnection_, fact.getId(), rules);
00244     for (auto& rule : rules)
00245     {
00246         auto reason = rule.getReason();
00247         getFact(reason);
00248         if (reason.getIsTrue() > 0)
00249         {
00250             auto updateFact = rule.getFact();
00251             updateFact.setIsTrue(1.0);
00252             updateFact.updateIsTrue(dbConnection_);
00253             checkRule(updateFact);
00254         }
00255     }
00256 }
```

References obelisk::Fact::getId(), and obelisk::Rule::selectByReason().

Here is the call graph for this function:



5.14.3.8 createTable()

```
void obelisk::KnowledgeBase::createTable (
            std::function< const char *()> function )  [private]
```

Create the tables in the database.

Parameters

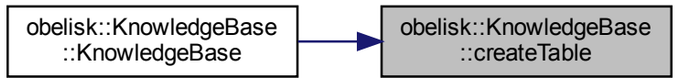
in	*function*	This function is called to create the table.

Definition at line 65 of file knowledge_base.cpp.

```
00066 {
00067     char* errmsg;
00068     int result = sqlite3_exec(dbConnection_, function(), NULL, NULL, &errmsg);
00069     if (result != SQLITE_OK)
00070     {
00071         if (errmsg)
00072         {
00073             throw obelisk::KnowledgeBaseException(errmsg);
00074         }
00075         else
00076         {
00077             throw obelisk::KnowledgeBaseException();
00078         }
00079     }
00080 }
```

Referenced by KnowledgeBase().

Here is the caller graph for this function:



5.14.3.9 enableForeignKeys()

```
void obelisk::KnowledgeBase::enableForeignKeys ( )  [private]
```

Enable foreign key functionality in the open database.

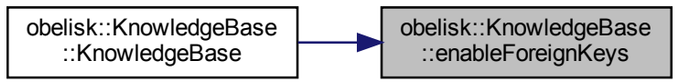
This must always be done when the connection is opened or it will not enforce the foreign key constraints.

Definition at line 44 of file knowledge_base.cpp.

```
00045 {
00046     char* errmsg;
00047     int result = sqlite3_exec(dbConnection_,
00048         "PRAGMA foreign_keys = ON;",
00049         NULL,
00050         NULL,
00051         &errmsg);
00052     if (result != SQLITE_OK)
00053     {
00054         if (errmsg)
00055         {
00056             throw obelisk::KnowledgeBaseException(errmsg);
00057         }
00058         else
00059         {
00060             throw obelisk::KnowledgeBaseException();
00061         }
00062     }
00063 }
```

Referenced by KnowledgeBase().

Here is the caller graph for this function:



5.14.3.10 getAction()

```
void obelisk::KnowledgeBase::getAction (
            obelisk::Action & action )
```

Get an Action based on the ID it contains.

Parameters

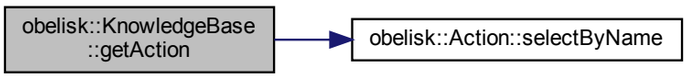
in	*action*	The Action object should contain just the ID and the rest will be filled in.

Definition at line 219 of file knowledge_base.cpp.

```
00220 {
00221     action.selectByName(dbConnection_);
00222 }
```

References obelisk::Action::selectByName().

Here is the call graph for this function:



5.14.3.11 getDouble()

```
void obelisk::KnowledgeBase::getDouble (
            double & result,
            float var1,
            float var2 )
```

Combines 2 separated floats back into a double.

This will recombine the separated floats from the getFloat method.

Parameters

out	*result*	The double generated from the combined floats.
in	*var1*	The first float to combine.
in	*var2*	The second float to combine.

Definition at line 282 of file knowledge_base.cpp.

```
00283 {
00284     result = (double) ((double) var2 + (double) var1);
00285 }
```

5.14.3.12 getEntity()

```
void obelisk::KnowledgeBase::getEntity (
            obelisk::Entity & entity )
```

Get an Entity object based on the ID it contains.

Parameters

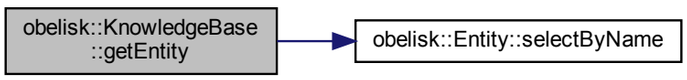
in,out	*entity*	The Entity object should contain just the ID and the rest will be filled in.

Definition at line 209 of file knowledge_base.cpp.

```
00210 {
00211     entity.selectByName(dbConnection_);
00212 }
```

References obelisk::Entity::selectByName().

Here is the call graph for this function:



5.14.3.13 getFact()

```
void obelisk::KnowledgeBase::getFact (
            obelisk::Fact & fact )
```

Get a Fact object based on the ID it contains.

Parameters

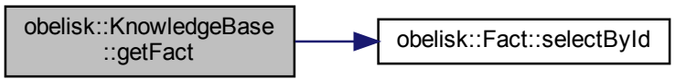
in,out	*fact*	The Fact object should contain just the ID and the rest will be filled in.

Definition at line 224 of file knowledge_base.cpp.

```
00225 {
00226     fact.selectById(dbConnection_);
00227 }
```

References obelisk::Fact::selectById().

Here is the call graph for this function:



5.14.3.14 getFloat()

```
void obelisk::KnowledgeBase::getFloat (
            float & result1,
            float & result2,
            double var )
```

Take a float and divide it into 2 floats.

This is useful to store doubles in SQLite since SQLite doesn't have a double type. Instead just store the 2 floats in the database. Then after selecting them combine them.

Parameters

out	*result1*	The first float generated from the double.
out	*result2*	The second float generated from the double.
in	*var*	The double to split into the 2 floats.

Definition at line 274 of file knowledge_base.cpp.

```
00277 {
00278     result1 = (float) var;
00279     result2 = (float) (var - (double) result1);
00280 }
```

5.14.3.15 getRule()

```
void obelisk::KnowledgeBase::getRule (
            obelisk::Rule & rule )
```

Get a Rule based on the ID it contains.

Parameters

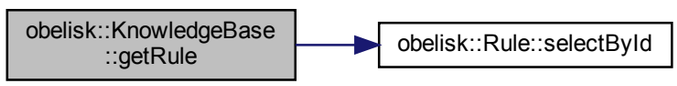
in,out	*rule*	The Rule object should contain just the ID and the rest will be filled in.

Definition at line 235 of file knowledge_base.cpp.

```
00236 {
00237     rule.selectById(dbConnection_);
00238 }
```

References obelisk::Rule::selectById().

Here is the call graph for this function:



5.14.3.16 getSuggestAction()

```
void obelisk::KnowledgeBase::getSuggestAction (
            obelisk::SuggestAction & suggestAction )
```

Get a SuggestAction based on the ID it contains.

Parameters

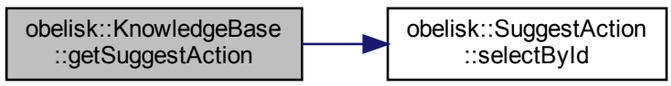
in,out	*suggestAction*	The SuggestAction object should contain just the ID and the rest will be filled in.

Definition at line 229 of file knowledge_base.cpp.

```
00231 {
00232     suggestAction.selectById(dbConnection_);
00233 }
```

References obelisk::SuggestAction::selectById().

Here is the call graph for this function:



5.14.3.17 getVerb()

```
void obelisk::KnowledgeBase::getVerb (
            obelisk::Verb & verb )
```

Get a Verb object based on the ID it contains.

Parameters

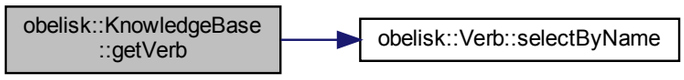
in,out	*verb*	The Verb object should contain just the ID and the rest will be filled in.

Definition at line 214 of file knowledge_base.cpp.

```
00215 {
00216     verb.selectByName(dbConnection_);
00217 }
```

References obelisk::Verb::selectByName().

Here is the call graph for this function:



5.14.3.18 queryFact()

```
void obelisk::KnowledgeBase::queryFact (
            obelisk::Fact & fact )
```

Query the KnowledgeBase to see if a Fact is true or false.

Parameters

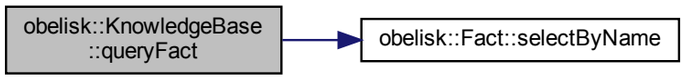
in	*fact*	The Fact to check.

Definition at line 263 of file knowledge_base.cpp.

```
00264 {
00265     fact.selectByName(dbConnection_);
00266 }
```

References obelisk::Fact::selectByName().

Here is the call graph for this function:



5.14.3.19 querySuggestAction()

```
void obelisk::KnowledgeBase::querySuggestAction (
            obelisk::Fact & fact,
            obelisk::Action & action )
```

Query the KnowledgeBase to get a suggested action based on a Fact. If a SuggestAction doesn't exist, it will return an empty Action.

Parameters

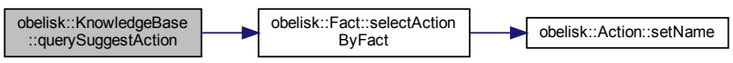
in	*fact*	The Fact to search for.
out	*action*	The Action that is suggested to take.

Definition at line 268 of file knowledge_base.cpp.

```
00270 {
00271     fact.selectActionByFact(dbConnection_, action);
00272 }
```

References obelisk::Fact::selectActionByFact().

Here is the call graph for this function:



5.14.3.20 updateIsTrue()

```
void obelisk::KnowledgeBase::updateIsTrue (
            obelisk::Fact & fact )
```

Update the is true field in the KnowledgeBase.

Parameters

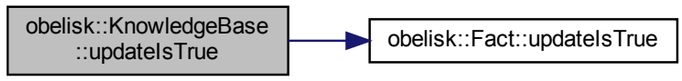
in,out	*fact*	The fact to update.

Definition at line 258 of file knowledge_base.cpp.

```
00259 {
00260     fact.updateIsTrue(dbConnection_);
00261 }
```

References obelisk::Fact::updateIsTrue().

Here is the call graph for this function:



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

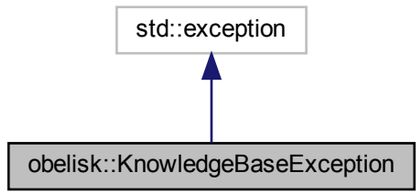
- src/lib/include/knowledge_base.h
- src/lib/knowledge_base.cpp

5.15 obelisk::KnowledgeBaseException Class Reference

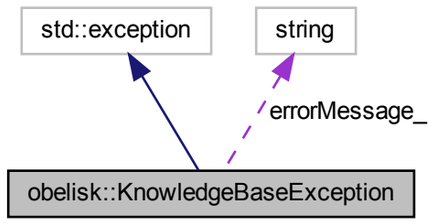
Exception thrown by the KnowledgeBase.

`#include <knowledge_base.h>`

Inheritance diagram for obelisk::KnowledgeBaseException:



Collaboration diagram for obelisk::KnowledgeBaseException:



Public Member Functions

- KnowledgeBaseException ()

  Construct a new KnowledgeBaseException object.
- KnowledgeBaseException (const std::string &errorMessage)

  Construct a new KnowledgeBaseException object.
- const char * what () const noexcept

  Get the error message that occurred.

Private Attributes

- const std::string errorMessage_

  The error message given.

5.15.1 Detailed Description

Exception thrown by the KnowledgeBase.

Definition at line 251 of file knowledge_base.h.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 KnowledgeBaseException()

```
obelisk::KnowledgeBaseException::KnowledgeBaseException (
            const std::string & errorMessage )  [inline]
```

Construct a new KnowledgeBaseException object.

Parameters

in	*errorMessage*	The error message given when thrown.

Definition at line 275 of file knowledge_base.h.

```
00275                                                                    :
00276                 errorMessage_(errorMessage)
00277             {
00278             }
```

5.15.3 Member Function Documentation

5.15.3.1 what()

```
const char* obelisk::KnowledgeBaseException::what ( ) const  [inline], [noexcept]
```

Get the error message that occurred.

Returns

const char* Returns the error message.

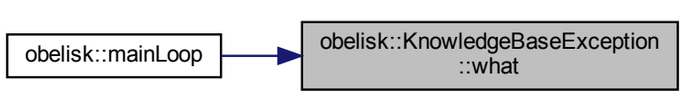
Definition at line 285 of file knowledge_base.h.

```
00286             {
00287                 return errorMessage_.c_str();
00288             }
```

References errorMessage_.

Referenced by obelisk::mainLoop().

Here is the caller graph for this function:



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

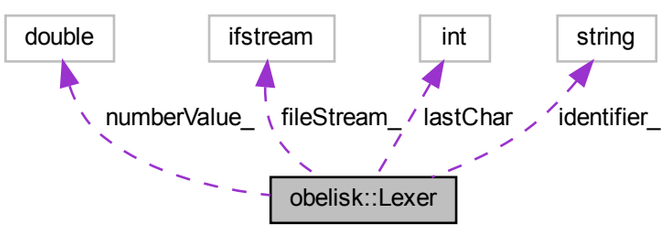
- src/lib/include/knowledge_base.h

5.16 obelisk::Lexer Class Reference

The Lexer reads and identifies tokens in the obelisk source code.

`#include <lexer.h>`

Collaboration diagram for obelisk::Lexer:



Public Types

- enum Token {
  kTokenEof = -1 , kTokenFact = -2 , kTokenRule = -3 , kTokenAction = -4 ,
  kTokenDef = -5 , kTokenExtern = -6 , kTokenIdentifier = -7 , kTokenNumber = -8 ,
  kTokenString = -9 }

  These token represent recognized language keywords and language functionality.

Public Member Functions

- Lexer (const std::string &sourceFile)

  Construct a new Lexer object.
- ~Lexer ()

  Destroy the Lexer object.
- int getToken ()

  Gets the next token in the source code.
- const std::string & getIdentifier ()

  Get the last identifier.
- double getNumberValue ()

  Get the last number value.

Private Member Functions

- void setIdentifier (const std::string &identifier)

  Set the identifier.
- void eraseIdentifier ()

  Erase the last identifier.
- void appendIdentifier (int lastChar)

  Add the last found char to the end of the identifier.
- void setNumberValue (double numberValue)

  Set the number value.
- void commentLine (int *lastChar)

  Comment the rest of the line.

Private Attributes

- int **lastChar** = ' '
- std::ifstream fileStream_

  The stream of the source file being read.
- std::string identifier_

  The last found identifier.
- double numberValue_ = 0

  The last found number.

5.16.1 Detailed Description

The Lexer reads and identifies tokens in the obelisk source code.

Definition at line 13 of file lexer.h.

5.16.2 Member Enumeration Documentation

5.16.2.1 Token

enum obelisk::Lexer::Token

These token represent recognized language keywords and language functionality.

Enumerator

kTokenEof	End of file is returned when the source code is finished.
kTokenFact	A fact which is a relationship between 2 entities.
kTokenRule	A rule which is a relationship between a new fact a existing fact.
kTokenAction	An action to take if a fact is true.
kTokenDef	A definition of a new function.
kTokenExtern	An external function that will be linked to.
kTokenIdentifier	An identifier which is a alphanumeric value.
kTokenNumber	A double floating point value.
kTokenString	A string.

Definition at line 70 of file lexer.h.

```
00071             {
00077                 kTokenEof = -1,
00078
00083                 kTokenFact = -2,
00084
00090                 kTokenRule = -3,
00091
00096                 kTokenAction = -4,
00097
00102                 kTokenDef = -5,
00103
00108                 kTokenExtern = -6,
00109
00114                 kTokenIdentifier = -7,
00115
00120                 kTokenNumber = -8,
00121
00126                 kTokenString = -9
00127             };
```

5.16.3 Constructor & Destructor Documentation

5.16.3.1 Lexer()

```
obelisk::Lexer::Lexer (
            const std::string & sourceFile )
```

Construct a new Lexer object.

Parameters

in	*sourceFile*	The source file to read.

Definition at line 5 of file lexer.cpp.

```
00006 {
00007     fileStream_.open(sourceFile, std::ifstream::in);
00008     if (!fileStream_)
00009     {
00010         throw obelisk::LexerException(
00011             "could not open source file " + sourceFile);
00012     }
00013 }
```

References fileStream_.

5.16.4 Member Function Documentation

5.16.4.1 appendIdentifier()

```
void obelisk::Lexer::appendIdentifier (
            int lastChar )  [private]
```

Add the last found char to the end of the identifier.

Parameters

in	*lastChar*	The last char that was found.

Definition at line 146 of file lexer.cpp.

```
00147 {
00148     identifier_ += lastChar;
00149 }
```

5.16.4.2 commentLine()

```
void obelisk::Lexer::commentLine (
            int * lastChar )  [private]
```

Comment the rest of the line.

Parameters

in	*lastChar*	The char to check to see if it in the end of the line.

Definition at line 122 of file lexer.cpp.

```
00123 {
00124     do
00125     {
00126         *lastChar = fileStream_.get();
00127     }
00128     while (*lastChar != EOF && *lastChar != '\n' && *lastChar != '\r');
00129 }
```

5.16.4.3 getIdentifier()

```
const std::string & obelisk::Lexer::getIdentifier ( )
```

Get the last identifier.

Returns

const std::string& Returns a string that contains the last found identifier.

Definition at line 131 of file lexer.cpp.

```
00132 {
00133     return identifier_;
00134 }
```

5.16.4.4 getNumberValue()

```
double obelisk::Lexer::getNumberValue ( )
```

Get the last number value.

Returns

double Return the last number that was found.

Definition at line 151 of file lexer.cpp.

```
00152 {
00153     return numberValue_;
00154 }
```

5.16.4.5 getToken()

```
int obelisk::Lexer::getToken ( )
```

Gets the next token in the source code.

Exceptions

LexerException	when an invalid token is found.

Returns

int Returns a Token value or char if no known token was found.

Definition at line 21 of file lexer.cpp.

```
00022 {
00023     while (isspace(lastChar))
00024     {
00025         lastChar = fileStream_.get();
00026     }
00027
00028     if (isalpha(lastChar))
00029     {
00030         eraseIdentifier();
00031         appendIdentifier(lastChar);
00032         while (isalnum((lastChar = fileStream_.get())))
00033         {
00034             appendIdentifier(lastChar);
00035         }
00036
00037         if (getIdentifier() == "fact")
00038         {
00039             return Token::kTokenFact;
00040         }
00041
00042         if (getIdentifier() == "rule")
00043         {
00044             return Token::kTokenRule;
00045         }
00046
00047         if (getIdentifier() == "action")
00048         {
00049             return Token::kTokenAction;
00050         }
00051
00052         if (getIdentifier() == "def")
00053         {
00054             return Token::kTokenDef;
00055         }
00056
00057         if (getIdentifier() == "extern")
```

```
00058         {
00059             return Token::kTokenExtern;
00060         }
00061
00062         return Token::kTokenIdentifier;
00063     }
00064
00065     if (isdigit(lastChar))
00066     {
00067         bool firstPeriod = false;
00068         std::string numberStr;
00069         do
00070         {
00071             if (firstPeriod && lastChar == '.')
00072             {
00073                 throw obelisk::LexerException("invalid double value");
00074             }
00075             else if (!firstPeriod && lastChar == '.')
00076             {
00077                 firstPeriod = true;
00078             }
00079             numberStr += lastChar;
00080             lastChar = fileStream_.get();
00081         }
00082         while (isdigit(lastChar) || lastChar == '.');
00083
00084         setNumberValue(strtod(numberStr.c_str(), nullptr));
00085
00086         return kTokenNumber;
00087     }
00088
00089     if (lastChar == '#')
00090     {
00091         commentLine(&lastChar);
00092
00093         if (lastChar != EOF)
00094         {
00095             return getToken();
00096         }
00097     }
00098     else if (lastChar == '/')
00099     {
00100         lastChar = fileStream_.get();
00101         if (lastChar == '/')
00102         {
00103             commentLine(&lastChar);
00104
00105             if (lastChar != EOF)
00106             {
00107                 return getToken();
00108             }
00109         }
00110     }
00111
00112     if (lastChar == EOF)
00113     {
00114         return kTokenEof;
00115     }
00116
00117     int thisChar = lastChar;
00118     lastChar     = fileStream_.get();
00119     return thisChar;
00120 }
```

5.16.4.6 setIdentifier()

```
void obelisk::Lexer::setIdentifier (
            const std::string & identifier )  [private]
```

Set the identifier.

Parameters

in	*identifier*	The new identifier.

Definition at line 136 of file lexer.cpp.

```
00137 {
00138     identifier_ = identifier;
00139 }
```

5.16.4.7 setNumberValue()

```
void obelisk::Lexer::setNumberValue (
            double numberValue )  [private]
```

Set the number value.

Parameters

in	*numberValue*	The new number value.

Definition at line 156 of file lexer.cpp.

```
00157 {
00158     numberValue_ = numberValue;
00159 }
```

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

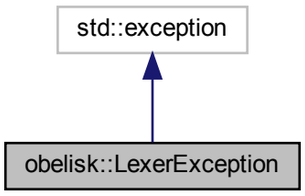
- src/lexer.h
- src/lexer.cpp

5.17 obelisk::LexerException Class Reference

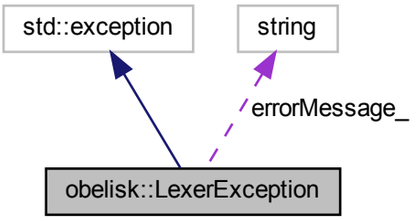
Lexer exception class.

`#include <lexer.h>`

Inheritance diagram for obelisk::LexerException:



Collaboration diagram for obelisk::LexerException:



Public Member Functions

- LexerException ()

  Construct a new LexerException object.
- LexerException (const std::string &errorMessage)

  Construct a new LexerException object.
- const char * what () const noexcept

  Return the exception's error message.

Private Attributes

- const std::string errorMessage_

  The error message from the exception.

5.17.1 Detailed Description

Lexer exception class.

Definition at line 171 of file lexer.h.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 LexerException()

```
obelisk::LexerException::LexerException (
            const std::string & errorMessage )  [inline]
```

Construct a new LexerException object.

Parameters

in	*errorMessage*	Error message to include.

Definition at line 195 of file lexer.h.

```
00195                                                   :
00196                 errorMessage_(errorMessage)
00197             {
00198             }
```

5.17.3 Member Function Documentation

5.17.3.1 what()

```
const char* obelisk::LexerException::what ( ) const  [inline], [noexcept]
```

Return the exception's error message.

Returns

const char* Returns a string containing the error message.

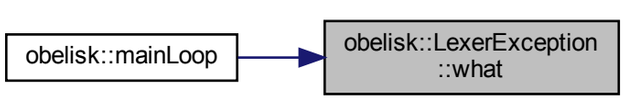
Definition at line 206 of file lexer.h.

```
00207             {
00208                 return errorMessage_.c_str();
00209             }
```

References errorMessage_.

Referenced by obelisk::mainLoop().

Here is the caller graph for this function:



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

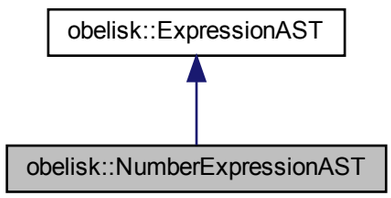
- src/lexer.h

5.18 obelisk::NumberExpressionAST Class Reference

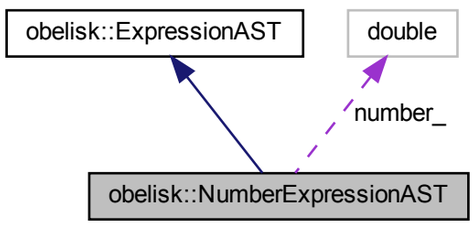
A number expression AST node.

`#include <number_expression_ast.h>`

Inheritance diagram for obelisk::NumberExpressionAST:



Collaboration diagram for obelisk::NumberExpressionAST:



Public Member Functions

- NumberExpressionAST (double number)
  Construct a new NumberExpressionAST object.
- llvm::Value * codegen () override
  Generate LLVM IR code for the number.

Private Member Functions

- double getNumber ()
  Get the number.
- void setNumber (double number)
  Set the number.

Private Attributes

- double number_
  The number.

5.18.1 Detailed Description

A number expression AST node.

Definition at line 12 of file number_expression_ast.h.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 NumberExpressionAST()

```
obelisk::NumberExpressionAST::NumberExpressionAST (
            double number )  [inline]
```

Construct a new NumberExpressionAST object.

Parameters

in	number	The number.

Definition at line 41 of file number_expression_ast.h.

```
00041                                                 :
00042                 number_(number)
00043             {
00044             }
```

5.18.3 Member Function Documentation

5.18.3.1 codegen()

```
llvm::Value * obelisk::NumberExpressionAST::codegen ( )  [override], [virtual]
```

Generate LLVM IR code for the number.

Returns

llvm::Value* Returns the genrated IR code.

Implements obelisk::ExpressionAST.

Definition at line 4 of file number_expression_ast.cpp.

```
00005 {
00006     return llvm::ConstantFP::get(*TheContext, llvm::APFloat(number_));
00007 }
```

References number_, and obelisk::TheContext.

5.18.3.2 getNumber()

```
double obelisk::NumberExpressionAST::getNumber ( )  [private]
```

Get the number.

Returns

double Returns the number.

5.18.3.3 setNumber()

```
void obelisk::NumberExpressionAST::setNumber (
            double number )  [private]
```

Set the number.

Parameters

in	*number*	The number.

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

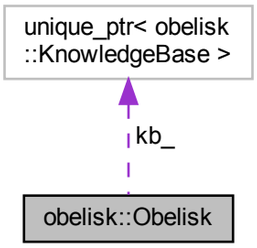
- src/ast/number_expression_ast.h
- src/ast/number_expression_ast.cpp

5.19 obelisk::Obelisk Class Reference

The obelisk library provides everything needed to consult the KnowledgeBase.

`#include <obelisk.h>`

Collaboration diagram for obelisk::Obelisk:



Public Member Functions

- Obelisk (std::string filename)
  Construct a new Obelisk object.
- ~Obelisk ()=default
  Destroy the Obelisk object.
- std::string getVersion ()
  Get the obelisk version.
- int getLibVersion ()
  Get the obelisk library so version.
- double query (const std::string &leftEntity, const std::string &verb, const std::string &rightEntity)
  Query the obelisk KnowledgeBase to see if a Fact is true or not.
- std::string queryAction (const std::string &leftEntity, const std::string &verb, const std::string &rightEntity)
  Query the Obelisk KnowledgeBase and return the suggested action to take.

Private Attributes

- std::unique_ptr< obelisk::KnowledgeBase > **kb_**

5.19.1 Detailed Description

The obelisk library provides everything needed to consult the KnowledgeBase.

Definition at line 21 of file obelisk.h.

5.19.2 Member Function Documentation

5.19.2.1 getLibVersion()

```
int obelisk::Obelisk::getLibVersion ( )
```

Get the obelisk library so version.

Returns

int The version.

Definition at line 15 of file obelisk.cpp.

```
00016 {
00017     return obelisk::soVersion;
00018 }
```

5.19.2.2 getVersion()

```
std::string obelisk::Obelisk::getVersion ( )
```

Get the obelisk version.

Returns

std::string The version.

Definition at line 10 of file obelisk.cpp.

```
00011 {
00012     return obelisk::version;
00013 }
```

5.19.2.3 query()

```
double obelisk::Obelisk::query (
            const std::string & leftEntity,
            const std::string & verb,
            const std::string & rightEntity )
```

Query the obelisk KnowledgeBase to see if a Fact is true or not.

Parameters

in	*p_obelisk*	The obelisk object pointer.
in	*left_entity*	The left entity.
in	*verb*	The verb.
in	*right_entity*	The right entity.

Returns

double Returns whether or not the Fact is true.

Definition at line 20 of file obelisk.cpp.

```
00023 {
00024     obelisk::Fact fact = obelisk::Fact(obelisk::Entity(leftEntity),
00025         obelisk::Entity(rightEntity),
00026         obelisk::Verb(verb));
00027
00028     kb_->queryFact(fact);
```

```
00029
00030     return fact.getIsTrue();
00031 }
```

References obelisk::Fact::getIsTrue().

Here is the call graph for this function:

5.19.2.4 queryAction()

```
std::string obelisk::Obelisk::queryAction (
             const std::string & leftEntity,
             const std::string & verb,
             const std::string & rightEntity )
```

Query the Obelisk KnowledgeBase and return the suggested action to take.

Parameters

in	*leftEntity*	The left entity.
in	*verb*	The verb.
in	*rightEntity*	The right entity.

Returns

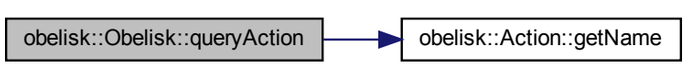
std::string Returns the suggested action.

Definition at line 33 of file obelisk.cpp.

```
00036 {
00037     obelisk::Fact fact = obelisk::Fact(obelisk::Entity(leftEntity),
00038         obelisk::Entity(rightEntity),
00039         obelisk::Verb(verb));
00040
00041     kb_->queryFact(fact);
00042
00043     obelisk::Action action;
00044     kb_->querySuggestAction(fact, action);
00045
00046     return action.getName();
00047 }
```

References obelisk::Action::getName().

Here is the call graph for this function:



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

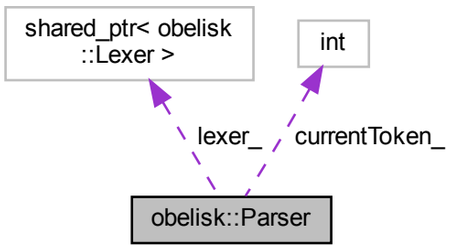
- src/lib/include/obelisk.h
- src/lib/obelisk.cpp

5.20 obelisk::Parser Class Reference

The Parser is responsible for analyzing the language's key words and taking action based on its analysis.

`#include <parser.h>`

Collaboration diagram for obelisk::Parser:



Public Member Functions

- Parser (std::shared_ptr< obelisk::Lexer > lexer)
  Construct a new Parser object.
- std::shared_ptr< obelisk::Lexer > getLexer ()
  Get the Lexer.
- void setLexer (std::shared_ptr< obelisk::Lexer > lexer)
  Set the Lexer to use during the parsing phase.
- int getCurrentToken ()
  Gets the current token held inside the Lexer.
- int getNextToken ()
  Instructs the Lexer to retrieve a new token.
- void handleAction (std::unique_ptr< obelisk::KnowledgeBase > &kb)
  Parse the SuggestAction and then insert it into the KnowledgeBase.
- void handleRule (std::unique_ptr< obelisk::KnowledgeBase > &kb)
  Parse the Rule and then insert it into the KnowledgeBase.
- void handleFact (std::unique_ptr< obelisk::KnowledgeBase > &kb)
  Parse the Fact and then insert it into the KnowledgeBase.
- void insertEntity (std::unique_ptr< obelisk::KnowledgeBase > &kb, obelisk::Entity &entity)
  Helper used to insert an Entity into the KnowledgeBase.
- void insertVerb (std::unique_ptr< obelisk::KnowledgeBase > &kb, obelisk::Verb &verb)
  Helper used to insert a Verb into the KnowledgeBase.
- void insertAction (std::unique_ptr< obelisk::KnowledgeBase > &kb, obelisk::Action &action)
  Helper used to insert an Action into the KnowledgeBase.
- void insertFact (std::unique_ptr< obelisk::KnowledgeBase > &kb, obelisk::Fact &fact, bool updateIsTrue=false)
  Helper used to insert a Fact into the KnowledgeBase.
- void insertSuggestAction (std::unique_ptr< obelisk::KnowledgeBase > &kb, obelisk::SuggestAction &suggest↩ Action)
  Helper used to insert a SuggestAction into the KnowledgeBase.
- void insertRule (std::unique_ptr< obelisk::KnowledgeBase > &kb, obelisk::Rule &rule)
  Helper usedto insert a Rule into the KnowledgeBase.

Private Member Functions

- void setCurrentToken (int currentToken)

  Set the current token.
- std::unique_ptr< obelisk::ExpressionAST > logError (const char *str)

  Log errors from the LLVM parsing.
- std::unique_ptr< obelisk::PrototypeAST > logErrorPrototype (const char *str)

  Log errors from the LLVM parsing involving the prototypes.
- std::unique_ptr< obelisk::ExpressionAST > parseExpression ()

  The AST expression parser.
- std::unique_ptr< obelisk::ExpressionAST > parseNumberExpression ()

  The AST number expression parser.
- std::unique_ptr< obelisk::ExpressionAST > parseParenthesisExpression ()

  The AST parenthesis expression parser.
- std::unique_ptr< obelisk::ExpressionAST > parseIdentifierExpression ()

  The AST identifier expression parser.
- std::unique_ptr< obelisk::ExpressionAST > parsePrimary ()

  The AST primary expression parser.
- std::unique_ptr< obelisk::PrototypeAST > parsePrototype ()

  The AST prototype parser.
- std::unique_ptr< obelisk::FunctionAST > parseDefinition ()

  The AST definition parser.
- std::unique_ptr< obelisk::FunctionAST > parseTopLevelExpression ()

  The AST top level expression parser.
- std::unique_ptr< obelisk::PrototypeAST > parseExtern ()

  The AST external definition parser.
- void parseAction (obelisk::SuggestAction &suggestAction)

  Parse a SuggestAction.
- void parseRule (obelisk::Rule &rule)

  Parse a Rule.
- void parseFact (std::vector< obelisk::Fact > &facts)

  Parse Facts.

Private Attributes

- std::shared_ptr< obelisk::Lexer > lexer_

  The Lexer object that the Parser is using to Parse a specific source file.
- int currentToken_ = 0

  The current token that the lexer has retrieved.

5.20.1 Detailed Description

The Parser is responsible for analyzing the language's key words and taking action based on its analysis.

Definition at line 25 of file parser.h.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 Parser()

```
obelisk::Parser::Parser (
            std::shared_ptr< obelisk::Lexer > lexer )  [inline]
```

Construct a new Parser object.

Parameters

in	*lexer*	The lexer the parser uses to retrieve parts of the language.

Definition at line 168 of file parser.h.

```
00168                                                   :
00169                 lexer_(lexer)
00170             {
00171             }
```

5.20.3 Member Function Documentation

5.20.3.1 getCurrentToken()

```
int obelisk::Parser::getCurrentToken ( )
```

Gets the current token held inside the Lexer.

Returns

int Returns the current token.

Definition at line 35 of file parser.cpp.

```
00036 {
00037     return currentToken_;
00038 }
```

5.20.3.2 getLexer()

```
std::shared_ptr< obelisk::Lexer > obelisk::Parser::getLexer ( )
```

Get the Lexer.

Returns

std::shared_ptr<obelisk::Lexer> Returns the current Lexer in use by the Parser.

Definition at line 11 of file parser.cpp.

```
00012 {
00013     return lexer_;
00014 }
```

References lexer_.

5.20.3.3 getNextToken()

```
int obelisk::Parser::getNextToken ( )
```

Instructs the Lexer to retrieve a new token.

Returns

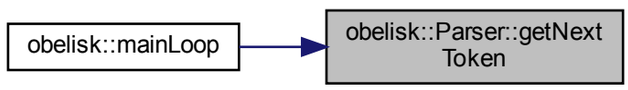
int Returns the next token.

Definition at line 22 of file parser.cpp.

```
00023 {
00024     try
00025     {
00026         setCurrentToken(getLexer()->getToken());
00027     }
00028     catch (obelisk::LexerException& exception)
00029     {
00030         throw;
00031     }
00032     return getCurrentToken();
00033 }
```

Referenced by obelisk::mainLoop().

Here is the caller graph for this function:



5.20.3.4 handleAction()

```
void obelisk::Parser::handleAction (
            std::unique_ptr< obelisk::KnowledgeBase > & kb )
```

Parse the SuggestAction and then insert it into the KnowledgeBase.

Parameters

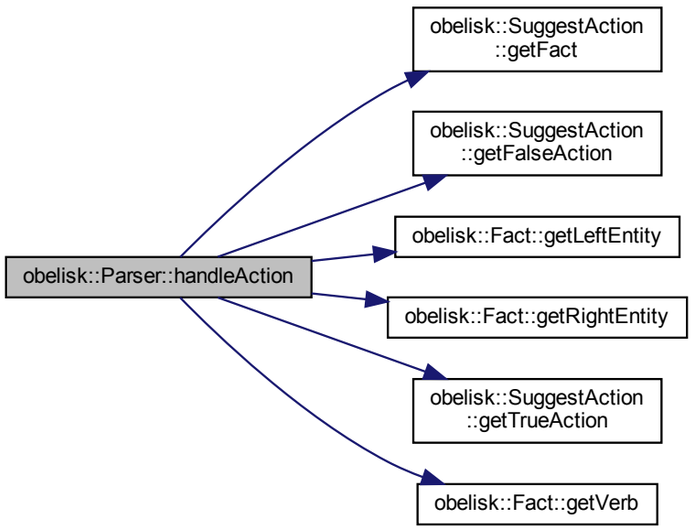
in	*kb*	The KnowledgeBase to insert the SuggestAction into.

Definition at line 774 of file parser.cpp.

```
00775 {
00776     obelisk::SuggestAction suggestAction;
00777
00778     try
00779     {
00780         parseAction(suggestAction);
00781         insertEntity(kb, suggestAction.getFact().getLeftEntity());
00782         insertEntity(kb, suggestAction.getFact().getRightEntity());
00783         insertVerb(kb, suggestAction.getFact().getVerb());
00784         insertFact(kb, suggestAction.getFact());
00785         insertAction(kb, suggestAction.getTrueAction());
00786         insertAction(kb, suggestAction.getFalseAction());
00787         insertSuggestAction(kb, suggestAction);
00788     }
00789     catch (obelisk::ParserException& exception)
00790     {
00791         throw;
00792     }
00793 }
```

References obelisk::SuggestAction::getFact(), obelisk::SuggestAction::getFalseAction(), obelisk::Fact::getLeftEntity(), obelisk::Fact::getRightEntity(), obelisk::SuggestAction::getTrueAction(), and obelisk::Fact::getVerb().

Here is the call graph for this function:



5.20.3.5 handleFact()

```
void obelisk::Parser::handleFact (
            std::unique_ptr< obelisk::KnowledgeBase > & kb )
```

Parse the Fact and then insert it into the KnowledgeBase.

Parameters

in	*kb*	The KnowledgeBase to insert the Fact into.

Definition at line 827 of file parser.cpp.

```
00828 {
00829     std::vector<obelisk::Fact> facts;
00830     try
00831     {
00832         parseFact(facts);
00833     }
00834     catch (obelisk::ParserException& exception)
00835     {
00836         throw;
00837     }
00838
00839     int verbId = 0;
00840     for (auto& fact : facts)
00841     {
00842         try
00843         {
00844             insertEntity(kb, fact.getLeftEntity());
00845             insertEntity(kb, fact.getRightEntity());
00846         }
00847         catch (obelisk::ParserException& exception)
00848         {
00849             throw;
00850         }
00851
00852         if (verbId == 0)
00853         {
00854             try
00855             {
00856                 insertVerb(kb, fact.getVerb());
00857             }
00858             catch (obelisk::ParserException& exception)
00859             {
00860                 throw;
00861             }
00862             verbId = fact.getVerb().getId();
00863         }
00864         else
00865         {
00866             fact.getVerb().setId(verbId);
```

```
00867         }
00868
00869         try
00870         {
00871             insertFact(kb, fact, true);
00872         }
00873         catch (obelisk::ParserException& exception)
00874         {
00875             throw;
00876         }
00877
00878         kb->checkRule(fact);
00879     }
00880 }
```

5.20.3.6 handleRule()

```
void obelisk::Parser::handleRule (
            std::unique_ptr< obelisk::KnowledgeBase > & kb )
```

Parse the Rule and then insert it into the KnowledgeBase.

Parameters

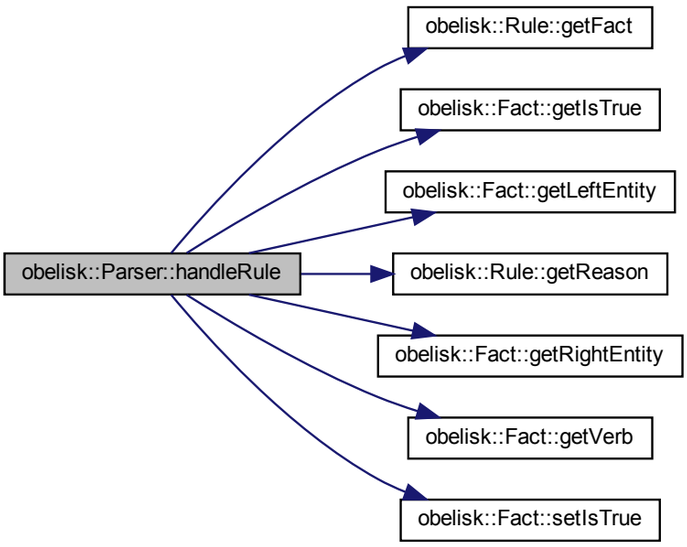
in	*kb*	The KnowledgeBase to insert the Rule into.

Definition at line 795 of file parser.cpp.

```
00796 {
00797     obelisk::Rule rule;
00798
00799     try
00800     {
00801         parseRule(rule);
00802
00803         insertEntity(kb, rule.getReason().getLeftEntity());
00804         insertEntity(kb, rule.getReason().getRightEntity());
00805         insertVerb(kb, rule.getReason().getVerb());
00806         insertFact(kb, rule.getReason());
00807
00808         // The rule is true, so the fact must be true to.
00809         if (rule.getReason().getIsTrue() > 0)
00810         {
00811             rule.getFact().setIsTrue(1.0);
00812         }
00813
00814         insertEntity(kb, rule.getFact().getLeftEntity());
00815         insertEntity(kb, rule.getFact().getRightEntity());
00816         insertVerb(kb, rule.getFact().getVerb());
00817         insertFact(kb, rule.getFact());
00818
00819         insertRule(kb, rule);
00820     }
00821     catch (obelisk::ParserException& exception)
00822     {
00823         throw;
00824     }
00825 }
```

References obelisk::Rule::getFact(), obelisk::Fact::getIsTrue(), obelisk::Fact::getLeftEntity(), obelisk::Rule::getReason(), obelisk::Fact::getRightEntity(), obelisk::Fact::getVerb(), and obelisk::Fact::setIsTrue().

Here is the call graph for this function:



5.20.3.7 insertAction()

```
void obelisk::Parser::insertAction (
            std::unique_ptr< obelisk::KnowledgeBase > & kb,
            obelisk::Action & action )
```

Helper used to insert an Action into the KnowledgeBase.

Parameters

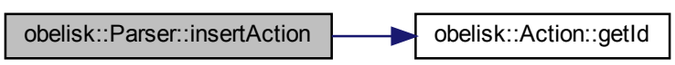
in	*kb*	The KnowledgeBase to use.
in,out	*action*	The Action to insert. It will contain the ID of the Action after inserting it.

Definition at line 920 of file parser.cpp.

```
00922 {
00923     std::vector<obelisk::Action> actions {action};
00924     kb->addActions(actions);
00925     action = std::move(actions.front());
00926
00927     // the id was not inserted, so check if it exists in the database
00928     if (action.getId() == 0)
00929     {
00930         kb->getAction(action);
00931         if (action.getId() == 0)
00932         {
00933             throw obelisk::ParserException(
00934                 "action could not be inserted into the database");
00935         }
00936     }
00937 }
```

References obelisk::Action::getId().

Here is the call graph for this function:



5.20.3.8 insertEntity()

```
void obelisk::Parser::insertEntity (
            std::unique_ptr< obelisk::KnowledgeBase > & kb,
            obelisk::Entity & entity )
```

Helper used to insert an Entity into the KnowledgeBase.

Parameters

in	*kb*	The KnowledgeBase to use.
in,out	*entity*	The Entity to insert. It will contain the ID of the Entity after inserting it.

Definition at line 882 of file parser.cpp.

```
00884 {
00885     std::vector<obelisk::Entity> entities {entity};
00886     kb->addEntities(entities);
00887     entity = std::move(entities.front());
00888
00889     // the id was not inserted, so check if it exists in the database
00890     if (entity.getId() == 0)
00891     {
00892         kb->getEntity(entity);
00893         if (entity.getId() == 0)
00894         {
00895             throw obelisk::ParserException(
00896                 "entity could not be inserted into the database");
00897         }
00898     }
00899 }
```

References obelisk::Entity::getId().

Here is the call graph for this function:

5.20.3.9 insertFact()

```
void obelisk::Parser::insertFact (
            std::unique_ptr< obelisk::KnowledgeBase > & kb,
            obelisk::Fact & fact,
            bool updateIsTrue = false )
```

Helper used to insert a Fact into the KnowledgeBase.

Parameters

in	*kb*	The KnowledgeBase to use.
in,out	*fact*	The Fact to insert. It will contain the ID of the Fact after inserting it.
in	*updateIsTrue*	If true, it will update the value of is_true in the KnowledgeBase if the Fact already exists.

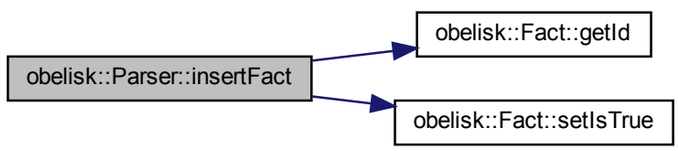
Definition at line 939 of file parser.cpp.

```
00942 {
00943     std::vector<obelisk::Fact> facts {fact};
00944     kb->addFacts(facts);
00945     fact = std::move(facts.front());
00946
00947     // the id was not inserted, so check if it exists in the database
```

```
00948     if (fact.getId() == 0)
00949     {
00950         kb->getFact(fact);
00951         if (fact.getId() == 0)
00952         {
00953             throw obelisk::ParserException(
00954                 "fact could not be inserted into the database");
00955         }
00956         else
00957         {
00958             if (updateIsTrue)
00959             {
00960                 fact.setIsTrue(1.0);
00961                 kb->updateIsTrue(fact);
00962             }
00963         }
00964     }
00965 }
```

References obelisk::Fact::getId(), and obelisk::Fact::setIsTrue().

Here is the call graph for this function:



5.20.3.10 insertRule()

```
void obelisk::Parser::insertRule (
            std::unique_ptr< obelisk::KnowledgeBase > & kb,
            obelisk::Rule & rule )
```

Helper usedto insert a Rule into the KnowledgeBase.

Parameters

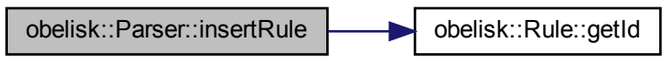
in	*kb*	The KnowledgeBase to use.
in,out	*rule*	The Rule to insert. It will contain the ID of the Rule after inserting it.

Definition at line 987 of file parser.cpp.

```
00989 {
00990     std::vector<obelisk::Rule> rules {rule};
00991     kb->addRules(rules);
00992     rule = std::move(rules.front());
00993
00994     // the id was not inserted, so check if it exists in the database
00995     if (rule.getId() == 0)
00996     {
00997         kb->getRule(rule);
00998         if (rule.getId() == 0)
00999         {
01000             throw obelisk::ParserException(
01001                 "rule could not be inserted into the database");
01002         }
01003     }
01004 }
```

References obelisk::Rule::getId().

Here is the call graph for this function:



5.20.3.11 insertSuggestAction()

```
void obelisk::Parser::insertSuggestAction (
            std::unique_ptr< obelisk::KnowledgeBase > & kb,
            obelisk::SuggestAction & suggestAction )
```

Helper used to insert a SuggestAction into the KnowledgeBase.

Parameters

in	*kb*	The KnowledgeBase to use.
in,out	*suggestAction*	The SuggestAction to insert. It will contain the ID of the SuggestAction after inserting it.

Definition at line 967 of file parser.cpp.

```
00970 {
00971     std::vector<obelisk::SuggestAction> suggestActions {suggestAction};
00972     kb->addSuggestActions(suggestActions);
00973     suggestAction = std::move(suggestActions.front());
00974
00975     // the id was not inserted, so check if it exists in the database
00976     if (suggestAction.getId() == 0)
00977     {
00978         kb->getSuggestAction(suggestAction);
00979         if (suggestAction.getId() == 0)
00980         {
00981             throw obelisk::ParserException(
00982                 "suggest_action could not be inserted into the database");
00983         }
00984     }
00985 }
```

References obelisk::SuggestAction::getId().

Here is the call graph for this function:

5.20.3.12 insertVerb()

```
void obelisk::Parser::insertVerb (
            std::unique_ptr< obelisk::KnowledgeBase > & kb,
            obelisk::Verb & verb )
```

Helper used to insert a Verb into the KnowledgeBase.

Parameters

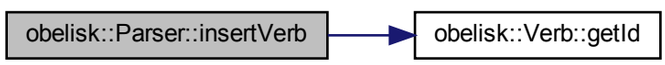
in	*kb*	The KnowledegeBase to use.
in,out	*verb*	The Verb to insert. It will contain the ID of the Verb after inserting it.

Definition at line 901 of file parser.cpp.

```
00903 {
00904     std::vector<obelisk::Verb> verbs {verb};
00905     kb->addVerbs(verbs);
00906     verb = std::move(verbs.front());
00907
00908     // the id was not inserted, so check if it exists in the database
00909     if (verb.getId() == 0)
00910     {
00911         kb->getVerb(verb);
00912         if (verb.getId() == 0)
00913         {
00914             throw obelisk::ParserException(
00915                 "verb could not be inserted into the database");
00916         }
00917     }
00918 }
```

References obelisk::Verb::getId().

Here is the call graph for this function:



5.20.3.13 logError()

```
std::unique_ptr< obelisk::ExpressionAST > obelisk::Parser::logError (
            const char * str )  [private]
```

Log errors from the LLVM parsing.

Parameters

in	*str*	The error message.

Returns

std::unique_ptr<obelisk::ExpressionAST> Returns the AST expression that caused the error.

Definition at line 45 of file parser.cpp.

```
00047 {
00048     fprintf(stderr, "Error: %s\n", str);
00049     return nullptr;
00050 }
```

5.20.3.14 logErrorPrototype()

```
std::unique_ptr< obelisk::PrototypeAST > obelisk::Parser::logErrorPrototype (
            const char * str )  [private]
```

Log errors from the LLVM parsing involving the prototypes.

Parameters

in	*str*	The error message.

Returns

std::unique_ptr<obelisk::PrototypeAST> Returns the AST for the prototype.

Definition at line 52 of file parser.cpp.

```
00054 {
00055     logError(str);
00056     return nullptr;
00057 }
```

5.20.3.15 parseAction()

```
void obelisk::Parser::parseAction (
            obelisk::SuggestAction & suggestAction )  [private]
```

Parse a SuggestAction.

Parameters

out	*suggestAction*	The parsed SuggestAction.

Definition at line 223 of file parser.cpp.

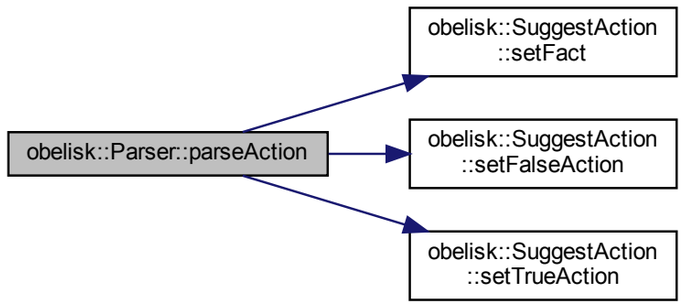
```
00224 {
00225     std::stack<char> syntax;
00226 
00227     getNextToken();
00228     if (getCurrentToken() != '(')
00229     {
00230         throw obelisk::ParserException(
00231             "expected '(' but got '" + std::to_string(getCurrentToken()) + "'");
00232     }
00233 
00234     syntax.push('(');
00235 
00236     getNextToken();
00237     if (getLexer()->getIdentifier() != "if")
00238     {
00239         throw obelisk::ParserException(
00240             "expected 'if' but got '" + getLexer()->getIdentifier() + "'");
00241     }
00242 
00243     bool getEntity {true};
00244     std::string leftEntity {""};
00245     std::string rightEntity {""};
00246     std::string trueAction {""};
00247     std::string falseAction {""};
00248     std::string entityName {""};
00249     std::string verb {""};
00250     getNextToken();
00251 
00252     // get the entity side of statement
00253     while (true)
00254     {
00255         if (getEntity)
00256         {
00257             if (getCurrentToken() == '"')
00258             {
00259                 if (syntax.top() != '"')
00260                 {
00261                     // open a double quote
00262                     syntax.push('"');
00263                     getNextToken();
00264                 }
00265                 else if (syntax.top() == '"')
00266                 {
00267                     // close a double quote
00268                     syntax.pop();
00269                     if (verb == "")
00270                     {
00271                         leftEntity = std::move(entityName);
00272                     }
00273                     else
00274                     {
00275                         rightEntity = std::move(entityName);
00276                     }
00277                     getEntity = false;
00278                     getNextToken();
00279                     continue;
00280                 }
00281             }
00282 
00283             if (syntax.top() == '"')
```

```
00284             {
00285                 if (entityName != "")
00286                 {
00287                     entityName += " ";
00288                 }
00289                 entityName += getLexer()->getIdentifier();
00290             }
00291             getNextToken();
00292         }
00293         else
00294         {
00295             if (getCurrentToken() == ')')
00296             {
00297                 throw obelisk::ParserException("unexpected ')'");
00298             }
00299 
00300             if (getCurrentToken() == '"')
00301             {
00302                 throw obelisk::ParserException("unexpected '\"'");
00303             }
00304 
00305             if (getLexer()->getIdentifier() == "then")
00306             {
00307                 break;
00308             }
00309             else
00310             {
00311                 verb = getLexer()->getIdentifier();
00312                 for (const auto& letter : verb)
00313                 {
00314                     if (!isalpha(letter))
00315                     {
00316                         throw new obelisk::ParserException(
00317                             "non alphabetic symbol in verb");
00318                     }
00319                 }
00320                 getEntity = true;
00321                 continue;
00322             }
00323         }
00324     }
00325 
00326     // get the action side of statement
00327     bool getAction {true};
00328     while (true)
00329     {
00330         if (getAction)
00331         {
00332             if (getCurrentToken() == '"')
00333             {
00334                 if (syntax.top() != '"')
00335                 {
00336                     // open a double quote
00337                     syntax.push('"');
00338                     getNextToken();
00339                 }
00340                 else if (syntax.top() == '"')
00341                 {
00342                     // close a double quote
00343                     syntax.pop();
00344                     if (trueAction == "")
00345                     {
00346                         trueAction = std::move(entityName);
00347                     }
00348                     else
00349                     {
00350                         falseAction = std::move(entityName);
00351                     }
00352                     getAction = false;
00353                     getNextToken();
00354                     continue;
00355                 }
00356             }
00357 
00358             if (syntax.top() == '"')
00359             {
00360                 if (entityName != "")
00361                 {
00362                     entityName += " ";
00363                 }
00364                 entityName += getLexer()->getIdentifier();
00365             }
00366             getNextToken();
00367         }
00368         else
00369         {
00370             if (getCurrentToken() == ')')
00371             {
00372                 // closing parenthesis found, make sure we have everything
00373                 // needed
00374                 if (syntax.top() != '(')
00375                 {
00376                     throw obelisk::ParserException("unexpected ')'");
00377                 }
00378                 else
00379                 {
00380                     syntax.pop();
00381                 }
00382 
00383                 if (leftEntity == "")
00384                 {
00385                     throw obelisk::ParserException("missing left entity");
00386                 }
00387 
00388                 if (rightEntity == "")
00389                 {
00390                     throw obelisk::ParserException("missing left entity");
00391                 }
```

```
00392
00393                 if (verb == "")
00394                 {
00395                     throw obelisk::ParserException("missing verb");
00396                 }
00397
00398                 if (trueAction == "")
00399                 {
00400                     throw obelisk::ParserException("missing true action");
00401                 }
00402
00403                 if (falseAction == "")
00404                 {
00405                     throw obelisk::ParserException("missing false action");
00406                 }
00407
00408                 getNextToken();
00409                 if (getCurrentToken() != ';')
00410                 {
00411                     throw obelisk::ParserException("missing ';'");
00412                 }
00413
00414                 break;
00415             }
00416
00417             if (getCurrentToken() == '"')
00418             {
00419                 throw obelisk::ParserException("unexpected '\"'");
00420                 break;
00421             }
00422
00423             if (getLexer()->getIdentifier() == "else")
00424             {
00425                 getNextToken();
00426                 getAction = true;
00427                 continue;
00428             }
00429             else
00430             {
00431                 getAction = true;
00432                 continue;
00433             }
00434         }
00435     }
00436
00437     suggestAction.setFact(obelisk::Fact(obelisk::Entity(leftEntity),
00438         obelisk::Entity(rightEntity),
00439         obelisk::Verb(verb)));
00440     suggestAction.setTrueAction(obelisk::Action(trueAction));
00441     suggestAction.setFalseAction(obelisk::Action(falseAction));
00442 }
```

References obelisk::SuggestAction::setFact(), obelisk::SuggestAction::setFalseAction(), and obelisk::SuggestAction::setTrueAction().

Here is the call graph for this function:



5.20.3.16 parseDefinition()

```
std::unique_ptr< obelisk::FunctionAST > obelisk::Parser::parseDefinition ( )  [private]
```

The AST definition parser.

Returns

std::unique_ptr<obelisk::FunctionAST> Returns the parsed AST definition expression.

Definition at line 186 of file parser.cpp.

```
00187 {
00188     getNextToken();
00189     auto prototype = parsePrototype();
00190     if (!prototype)
00191     {
00192         return nullptr;
00193     }
00194
00195     if (auto expression = parseExpression())
00196     {
00197         return std::make_unique<FunctionAST>(std::move(prototype),
00198             std::move(expression));
00199     }
00200
00201     return nullptr;
00202 }
```

5.20.3.17 parseExpression()

```
std::unique_ptr< obelisk::ExpressionAST > obelisk::Parser::parseExpression ( )  [private]
```

The AST expression parser.

Returns

std::unique_ptr<obelisk::ExpressionAST> Returns the parsed AST expression.

Definition at line 59 of file parser.cpp.

```
00060 {
00061     auto LHS = parsePrimary();
00062     if (!LHS)
00063     {
00064         return nullptr;
00065     }
00066
00067     return LHS;
00068 }
```

5.20.3.18 parseExtern()

```
std::unique_ptr< obelisk::PrototypeAST > obelisk::Parser::parseExtern ( )  [private]
```

The AST external definition parser.

Returns

std::unique_ptr<obelisk::PrototypeAST> Returns the parsed AST external definition.

Definition at line 217 of file parser.cpp.

```
00218 {
00219     getNextToken();
00220     return parsePrototype();
00221 }
```

5.20.3.19 parseFact()

```
void obelisk::Parser::parseFact (
            std::vector< obelisk::Fact > & facts )  [private]
```

Parse Facts.

Parameters

out	*facts*	The parsed Facts.

Definition at line 633 of file parser.cpp.

```
00634 {
00635     std::stack<char> syntax;
00636
00637     getNextToken();
00638     if (getCurrentToken() != '(')
00639     {
00640         throw obelisk::ParserException(
00641             "expected '(' but got '" + std::to_string(getCurrentToken()) + "'");
00642     }
00643
00644     syntax.push('(');
00645
00646     bool getEntity {true};
00647     std::vector<std::string> leftEntities;
00648     std::vector<std::string> rightEntities;
00649     std::string entityName {""};
00650     std::string verb {""};
00651     getNextToken();
00652     while (true)
00653     {
00654         if (getEntity)
00655         {
00656             if (getCurrentToken() == '"')
00657             {
00658                 if (syntax.top() != '"')
00659                 {
00660                     // open a double quote
00661                     syntax.push('"');
00662                     getNextToken();
00663                 }
00664                 else if (syntax.top() == '"')
00665                 {
00666                     // close a double quote
00667                     syntax.pop();
00668                     if (verb == "")
00669                     {
00670                         leftEntities.push_back(entityName);
00671                     }
00672                     else
00673                     {
00674                         rightEntities.push_back(entityName);
00675                     }
00676                     entityName = "";
00677                     getEntity  = false;
00678                     getNextToken();
00679                     continue;
00680                 }
00681             }
00682
00683             if (syntax.top() == '"')
00684             {
00685                 if (entityName != "")
00686                 {
00687                     entityName += " ";
00688                 }
00689                 entityName += getLexer()->getIdentifier();
00690             }
00691             getNextToken();
00692         }
00693         else
00694         {
00695             if (getCurrentToken() == ')')
00696             {
00697                 // closing parenthesis found, make sure we have everything
00698                 // needed
00699                 if (syntax.top() != '(')
00700                 {
00701                     throw obelisk::ParserException("unexpected ')'");
00702                 }
00703                 else
00704                 {
00705                     syntax.pop();
00706                 }
00707
00708                 if (verb == "")
00709                 {
00710                     throw obelisk::ParserException("verb is empty");
00711                 }
00712
00713                 if (leftEntities.size() == 0)
00714                 {
00715                     throw obelisk::ParserException(
00716                         "missing left side entities");
00717                 }
00718
00719                 if (rightEntities.size() == 0)
00720                 {
00721                     throw obelisk::ParserException(
00722                         "missing right side entities");
00723                 }
00724
00725                 getNextToken();
00726                 if (getCurrentToken() != ';')
00727                 {
00728                     throw obelisk::ParserException("missing ';'");
00729                 }
00730
00731                 break;
00732             }
```

```
00733
00734             if (getCurrentToken() == '"')
00735             {
00736                 throw obelisk::ParserException("unexpected '\"'");
00737             }
00738
00739             if (getLexer()->getIdentifier() == "and")
00740             {
00741                 getNextToken();
00742                 getEntity = true;
00743                 continue;
00744             }
00745             else
00746             {
00747                 verb = getLexer()->getIdentifier();
00748                 for (const auto& letter : verb)
00749                 {
00750                     if (!isalpha(letter))
00751                     {
00752                         throw new obelisk::ParserException(
00753                             "non alphabetic symbol in verb");
00754                     }
00755                 }
00756                 getEntity = true;
00757                 continue;
00758             }
00759         }
00760     }
00761
00762     for (auto& leftEntity : leftEntities)
00763     {
00764         for (auto& rightEntity : rightEntities)
00765         {
00766             facts.push_back(obelisk::Fact(obelisk::Entity(leftEntity),
00767                 obelisk::Entity(rightEntity),
00768                 obelisk::Verb(verb),
00769                 true));
00770         }
00771     }
00772 }
```

5.20.3.20 parseIdentifierExpression()

`std::unique_ptr< obelisk::ExpressionAST > obelisk::Parser::parseIdentifierExpression ( )  [private]`

The AST identifier expression parser.

Returns

std::unique_ptr<obelisk::ExpressionAST> Returns the parsed AST expression.

Definition at line 112 of file parser.cpp.

```
00113 {
00114     std::string idName = getLexer()->getIdentifier();
00115     getNextToken();
00116     if (getCurrentToken() != '(')
00117     {
00118         return std::make_unique<obelisk::VariableExpressionAST>(idName);
00119     }
00120
00121     getNextToken();
00122     std::vector<std::unique_ptr<obelisk::ExpressionAST» args;
00123     if (getCurrentToken() != ')')
00124     {
00125         while (true)
00126         {
00127             if (auto arg = parseExpression())
00128             {
00129                 args.push_back(std::move(arg));
00130             }
00131             else
00132             {
00133                 return nullptr;
00134             }
00135
00136             if (getCurrentToken() == ')')
00137             {
00138                 break;
00139             }
00140
00141             if (getCurrentToken() != ',')
00142             {
00143                 return logError("Expected ')' or ',' in argument list");
00144             }
00145
00146             getNextToken();
00147         }
00148     }
00149
00150     getNextToken();
00151     return std::make_unique<CallExpressionAST>(idName, std::move(args));
00152 }
```

5.20.3.21 parseNumberExpression()

```
std::unique_ptr< obelisk::ExpressionAST > obelisk::Parser::parseNumberExpression ( )  [private]
```

The AST number expression parser.

Returns

std::unique_ptr<obelisk::ExpressionAST> Returns the parsed AST expression.

Definition at line 85 of file parser.cpp.

```
00086 {
00087     auto result = std::make_unique<obelisk::NumberExpressionAST>(
00088         getLexer()->getNumberValue());
00089     getNextToken();
00090     return result;
00091 }
```

5.20.3.22 parseParenthesisExpression()

```
std::unique_ptr< obelisk::ExpressionAST > obelisk::Parser::parseParenthesisExpression ( )  [private]
```

The AST parenthesis expression parser.

Returns

std::unique_ptr<obelisk::ExpressionAST> Returns the parsed AST expression.

Definition at line 94 of file parser.cpp.

```
00095 {
00096     getNextToken();
00097     auto v = parseExpression();
00098     if (!v)
00099     {
00100         return nullptr;
00101     }
00102
00103     if (getCurrentToken() != ')')
00104     {
00105         return logError("expected ')'");
00106     }
00107     getNextToken();
00108     return v;
00109 }
```

5.20.3.23 parsePrimary()

```
std::unique_ptr< obelisk::ExpressionAST > obelisk::Parser::parsePrimary ( )  [private]
```

The AST primary expression parser.

Returns

std::unique_ptr<obelisk::ExpressionAST> Returns the parsed AST expression.

Definition at line 70 of file parser.cpp.

```
00071 {
00072     switch (getCurrentToken())
00073     {
00074         case obelisk::Lexer::kTokenIdentifier :
00075             return parseIdentifierExpression();
00076         case obelisk::Lexer::kTokenNumber :
00077             return parseNumberExpression();
00078         case '(' :
00079             return parseParenthesisExpression();
00080         default :
00081             return logError("unknown token when expecting and expression");
00082     }
00083 }
```

References obelisk::Lexer::kTokenIdentifier, and obelisk::Lexer::kTokenNumber.

5.20.3.24 parsePrototype()

```
std::unique_ptr< obelisk::PrototypeAST > obelisk::Parser::parsePrototype ( )  [private]
```

The AST prototype parser.

Returns

std::unique_ptr<obelisk::PrototypeAST> Returns the parsed AST prototype expression.

Definition at line 154 of file parser.cpp.

```
00155 {
00156     if (getCurrentToken() != obelisk::Lexer::kTokenIdentifier)
00157     {
00158         return logErrorPrototype("Expected function name in prototype");
00159     }
00160
00161     std::string functionName = getLexer()->getIdentifier();
00162     getNextToken();
00163
00164     if (getCurrentToken() != '(')
00165     {
00166         return logErrorPrototype("Expected '(' in prototype");
00167     }
00168
00169     std::vector<std::string> argNames;
00170     while (getNextToken() == obelisk::Lexer::kTokenIdentifier)
00171     {
00172         argNames.push_back(getLexer()->getIdentifier());
00173     }
00174
00175     if (getCurrentToken() != ')')
00176     {
00177         return logErrorPrototype("Expected ')' in prototype");
00178     }
00179
00180     getNextToken();
00181
00182     return std::make_unique<obelisk::PrototypeAST>(functionName,
00183         std::move(argNames));
00184 }
```

References obelisk::Lexer::kTokenIdentifier.

5.20.3.25 parseRule()

```
void obelisk::Parser::parseRule (
            obelisk::Rule & rule )  [private]
```

Parse a Rule.

Parameters

out	rule	The parsed Rule.

Definition at line 444 of file parser.cpp.

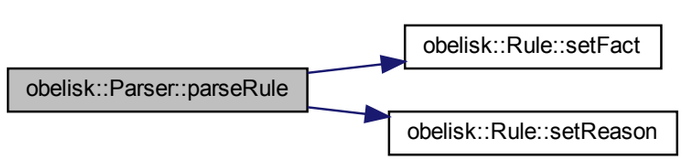
```
00445 {
00446     std::stack<char> syntax;
00447
00448     getNextToken();
00449     if (getCurrentToken() != '(')
00450     {
00451         throw obelisk::ParserException(
00452             "expected '(' but got '" + std::to_string(getCurrentToken()) + "'");
00453     }
00454
00455     syntax.push('(');
00456
00457     bool getEntity {true};
00458     bool getReason {false};
00459     std::string leftEntity {""};
00460     std::string rightEntity {""};
00461     std::string verb {""};
00462     std::string leftReasonEntity {""};
00463     std::string rightReasonEntity {""};
00464     std::string reasonVerb {""};
00465     std::string entityName {""};
00466     getNextToken();
00467
00468     // get the entity side of statement
00469     while (true)
00470     {
00471         if (getEntity)
00472         {
00473             if (getCurrentToken() == '"')
```

```
00474             {
00475                 if (syntax.top() != '"')
00476                 {
00477                     // open a double quote
00478                     syntax.push('"');
00479                     getNextToken();
00480                 }
00481                 else if (syntax.top() == '"')
00482                 {
00483                     // close a double quote
00484                     syntax.pop();
00485                     if (!getReason)
00486                     {
00487                         if (verb == "")
00488                         {
00489                             leftEntity = std::move(entityName);
00490                         }
00491                         else
00492                         {
00493                             rightEntity = std::move(entityName);
00494                         }
00495                     }
00496                     else
00497                     {
00498                         if (reasonVerb == "")
00499                         {
00500                             leftReasonEntity = std::move(entityName);
00501                         }
00502                         else
00503                         {
00504                             rightReasonEntity = std::move(entityName);
00505                         }
00506                     }
00507                     getEntity = false;
00508                     getNextToken();
00509                     continue;
00510                 }
00511             }
00512 
00513             if (syntax.top() == '"')
00514             {
00515                 if (entityName != "")
00516                 {
00517                     entityName += " ";
00518                 }
00519                 entityName += getLexer()->getIdentifier();
00520             }
00521             getNextToken();
00522         }
00523         else
00524         {
00525             if (getCurrentToken() == ')')
00526             {
00527                 // closing parenthesis found, make sure we have everything
00528                 // needed
00529                 if (syntax.top() != '(')
00530                 {
00531                     throw obelisk::ParserException("unexpected ')'");
00532                 }
00533                 else
00534                 {
00535                     syntax.pop();
00536                 }
00537 
00538                 if (leftEntity == "")
00539                 {
00540                     throw obelisk::ParserException("missing left entity");
00541                 }
00542 
00543                 if (rightEntity == "")
00544                 {
00545                     throw obelisk::ParserException("missing right entity");
00546                 }
00547 
00548                 if (verb == "")
00549                 {
00550                     throw obelisk::ParserException("missing verb");
00551                 }
00552 
00553                 if (leftReasonEntity == "")
00554                 {
00555                     throw obelisk::ParserException(
00556                         "missing left reason entity");
00557                 }
00558 
00559                 if (rightReasonEntity == "")
00560                 {
00561                     throw obelisk::ParserException(
00562                         "missing right reason entity");
00563                 }
00564 
00565                 if (reasonVerb == "")
00566                 {
00567                     throw obelisk::ParserException("missing reason verb");
00568                 }
00569 
00570                 getNextToken();
00571                 if (getCurrentToken() != ';')
00572                 {
00573                     throw obelisk::ParserException("missing ';'");
00574                 }
00575 
00576                 break;
00577             }
00578 
00579             if (getCurrentToken() == '"')
00580             {
00581                 throw obelisk::ParserException("unexpected '\"'");
```

```
00582             }
00583
00584             if (getLexer()->getIdentifier() == "if")
00585             {
00586                 getReason = true;
00587                 getEntity = true;
00588                 getNextToken();
00589                 continue;
00590             }
00591             else
00592             {
00593                 if (!getReason)
00594                 {
00595                     verb = getLexer()->getIdentifier();
00596                     for (const auto& letter : verb)
00597                     {
00598                         if (!isalpha(letter))
00599                         {
00600                             throw new obelisk::ParserException(
00601                                 "non alphabetic symbol in verb");
00602                         }
00603                     }
00604                     getEntity = true;
00605                     continue;
00606                 }
00607                 else
00608                 {
00609                     reasonVerb = getLexer()->getIdentifier();
00610                     for (const auto& letter : reasonVerb)
00611                     {
00612                         if (!isalpha(letter))
00613                         {
00614                             throw new obelisk::ParserException(
00615                                 "non alphabetic symbol in verb");
00616                         }
00617                     }
00618                     getEntity = true;
00619                     continue;
00620                 }
00621             }
00622         }
00623     }
00624
00625     rule.setFact(obelisk::Fact(obelisk::Entity(leftEntity),
00626         obelisk::Entity(rightEntity),
00627         obelisk::Verb(verb)));
00628     rule.setReason(obelisk::Fact(obelisk::Entity(leftReasonEntity),
00629         obelisk::Entity(rightReasonEntity),
00630         obelisk::Verb(reasonVerb)));
00631 }
```

References obelisk::Rule::setFact(), and obelisk::Rule::setReason().

Here is the call graph for this function:



5.20.3.26 parseTopLevelExpression()

```
std::unique_ptr< obelisk::FunctionAST > obelisk::Parser::parseTopLevelExpression ( )  [private]
```

The AST top level expression parser.

Returns

std::unique_ptr<obelisk::FunctionAST> Returns the parsed AST top level expression.

Definition at line 204 of file parser.cpp.

```
00205 {
00206     if (auto expression = parseExpression())
00207     {
00208         // Make an anonymous prototype
00209         auto prototype = std::make_unique<obelisk::PrototypeAST>("__anon_expr",
00210             std::vector<std::string>());
00211         return std::make_unique<obelisk::FunctionAST>(std::move(prototype),
00212             std::move(expression));
00213     }
00214     return nullptr;
00215 }
```

5.20.3.27 setCurrentToken()

```
void obelisk::Parser::setCurrentToken (
            int currentToken )  [private]
```

Set the current token.

Parameters

in	*currentToken*	The token should be ASCII character.

Definition at line 40 of file parser.cpp.

```
00041 {
00042     currentToken_ = currentToken;
00043 }
```

5.20.3.28 setLexer()

```
void obelisk::Parser::setLexer (
            std::shared_ptr< obelisk::Lexer > lexer )
```

Set the Lexer to use during the parsing phase.

Parameters

in	*lexer*	The Lexer.

Definition at line 16 of file parser.cpp.

```
00017 {
00018     lexer_        = lexer;
00019     currentToken_ = 0;
00020 }
```

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

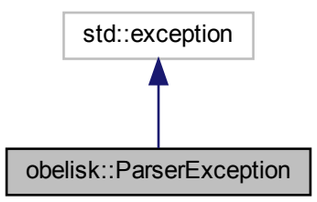
- src/parser.h
- src/parser.cpp

5.21 obelisk::ParserException Class Reference

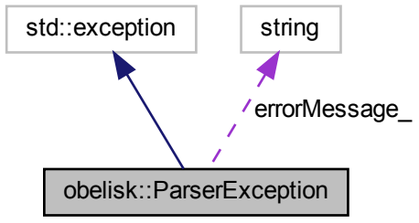
The exceptions thrown by the Parser.

```
#include <parser.h>
```

Inheritance diagram for obelisk::ParserException:



Collaboration diagram for obelisk::ParserException:



Public Member Functions

- ParserException ()
  Construct a new ParserException object.
- ParserException (const std::string &errorMessage)
  Construct a new ParserException object.
- const char * what () const noexcept
  Return the error message as a C style string.

Private Attributes

- const std::string errorMessage_
  The error message.

5.21.1 Detailed Description

The exceptions thrown by the Parser.

Definition at line 294 of file parser.h.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 ParserException()

```
obelisk::ParserException::ParserException (
            const std::string & errorMessage )  [inline]
```

Construct a new ParserException object.

Parameters

in	*errorMessage*	The error message.

Definition at line 318 of file parser.h.

```
00318                                                       :
00319                 errorMessage_(errorMessage)
00320             {
00321             }
```

5.21.3 Member Function Documentation

5.21.3.1 what()

```
const char* obelisk::ParserException::what ( ) const  [inline], [noexcept]
```

Return the error message as a C style string.

Returns

const char* Returns the error message.

Definition at line 328 of file parser.h.

```
00329             {
00330                 return errorMessage_.c_str();
00331             }
```

References errorMessage_.

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

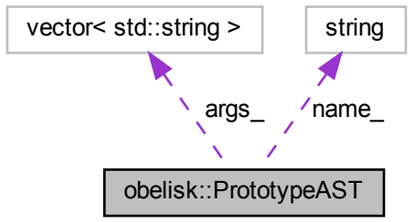
- src/parser.h

5.22 obelisk::PrototypeAST Class Reference

The prototype AST node.

```
#include <prototype_ast.h>
```

Collaboration diagram for obelisk::PrototypeAST:



Public Member Functions

- PrototypeAST (const std::string &name, std::vector< std::string > args)
  Construct a new PrototypeAST object.
- const std::string & getName () const
  Get the name of the prototype.
- llvm::Function * codegen ()
  Generate LLVM IR code for the prototype.

Private Member Functions

- void setName (const std::string &name)
  Set the name of the prototype.
- std::vector< std::string > getArgs ()
  Get the arguments the prototype accepts.
- void setArgs (std::vector< std::string > args)
  Set the arguments the prototype accepts.

Private Attributes

- std::string name_

  The name of the prototype.
- std::vector< std::string > args_

  The arguments the protype accepts.

5.22.1 Detailed Description

The prototype AST node.

Definition at line 15 of file prototype_ast.h.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 PrototypeAST()

```
obelisk::PrototypeAST::PrototypeAST (
            const std::string & name,
            std::vector< std::string > args )  [inline]
```

Construct a new PrototypeAST object.

Parameters

in	*name*	The name of the prototype.
in	*args*	The arguments the prototype accepts.

Definition at line 57 of file prototype_ast.h.

```
00058                                                :
00059                 name_(name),
00060                 args_(std::move(args))
00061             {
00062             }
```

5.22.3 Member Function Documentation

5.22.3.1 codegen()

```
llvm::Function * obelisk::PrototypeAST::codegen ( )
```

Generate LLVM IR code for the prototype.

Returns

llvm::Function* Returns IR code for the prototype.

Definition at line 4 of file prototype_ast.cpp.

```
00005 {
00006     std::vector<llvm::Type *> doubles(args_.size(),
00007         llvm::Type::getDoubleTy(*TheContext));
00008     llvm::FunctionType *FT
00009         = llvm::FunctionType::get(llvm::Type::getDoubleTy(*TheContext),
00010             doubles,
00011             false);
00012
00013     llvm::Function *F = llvm::Function::Create(FT,
00014         llvm::Function::ExternalLinkage,
00015         name_,
00016         obelisk::TheModule.get());
00017
```

```
00018     unsigned idx = 0;
00019     for (auto &arg : F->args())
00020     {
00021         arg.setName(args_[idx++]);
00022     }
00023
00024     return F;
00025 }
```

References args_, name_, obelisk::TheContext, and obelisk::TheModule.

5.22.3.2 getArgs()

```
std::vector<std::string> obelisk::PrototypeAST::getArgs ( )  [private]
```

Get the arguments the prototype accepts.

Returns

std::vector<std::string> Returns the arguments.

5.22.3.3 getName()

```
const std::string& obelisk::PrototypeAST::getName ( ) const  [inline]
```

Get the name of the prototype.

Returns

const std::string& Returns the name of the prototype.

Definition at line 69 of file prototype_ast.h.

```
00070             {
00071                 return name_;
00072             }
```

References name_.

5.22.3.4 setArgs()

```
void obelisk::PrototypeAST::setArgs (
             std::vector< std::string > args )  [private]
```

Set the arguments the prototype accepts.

Parameters

in	*args*	The arguments.

5.22.3.5 setName()

```
void obelisk::PrototypeAST::setName (
             const std::string & name )  [private]
```

Set the name of the prototype.

Parameters

in	*name*	The name.

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

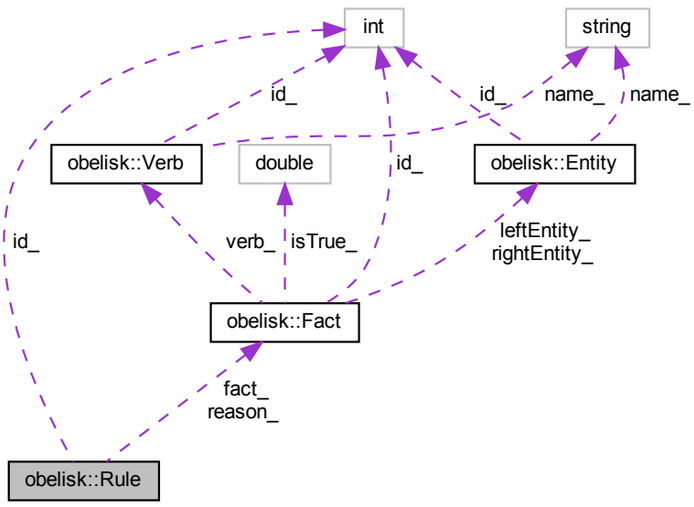
- src/ast/prototype_ast.h
- src/ast/prototype_ast.cpp

5.23 obelisk::Rule Class Reference

The Rule model represents a truth relation between 2 Facts.

`#include <rule.h>`

Collaboration diagram for obelisk::Rule:



Public Member Functions

- Rule ()
  Construct a new Rule object.
- Rule (int id)
  Construct a new Rule object.
- Rule (obelisk::Fact fact, obelisk::Fact reason)
  Construct a new Rule object.
- Rule (int id, obelisk::Fact fact, obelisk::Fact reason)
  Construct a new Rule object.
- int & getId ()
  Get the ID of the Rule.
- void setId (int id)
  Set the ID of the Rule.
- obelisk::Fact & getFact ()
  Get the Fact object.
- void setFact (obelisk::Fact fact)
  Set the Fact object.
- obelisk::Fact & getReason ()
  Get the reason Fact object.
- void setReason (obelisk::Fact reason)
  Set the reason Fact object.
- void selectById (sqlite3 ∗dbConnection)
  Select the Rule from the KnowledgeBase by IDs of the sub-objects.
- void insert (sqlite3 ∗dbConnection)
  Insert the Rule into the KnowledgeBase.

Static Public Member Functions

- static const char * createTable ()

  Create the Rule table in the KnowledgeBase.
- static void selectByReason (sqlite3 *dbConnection, int reasonId, std::vector< obelisk::Rule > &rules)

  Get the rules that match the reason.

Private Attributes

- int id_

  The ID of the Rule in the KnowledgeBase.
- obelisk::Fact fact_

  The Fact that depends on the Fact reason being true.
- obelisk::Fact reason_

  The Fact that makes the other Fact true or false.

5.23.1 Detailed Description

The Rule model represents a truth relation between 2 Facts.

Definition at line 15 of file rule.h.

5.23.2 Constructor & Destructor Documentation

5.23.2.1 Rule() [1/3]

```
obelisk::Rule::Rule (
            int id ) [inline]
```

Construct a new Rule object.

Parameters

in	*id*	The ID of the Rule in the KnowledgeBase.

Definition at line 53 of file rule.h.

```
00053                       :
00054                 id_(id),
00055                 fact_(),
00056                 reason_()
00057             {
00058             }
```

5.23.2.2 Rule() [2/3]

```
obelisk::Rule::Rule (
            obelisk::Fact fact,
            obelisk::Fact reason ) [inline]
```

Construct a new Rule object.

Parameters

in	*fact*	The Fact.
in	*reason*	The reason Fact.

Definition at line 66 of file rule.h.

```
00066                                                   :
00067                 id_(0),
00068                 fact_(fact),
00069                 reason_(reason)
00070             {
00071             }
```

5.23.2.3 Rule() [3/3]

```
obelisk::Rule::Rule (
            int id,
            obelisk::Fact fact,
            obelisk::Fact reason )  [inline]
```

Construct a new Rule object.

Parameters

in	*id*	The ID of the Rule.
in	*fact*	The Fact.
in	*reason*	The reason Fact.

Definition at line 80 of file rule.h.

```
00080                                                                   :
00081                 id_(id),
00082                 fact_(fact),
00083                 reason_(reason)
00084             {
00085             }
```

5.23.3 Member Function Documentation

5.23.3.1 createTable()

```
const char * obelisk::Rule::createTable ( )  [static]
```

Create the Rule table in the KnowledgeBase.

Returns

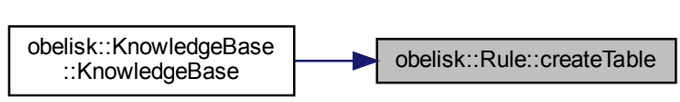
const char* Returns the query used to create the table.

Definition at line 4 of file rule.cpp.

```
00005 {
00006     return R"(
00007         CREATE TABLE "rule" (
00008             "id"     INTEGER NOT NULL UNIQUE,
00009             "fact"   INTEGER NOT NULL,
00010             "reason" INTEGER NOT NULL CHECK("reason" != "fact"),
00011             PRIMARY KEY("id" AUTOINCREMENT),
00012             UNIQUE("fact", "reason"),
00013             FOREIGN KEY("fact") REFERENCES "fact"("id") ON DELETE RESTRICT,
00014             FOREIGN KEY("reason") REFERENCES "fact"("id") ON DELETE RESTRICT
00015         );
00016     )";
00017 }
```

Referenced by obelisk::KnowledgeBase::KnowledgeBase().

Here is the caller graph for this function:



5.23.3.2 getFact()

```
obelisk::Fact & obelisk::Rule::getFact ( )
```

Get the Fact object.

Returns

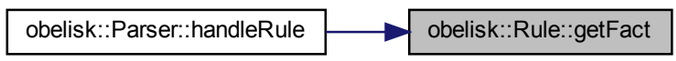
obelisk::Fact& The Fact.

Definition at line 268 of file rule.cpp.

```
00269 {
00270     return fact_;
00271 }
```

Referenced by obelisk::Parser::handleRule().

Here is the caller graph for this function:



5.23.3.3 getId()

```
int & obelisk::Rule::getId ( )
```

Get the ID of the Rule.

Returns

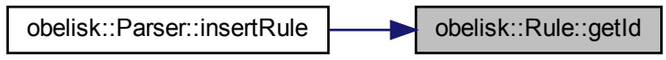
int& The ID.

Definition at line 258 of file rule.cpp.

```
00259 {
00260     return id_;
00261 }
```

Referenced by obelisk::Parser::insertRule().

Here is the caller graph for this function:



5.23.3.4 getReason()

```
obelisk::Fact & obelisk::Rule::getReason ( )
```

Get the reason Fact object.

Returns

obelisk::Fact& The reason Fact.

Definition at line 278 of file rule.cpp.

```
00279 {
00280     return reason_;
00281 }
```

Referenced by obelisk::Parser::handleRule().

Here is the caller graph for this function:

5.23.3.5 insert()

```
void obelisk::Rule::insert (
            sqlite3 * dbConnection )
```

Insert the Rule into the KnowledgeBase.

Parameters

in	*dbConnection*	The database connection to use.

Definition at line 105 of file rule.cpp.

```
00106 {
00107     if (dbConnection == nullptr)
00108     {
00109         throw obelisk::DatabaseException("database isn't open");
00110     }
00111
00112     sqlite3_stmt* ppStmt = nullptr;
00113
00114     auto result = sqlite3_prepare_v2(dbConnection,
00115         "INSERT INTO rule (fact, reason) VALUES (?, ?)",
00116         -1,
00117         &ppStmt,
00118         nullptr);
00119     if (result != SQLITE_OK)
00120     {
00121         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00122     }
00123
00124     result = sqlite3_bind_int(ppStmt, 1, getFact().getId());
00125     switch (result)
00126     {
00127         case SQLITE_OK :
00128             break;
00129         case SQLITE_TOOBIG :
00130             throw obelisk::DatabaseSizeException();
00131             break;
00132         case SQLITE_RANGE :
00133             throw obelisk::DatabaseRangeException();
00134             break;
00135         case SQLITE_NOMEM :
00136             throw obelisk::DatabaseMemoryException();
00137             break;
00138         default :
00139             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00140             break;
00141     }
00142
```

```
00143     result = sqlite3_bind_int(ppStmt, 2, getReason().getId());
00144     switch (result)
00145     {
00146         case SQLITE_OK :
00147             break;
00148         case SQLITE_TOOBIG :
00149             throw obelisk::DatabaseSizeException();
00150             break;
00151         case SQLITE_RANGE :
00152             throw obelisk::DatabaseRangeException();
00153             break;
00154         case SQLITE_NOMEM :
00155             throw obelisk::DatabaseMemoryException();
00156             break;
00157         default :
00158             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00159             break;
00160     }
00161
00162     result = sqlite3_step(ppStmt);
00163     switch (result)
00164     {
00165         case SQLITE_DONE :
00166             setId((int) sqlite3_last_insert_rowid(dbConnection));
00167             sqlite3_set_last_insert_rowid(dbConnection, 0);
00168             break;
00169         case SQLITE_CONSTRAINT :
00170             throw obelisk::DatabaseConstraintException(
00171                 sqlite3_errmsg(dbConnection));
00172         case SQLITE_BUSY :
00173             throw obelisk::DatabaseBusyException();
00174             break;
00175         case SQLITE_MISUSE :
00176             throw obelisk::DatabaseMisuseException();
00177             break;
00178         default :
00179             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00180             break;
00181     }
00182
00183     result = sqlite3_finalize(ppStmt);
00184     if (result != SQLITE_OK)
00185     {
00186         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00187     }
00188 }
```

5.23.3.6 selectById()

```
void obelisk::Rule::selectById (
            sqlite3 * dbConnection )
```

Select the Rule from the KnowledgeBase by IDs of the sub-objects.

Parameters

in	*dbConnection*	The database connection to use.

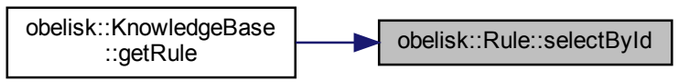
Definition at line 19 of file rule.cpp.

```
00020 {
00021     if (dbConnection == nullptr)
00022     {
00023         throw obelisk::DatabaseException("database isn't open");
00024     }
00025
00026     sqlite3_stmt* ppStmt = nullptr;
00027
00028     auto result = sqlite3_prepare_v2(dbConnection,
00029         "SELECT id, fact, reason FROM rule WHERE (fact=? AND reason=?)",
00030         -1,
00031         &ppStmt,
00032         nullptr);
00033     if (result != SQLITE_OK)
00034     {
00035         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00036     }
00037
00038     result = sqlite3_bind_int(ppStmt, 1, getFact().getId());
00039     switch (result)
00040     {
00041         case SQLITE_OK :
00042             break;
00043         case SQLITE_TOOBIG :
00044             throw obelisk::DatabaseSizeException();
00045             break;
00046         case SQLITE_RANGE :
00047             throw obelisk::DatabaseRangeException();
00048             break;
00049         case SQLITE_NOMEM :
00050             throw obelisk::DatabaseMemoryException();
00051             break;
00052         default :
00053             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00054             break;
```

```
00055     }
00056
00057     result = sqlite3_bind_int(ppStmt, 2, getReason().getId());
00058     switch (result)
00059     {
00060         case SQLITE_OK :
00061             break;
00062         case SQLITE_TOOBIG :
00063             throw obelisk::DatabaseSizeException();
00064             break;
00065         case SQLITE_RANGE :
00066             throw obelisk::DatabaseRangeException();
00067             break;
00068         case SQLITE_NOMEM :
00069             throw obelisk::DatabaseMemoryException();
00070             break;
00071         default :
00072             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00073             break;
00074     }
00075
00076     result = sqlite3_step(ppStmt);
00077     switch (result)
00078     {
00079         case SQLITE_DONE :
00080             // no rows in the database
00081             break;
00082         case SQLITE_ROW :
00083             setId(sqlite3_column_int(ppStmt, 0));
00084             getFact().setId(sqlite3_column_int(ppStmt, 1));
00085             getReason().setId(sqlite3_column_int(ppStmt, 2));
00086             break;
00087         case SQLITE_BUSY :
00088             throw obelisk::DatabaseBusyException();
00089             break;
00090         case SQLITE_MISUSE :
00091             throw obelisk::DatabaseMisuseException();
00092             break;
00093         default :
00094             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00095             break;
00096     }
00097
00098     result = sqlite3_finalize(ppStmt);
00099     if (result != SQLITE_OK)
00100     {
00101         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00102     }
00103 }
```

Referenced by obelisk::KnowledgeBase::getRule().

Here is the caller graph for this function:



5.23.3.7 selectByReason()

```
void obelisk::Rule::selectByReason (
            sqlite3 * dbConnection,
            int reasonId,
            std::vector< obelisk::Rule > & rules )  [static]
```

Get the rules that match the reason.

Parameters

in	*dbConnection*	The database connection to use.
out	*rules*	The rules to fill in from the database.

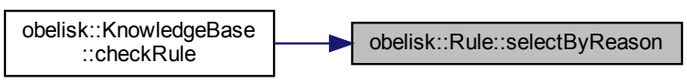
Definition at line 190 of file rule.cpp.

```
00193 {
00194     if (dbConnection == nullptr)
00195     {
```

```
00196         throw obelisk::DatabaseException("database isn't open");
00197     }
00198
00199     sqlite3_stmt* ppStmt = nullptr;
00200
00201     auto result = sqlite3_prepare_v2(dbConnection,
00202         "SELECT id, fact, reason FROM rule WHERE (reason=?)",
00203         -1,
00204         &ppStmt,
00205         nullptr);
00206     if (result != SQLITE_OK)
00207     {
00208         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00209     }
00210
00211     result = sqlite3_bind_int(ppStmt, 1, reasonId);
00212     switch (result)
00213     {
00214         case SQLITE_OK :
00215             break;
00216         case SQLITE_TOOBIG :
00217             throw obelisk::DatabaseSizeException();
00218             break;
00219         case SQLITE_RANGE :
00220             throw obelisk::DatabaseRangeException();
00221             break;
00222         case SQLITE_NOMEM :
00223             throw obelisk::DatabaseMemoryException();
00224             break;
00225         default :
00226             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00227             break;
00228     }
00229
00230     while ((result = sqlite3_step(ppStmt)) != SQLITE_DONE)
00231     {
00232         switch (result)
00233         {
00234             case SQLITE_ROW :
00235                 rules.push_back(obelisk::Rule(sqlite3_column_int(ppStmt, 0),
00236                     obelisk::Fact(sqlite3_column_int(ppStmt, 1)),
00237                     obelisk::Fact(sqlite3_column_int(ppStmt, 2))));
00238                 break;
00239             case SQLITE_BUSY :
00240                 throw obelisk::DatabaseBusyException();
00241                 break;
00242             case SQLITE_MISUSE :
00243                 throw obelisk::DatabaseMisuseException();
00244                 break;
00245             default :
00246                 throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00247                 break;
00248         }
00249     }
00250
00251     result = sqlite3_finalize(ppStmt);
00252     if (result != SQLITE_OK)
00253     {
00254         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00255     }
00256 }
```

Referenced by obelisk::KnowledgeBase::checkRule().

Here is the caller graph for this function:



5.23.3.8 setFact()

```
void obelisk::Rule::setFact (
            obelisk::Fact fact )
```

Set the Fact object.

Parameters

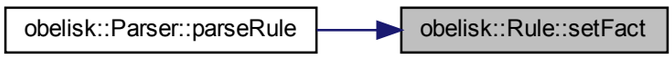
in	*fact*	The Fact.

Definition at line 273 of file rule.cpp.

```
00274 {
00275     fact_ = fact;
00276 }
```

Referenced by obelisk::Parser::parseRule().

Here is the caller graph for this function:



5.23.3.9 setId()

```
void obelisk::Rule::setId (
            int id )
```

Set the ID of the Rule.

Parameters

in	*id*	The ID.

Definition at line 263 of file rule.cpp.

```
00264 {
00265     id_ = id;
00266 }
```

5.23.3.10 setReason()

```
void obelisk::Rule::setReason (
            obelisk::Fact reason )
```

Set the reason Fact object.

Parameters

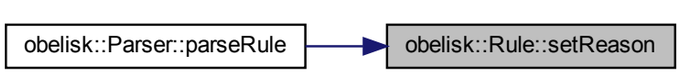
in	*reason*	The reason Fact.

Definition at line 283 of file rule.cpp.

```
00284 {
00285     reason_ = reason;
00286 }
```

Referenced by obelisk::Parser::parseRule().

Here is the caller graph for this function:



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

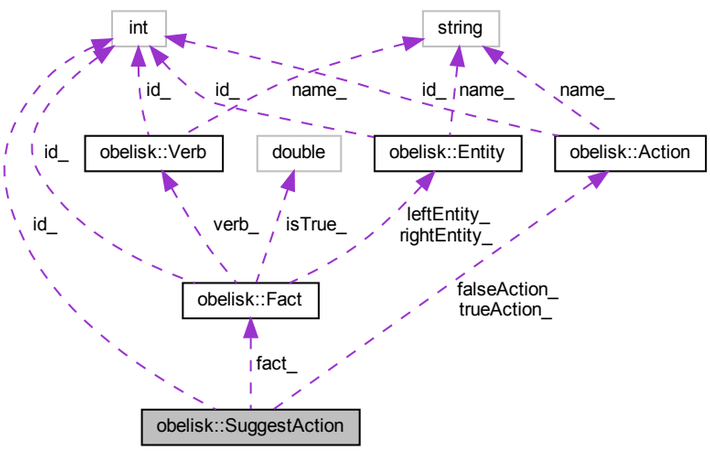
- src/lib/include/models/rule.h
- src/lib/models/rule.cpp

5.24 obelisk::SuggestAction Class Reference

The SuggestAction model representas the actions to take depending on if the Fact is true or false.

`#include <suggest_action.h>`

Collaboration diagram for obelisk::SuggestAction:



Public Member Functions

- SuggestAction ()
  Construct a new SuggestAction object.
- SuggestAction (int id)
  Construct a new SuggestAction object.
- SuggestAction (obelisk::Fact fact, obelisk::Action trueAction, obelisk::Action falseAction)
  Construct a new SuggestAction object.
- SuggestAction (int id, obelisk::Fact fact, obelisk::Action trueAction, obelisk::Action falseAction)
  Construct a new SuggestAction object.
- int & getId ()
  Get the ID of the SuggestAction.
- void setId (int id)
  Set the ID of the SuggestAction.
- obelisk::Fact & getFact ()
  Get the Fact object.
- void setFact (obelisk::Fact fact)
  Set the Fact object.
- obelisk::Action & getTrueAction ()
  Get the true Action object.
- void setTrueAction (obelisk::Action trueAction)
  Set the true Action object.
- obelisk::Action & getFalseAction ()
  Get the false Action object.
- void setFalseAction (obelisk::Action falseAction)
  Set the false Action object.
- void selectById (sqlite3 *dbConnection)
  Select the SuggestAction from the KnowledgeBase by IDs of the sub-objects.
- void insert (sqlite3 *dbConnection)
  Insert the SuggestAction into the KnowledgeBase.

Static Public Member Functions

- static const char * createTable ()

  Create the SuggestAction table in the database.

Private Attributes

- int id_

  The ID of the SuggestAction.
- obelisk::Fact fact_

  The Fact to check the truth of.
- obelisk::Action trueAction_

  The Action to take if the Fact is true.
- obelisk::Action falseAction_

  The Action to take if the Fact is false.

5.24.1 Detailed Description

The SuggestAction model representas the actions to take depending on if the Fact is true or false.

Definition at line 16 of file suggest_action.h.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 SuggestAction() [1/3]

```
obelisk::SuggestAction::SuggestAction (
            int id )  [inline]
```

Construct a new SuggestAction object.

Parameters

in	*id*	The ID of the SuggestAction in the KnowledgeBase.

Definition at line 61 of file suggest_action.h.

```
00061                                  :
00062                 id_(id),
00063                 fact_(),
00064                 trueAction_(),
00065                 falseAction_()
00066             {
00067             }
```

5.24.2.2 SuggestAction() [2/3]

```
obelisk::SuggestAction::SuggestAction (
            obelisk::Fact fact,
            obelisk::Action trueAction,
            obelisk::Action falseAction )  [inline]
```

Construct a new SuggestAction object.

Parameters

in	*fact*	The Fact.
in	*trueAction*	The true Action.
in	*falseAction*	The false Action.

Definition at line 76 of file suggest_action.h.

```
00078                                                 :
00079                 id_(0),
00080                 fact_(fact),
00081                 trueAction_(trueAction),
00082                 falseAction_(falseAction)
00083             {
00084             }
```

5.24.2.3 SuggestAction() [3/3]

```
obelisk::SuggestAction::SuggestAction (
            int id,
            obelisk::Fact fact,
            obelisk::Action trueAction,
            obelisk::Action falseAction )  [inline]
```

Construct a new SuggestAction object.

Parameters

in	*id*	The ID of the SuggestAction in the KnowledgeBase.
in	*fact*	The Fact.
in	*trueAction*	The true Action.
in	*falseAction*	The false Action.

Definition at line 94 of file suggest_action.h.

```
00097                                                 :
00098                 id_(id),
00099                 fact_(fact),
00100                 trueAction_(trueAction),
00101                 falseAction_(falseAction)
00102             {
00103             }
```

5.24.3 Member Function Documentation

5.24.3.1 createTable()

```
const char * obelisk::SuggestAction::createTable ( )  [static]
```

Create the SuggestAction table in the database.

Returns

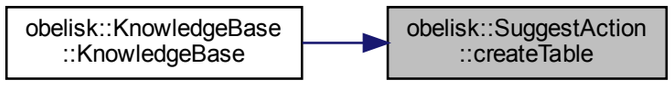
const char* Returns the query used to create the table.

Definition at line 4 of file suggest_action.cpp.

```
00005 {
00006     return R"(
00007         CREATE TABLE "suggest_action" (
00008             "id"           INTEGER NOT NULL UNIQUE,
00009             "fact"         INTEGER NOT NULL,
00010             "true_action"  INTEGER NOT NULL,
00011             "false_action" INTEGER NOT NULL,
00012             PRIMARY KEY("id" AUTOINCREMENT),
00013             UNIQUE("fact", "true_action", "false_action"),
00014             FOREIGN KEY("fact") REFERENCES "fact"("id") ON DELETE RESTRICT,
00015             FOREIGN KEY("true_action") REFERENCES "action"("id") ON DELETE RESTRICT,
00016             FOREIGN KEY("false_action") REFERENCES "action"("id") ON DELETE RESTRICT
00017         );
00018     )";
00019 }
```

Referenced by obelisk::KnowledgeBase::KnowledgeBase().

Here is the caller graph for this function:



5.24.3.2 getFact()

```
obelisk::Fact & obelisk::SuggestAction::getFact ( )
```

Get the Fact object.

Returns

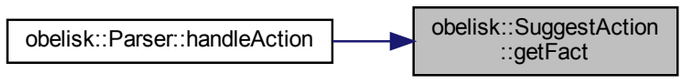
obelisk::Fact& Returns the Fact.

Definition at line 241 of file suggest_action.cpp.

```
00242 {
00243     return fact_;
00244 }
```

Referenced by obelisk::Parser::handleAction().

Here is the caller graph for this function:



5.24.3.3 getFalseAction()

```
obelisk::Action & obelisk::SuggestAction::getFalseAction ( )
```

Get the false Action object.

Returns

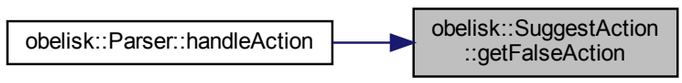
obelisk::Action& Returns the false Action.

Definition at line 261 of file suggest_action.cpp.

```
00262 {
00263     return falseAction_;
00264 }
```

Referenced by obelisk::Parser::handleAction().

Here is the caller graph for this function:



5.24.3.4 getId()

```
int & obelisk::SuggestAction::getId ( )
```

Get the ID of the SuggestAction.

Returns

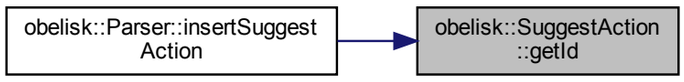
int& Returns the ID.

Definition at line 231 of file suggest_action.cpp.

```
00232 {
00233     return id_;
00234 }
```

Referenced by obelisk::Parser::insertSuggestAction().

Here is the caller graph for this function:



5.24.3.5 getTrueAction()

```
obelisk::Action & obelisk::SuggestAction::getTrueAction ( )
```

Get the true Action object.

Returns

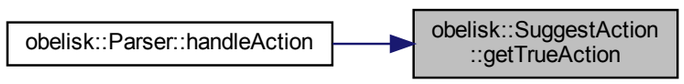
obelisk::Action& Returns the true Action.

Definition at line 251 of file suggest_action.cpp.

```
00252 {
00253     return trueAction_;
00254 }
```

Referenced by obelisk::Parser::handleAction().

Here is the caller graph for this function:



5.24.3.6 insert()

```
void obelisk::SuggestAction::insert (
            sqlite3 * dbConnection )
```

Insert the SuggestAction into the KnowledgeBase.

Parameters

in	*dbConnection*	The database connection to use.

Definition at line 127 of file suggest_action.cpp.

```
00128 {
00129     if (dbConnection == nullptr)
00130     {
00131         throw obelisk::DatabaseException("database isn't open");
00132     }
00133
00134     sqlite3_stmt* ppStmt = nullptr;
00135
00136     auto result = sqlite3_prepare_v2(dbConnection,
00137         "INSERT INTO suggest_action (fact, true_action, false_action) VALUES (?, ?, ?)",
00138         -1,
00139         &ppStmt,
00140         nullptr);
00141     if (result != SQLITE_OK)
00142     {
00143         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00144     }
00145
00146     result = sqlite3_bind_int(ppStmt, 1, getFact().getId());
00147     switch (result)
00148     {
00149         case SQLITE_OK :
00150             break;
00151         case SQLITE_TOOBIG :
00152             throw obelisk::DatabaseSizeException();
00153             break;
00154         case SQLITE_RANGE :
00155             throw obelisk::DatabaseRangeException();
00156             break;
00157         case SQLITE_NOMEM :
00158             throw obelisk::DatabaseMemoryException();
00159             break;
00160         default :
00161             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00162             break;
00163     }
00164
00165     result = sqlite3_bind_int(ppStmt, 2, getTrueAction().getId());
00166     switch (result)
00167     {
00168         case SQLITE_OK :
00169             break;
00170         case SQLITE_TOOBIG :
00171             throw obelisk::DatabaseSizeException();
00172             break;
00173         case SQLITE_RANGE :
00174             throw obelisk::DatabaseRangeException();
00175             break;
00176         case SQLITE_NOMEM :
00177             throw obelisk::DatabaseMemoryException();
00178             break;
00179         default :
00180             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00181             break;
00182     }
00183
00184     result = sqlite3_bind_int(ppStmt, 3, getFalseAction().getId());
00185     switch (result)
00186     {
00187         case SQLITE_OK :
00188             break;
00189         case SQLITE_TOOBIG :
00190             throw obelisk::DatabaseSizeException();
00191             break;
00192         case SQLITE_RANGE :
00193             throw obelisk::DatabaseRangeException();
00194             break;
00195         case SQLITE_NOMEM :
00196             throw obelisk::DatabaseMemoryException();
00197             break;
00198         default :
00199             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00200             break;
00201     }
00202
00203     result = sqlite3_step(ppStmt);
00204     switch (result)
00205     {
00206         case SQLITE_DONE :
00207             setId((int) sqlite3_last_insert_rowid(dbConnection));
00208             sqlite3_set_last_insert_rowid(dbConnection, 0);
00209             break;
00210         case SQLITE_CONSTRAINT :
00211             throw obelisk::DatabaseConstraintException(
00212                 sqlite3_errmsg(dbConnection));
00213         case SQLITE_BUSY :
00214             throw obelisk::DatabaseBusyException();
00215             break;
00216         case SQLITE_MISUSE :
00217             throw obelisk::DatabaseMisuseException();
00218             break;
00219         default :
00220             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00221             break;
00222     }
00223
00224     result = sqlite3_finalize(ppStmt);
00225     if (result != SQLITE_OK)
00226     {
```

```
00227         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00228     }
00229 }
```

5.24.3.7 selectById()

```
void obelisk::SuggestAction::selectById (
            sqlite3 * dbConnection )
```

Select the SuggestAction from the KnowledgeBase by IDs of the sub-objects.

Parameters

in	*dbConnection*	The database connection to use.

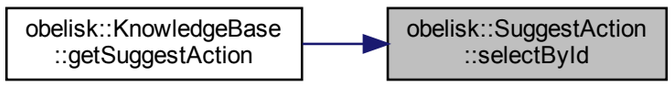
Definition at line 21 of file suggest_action.cpp.

```
00022 {
00023     if (dbConnection == nullptr)
00024     {
00025         throw obelisk::DatabaseException("database isn't open");
00026     }
00027
00028     sqlite3_stmt* ppStmt = nullptr;
00029
00030     auto result = sqlite3_prepare_v2(dbConnection,
00031         "SELECT id, fact, true_action, false_action FROM suggest_action WHERE (fact=? AND true_action=?
      AND false_action=?)",
00032         -1,
00033         &ppStmt,
00034         nullptr);
00035     if (result != SQLITE_OK)
00036     {
00037         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00038     }
00039
00040     result = sqlite3_bind_int(ppStmt, 1, getFact().getId());
00041     switch (result)
00042     {
00043         case SQLITE_OK :
00044             break;
00045         case SQLITE_TOOBIG :
00046             throw obelisk::DatabaseSizeException();
00047             break;
00048         case SQLITE_RANGE :
00049             throw obelisk::DatabaseRangeException();
00050             break;
00051         case SQLITE_NOMEM :
00052             throw obelisk::DatabaseMemoryException();
00053             break;
00054         default :
00055             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00056             break;
00057     }
00058
00059     result = sqlite3_bind_int(ppStmt, 2, getTrueAction().getId());
00060     switch (result)
00061     {
00062         case SQLITE_OK :
00063             break;
00064         case SQLITE_TOOBIG :
00065             throw obelisk::DatabaseSizeException();
00066             break;
00067         case SQLITE_RANGE :
00068             throw obelisk::DatabaseRangeException();
00069             break;
00070         case SQLITE_NOMEM :
00071             throw obelisk::DatabaseMemoryException();
00072             break;
00073         default :
00074             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00075             break;
00076     }
00077
00078     result = sqlite3_bind_int(ppStmt, 3, getFalseAction().getId());
00079     switch (result)
00080     {
00081         case SQLITE_OK :
00082             break;
00083         case SQLITE_TOOBIG :
00084             throw obelisk::DatabaseSizeException();
00085             break;
00086         case SQLITE_RANGE :
00087             throw obelisk::DatabaseRangeException();
00088             break;
00089         case SQLITE_NOMEM :
00090             throw obelisk::DatabaseMemoryException();
00091             break;
00092         default :
00093             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00094             break;
00095     }
00096
00097     result = sqlite3_step(ppStmt);
00098     switch (result)
```

```
00099     {
00100         case SQLITE_DONE :
00101             // no rows in the database
00102             break;
00103         case SQLITE_ROW :
00104             setId(sqlite3_column_int(ppStmt, 0));
00105             getFact().setId(sqlite3_column_int(ppStmt, 1));
00106             getTrueAction().setId(sqlite3_column_int(ppStmt, 2));
00107             getFalseAction().setId(sqlite3_column_int(ppStmt, 3));
00108             break;
00109         case SQLITE_BUSY :
00110             throw obelisk::DatabaseBusyException();
00111             break;
00112         case SQLITE_MISUSE :
00113             throw obelisk::DatabaseMisuseException();
00114             break;
00115         default :
00116             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00117             break;
00118     }
00119
00120     result = sqlite3_finalize(ppStmt);
00121     if (result != SQLITE_OK)
00122     {
00123         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00124     }
00125 }
```

Referenced by obelisk::KnowledgeBase::getSuggestAction().

Here is the caller graph for this function:



5.24.3.8 setFact()

```
void obelisk::SuggestAction::setFact (
            obelisk::Fact fact )
```

Set the Fact object.

Parameters

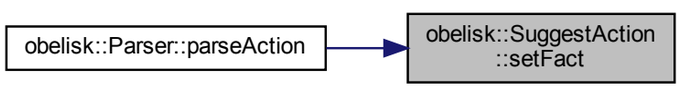
in	*fact*	The new Fact.

Definition at line 246 of file suggest_action.cpp.

```
00247 {
00248     fact_ = fact;
00249 }
```

Referenced by obelisk::Parser::parseAction().

Here is the caller graph for this function:



5.24.3.9 setFalseAction()

```
void obelisk::SuggestAction::setFalseAction (
            obelisk::Action falseAction )
```

Set the false Action object.

Parameters

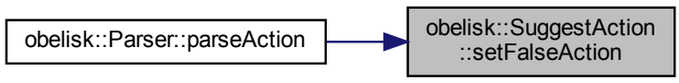
in	*falseAction*	The new false Action.

Definition at line 266 of file suggest_action.cpp.

```
00267 {
00268     falseAction_ = falseAction;
00269 }
```

Referenced by obelisk::Parser::parseAction().

Here is the caller graph for this function:



5.24.3.10 setId()

```
void obelisk::SuggestAction::setId (
            int id )
```

Set the ID of the SuggestAction.

Parameters

in	*id*	The new ID.

Definition at line 236 of file suggest_action.cpp.

```
00237 {
00238     id_ = id;
00239 }
```

5.24.3.11 setTrueAction()

```
void obelisk::SuggestAction::setTrueAction (
            obelisk::Action trueAction )
```

Set the true Action object.

Parameters

in	*trueAction*	The new true Action.

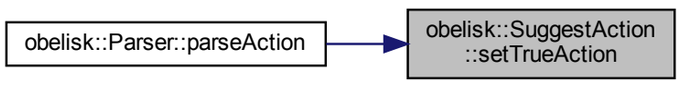
Definition at line 256 of file suggest_action.cpp.

```
00257 {
00258     trueAction_ = trueAction;
```

```
00259 }
```

Referenced by obelisk::Parser::parseAction().

Here is the caller graph for this function:



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

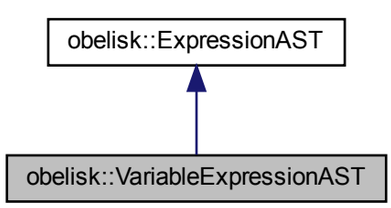
- src/lib/include/models/suggest_action.h
- src/lib/models/suggest_action.cpp

5.25 obelisk::VariableExpressionAST Class Reference

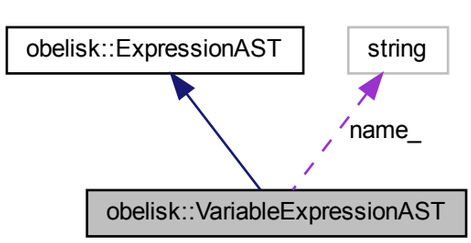
The variable expression AST node.

`#include <variable_expression_ast.h>`

Inheritance diagram for obelisk::VariableExpressionAST:



Collaboration diagram for obelisk::VariableExpressionAST:



Public Member Functions

- VariableExpressionAST (const std::string &name)

  Construct a new VariableExpressionAST object.
- llvm::Value * codegen () override

  Generate the variable LLVM IR code.

Private Member Functions

- std::string getName ()

  Get the name of the variable.
- void setName (const std::string name)

  Set the name of the variable.

Private Attributes

- std::string name_

  The name of the variable.

5.25.1 Detailed Description

The variable expression AST node.

Definition at line 14 of file variable_expression_ast.h.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 VariableExpressionAST()

```
obelisk::VariableExpressionAST::VariableExpressionAST (
            const std::string & name )  [inline]
```

Construct a new VariableExpressionAST object.

Parameters

in	*name*	The name of the variable.

Definition at line 43 of file variable_expression_ast.h.

```
00043                                                       :
00044                 name_(name)
00045             {
00046             }
```

5.25.3 Member Function Documentation

5.25.3.1 codegen()

```
llvm::Value * obelisk::VariableExpressionAST::codegen ( )  [override], [virtual]
```

Generate the variable LLVM IR code.

Returns

llvm::Value* Returns the generated IR code.

Implements obelisk::ExpressionAST.

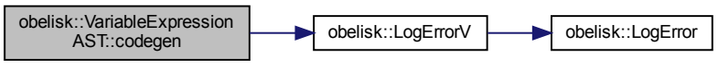
Definition at line 5 of file variable_expression_ast.cpp.

```
00006 {
00007     llvm::Value *V = NamedValues[name_];
00008     if (!V)
00009     {
00010         return obelisk::LogErrorV("Unknown variable name");
```

```
00011     }
00012     return V;
00013 }
```

References obelisk::LogErrorV(), name_, and obelisk::NamedValues.

Here is the call graph for this function:



5.25.3.2 getName()

```
std::string obelisk::VariableExpressionAST::getName ( )  [private]
```

Get the name of the variable.

Returns

std::string Returns the name of the variable.

5.25.3.3 setName()

```
void obelisk::VariableExpressionAST::setName (
            const std::string name )  [private]
```

Set the name of the variable.

Parameters

in	*name*	The name of the variable.

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

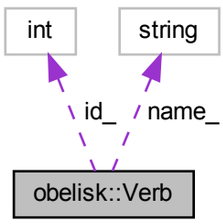
- src/ast/variable_expression_ast.h
- src/ast/variable_expression_ast.cpp

5.26 obelisk::Verb Class Reference

The Verb model represents a verb which is used to connnect entities.

```
#include <verb.h>
```

Collaboration diagram for obelisk::Verb:



Public Member Functions

- Verb ()
  Construct a new Verb object.
- Verb (int id)
  Construct a new Verb object.
- Verb (std::string name)
  Construct a new Verb object.
- Verb (int id, std::string name)
  Construct a new Verb object.
- int & getId ()
  Get the ID of the Verb.
- void setId (int id)
  Set the ID of the Verb.
- std::string & getName ()
  Get the name of the Verb.
- void setName (std::string name)
  Set the name of the Verb.
- void selectByName (sqlite3 *dbConnection)
  Select a verb by name from the KnowledgeBase.
- void insert (sqlite3 *dbConnection)
  Insert a new verb into the KnowledgeBase.

Static Public Member Functions

- static const char * createTable ()
  Create the Verb table in the KnowledgeBase.

Private Attributes

- int id_
  The ID of the Verb in the KnowledgeBase.
- std::string name_
  The name of the Verb.

5.26.1 Detailed Description

The Verb model represents a verb which is used to connnect entities.

Definition at line 15 of file verb.h.

5.26.2 Constructor & Destructor Documentation

5.26.2.1 Verb() [1/3]

```
obelisk::Verb::Verb (
            int id ) [inline]
```

Construct a new Verb object.

Parameters

in	*id*	The ID of the Verb.

Definition at line 46 of file verb.h.

```
00046                       :
00047                   id_(id),
00048                   name_("")
00049               {
00050               }
```

5.26.2.2 Verb() [2/3]

```
obelisk::Verb::Verb (
            std::string name )  [inline]
```

Construct a new Verb object.

Parameters

in	*name*	The name of the Verb.

Definition at line 57 of file verb.h.

```
00057                                :
00058                   id_(0),
00059                   name_(name)
00060               {
00061               }
```

5.26.2.3 Verb() [3/3]

```
obelisk::Verb::Verb (
            int id,
            std::string name )  [inline]
```

Construct a new Verb object.

Parameters

in	*id*	The ID of the Verb.
in	*name*	The name of the Verb.

Definition at line 69 of file verb.h.

```
00069                                       :
00070                   id_(id),
00071                   name_(name)
00072               {
00073               }
```

5.26.3 Member Function Documentation

5.26.3.1 createTable()

```
const char * obelisk::Verb::createTable ( )  [static]
```

Create the Verb table in the KnowledgeBase.

Returns

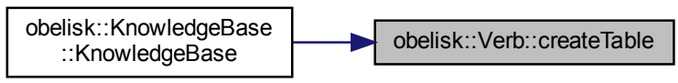
const char* Returns the query used to create the table.

Definition at line 6 of file verb.cpp.

```
00007 {
00008     return R"(
00009         CREATE TABLE "verb" (
00010             "id"   INTEGER NOT NULL UNIQUE,
00011             "name" TEXT NOT NULL CHECK(trim(name) != "") UNIQUE,
00012             PRIMARY KEY("id" AUTOINCREMENT)
00013         );
00014     )";
00015 }
```

Referenced by obelisk::KnowledgeBase::KnowledgeBase().

Here is the caller graph for this function:



5.26.3.2 getId()

```
int & obelisk::Verb::getId ( )
```

Get the ID of the Verb.

Returns

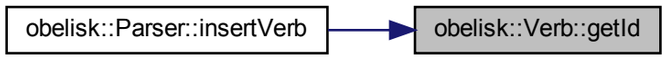
int& Returns the ID.

Definition at line 150 of file verb.cpp.

```
00151 {
00152     return id_;
00153 }
```

Referenced by obelisk::Parser::insertVerb().

Here is the caller graph for this function:



5.26.3.3 getName()

```
std::string & obelisk::Verb::getName ( )
```

Get the name of the Verb.

Returns

std::string& The Verb name.

Definition at line 160 of file verb.cpp.

```
00161 {
00162     return name_;
00163 }
```

5.26.3.4 insert()

```
void obelisk::Verb::insert (
            sqlite3 * dbConnection )
```

Insert a new verb into the KnowledgeBase.

Parameters

in	*dbConnection*	The database connection to use.

Definition at line 83 of file verb.cpp.

```
00084 {
00085     if (dbConnection == nullptr)
00086     {
00087         throw obelisk::DatabaseException("database isn't open");
00088     }
00089
00090     sqlite3_stmt* ppStmt = nullptr;
00091
00092     auto result = sqlite3_prepare_v2(dbConnection,
00093         "INSERT INTO verb (name) VALUES (?)",
00094         -1,
00095         &ppStmt,
00096         nullptr);
00097     if (result != SQLITE_OK)
00098     {
00099         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00100     }
00101
00102     result
00103         = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_TRANSIENT);
00104     switch (result)
00105     {
00106         case SQLITE_OK :
00107             break;
00108         case SQLITE_TOOBIG :
00109             throw obelisk::DatabaseSizeException();
00110             break;
00111         case SQLITE_RANGE :
00112             throw obelisk::DatabaseRangeException();
00113             break;
00114         case SQLITE_NOMEM :
00115             throw obelisk::DatabaseMemoryException();
00116             break;
00117         default :
00118             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00119             break;
00120     }
00121
00122     result = sqlite3_step(ppStmt);
00123     switch (result)
00124     {
00125         case SQLITE_DONE :
00126             setId((int) sqlite3_last_insert_rowid(dbConnection));
00127             sqlite3_set_last_insert_rowid(dbConnection, 0);
00128             break;
00129         case SQLITE_CONSTRAINT :
00130             throw obelisk::DatabaseConstraintException(
00131                 sqlite3_errmsg(dbConnection));
00132         case SQLITE_BUSY :
00133             throw obelisk::DatabaseBusyException();
00134             break;
00135         case SQLITE_MISUSE :
00136             throw obelisk::DatabaseMisuseException();
00137             break;
00138         default :
00139             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00140             break;
00141     }
00142
00143     result = sqlite3_finalize(ppStmt);
00144     if (result != SQLITE_OK)
00145     {
00146         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00147     }
00148 }
```

5.26.3.5 selectByName()

```
void obelisk::Verb::selectByName (
            sqlite3 * dbConnection )
```

Select a verb by name from the KnowledgeBase.

Parameters

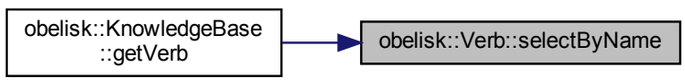
in	*dbConnection*	The database connection to use.

Definition at line 17 of file verb.cpp.

```
00018 {
00019     if (dbConnection == nullptr)
00020     {
00021         throw obelisk::DatabaseException("database isn't open");
00022     }
00023
00024     sqlite3_stmt* ppStmt = nullptr;
00025
00026     auto result = sqlite3_prepare_v2(dbConnection,
00027         "SELECT id, name FROM verb WHERE name=?",
00028         -1,
00029         &ppStmt,
00030         nullptr);
00031     if (result != SQLITE_OK)
00032     {
00033         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00034     }
00035
00036     result = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_STATIC);
00037     switch (result)
00038     {
00039         case SQLITE_OK :
00040             break;
00041         case SQLITE_TOOBIG :
00042             throw obelisk::DatabaseSizeException();
00043             break;
00044         case SQLITE_RANGE :
00045             throw obelisk::DatabaseRangeException();
00046             break;
00047         case SQLITE_NOMEM :
00048             throw obelisk::DatabaseMemoryException();
00049             break;
00050         default :
00051             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00052             break;
00053     }
00054
00055     result = sqlite3_step(ppStmt);
00056     switch (result)
00057     {
00058         case SQLITE_DONE :
00059             // no rows in the database
00060             break;
00061         case SQLITE_ROW :
00062             setId(sqlite3_column_int(ppStmt, 0));
00063             setName((char*) sqlite3_column_text(ppStmt, 1));
00064             break;
00065         case SQLITE_BUSY :
00066             throw obelisk::DatabaseBusyException();
00067             break;
00068         case SQLITE_MISUSE :
00069             throw obelisk::DatabaseMisuseException();
00070             break;
00071         default :
00072             throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00073             break;
00074     }
00075
00076     result = sqlite3_finalize(ppStmt);
00077     if (result != SQLITE_OK)
00078     {
00079         throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
00080     }
00081 }
```

Referenced by obelisk::KnowledgeBase::getVerb().

Here is the caller graph for this function:



5.26.3.6 setId()

```
void obelisk::Verb::setId (
            int id )
```

Set the ID of the Verb.

Parameters

in	*id*	Set the ID of the Verb.

Definition at line 155 of file verb.cpp.

```
00156 {
00157     id_ = id;
00158 }
```

5.26.3.7 setName()

```
void obelisk::Verb::setName (
            std::string name )
```

Set the name of the Verb.

Parameters

in	*name*	The Verb name.

Definition at line 165 of file verb.cpp.

```
00166 {
00167     name_ = name;
00168 }
```

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

- src/lib/include/models/verb.h
- src/lib/models/verb.cpp

Index