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

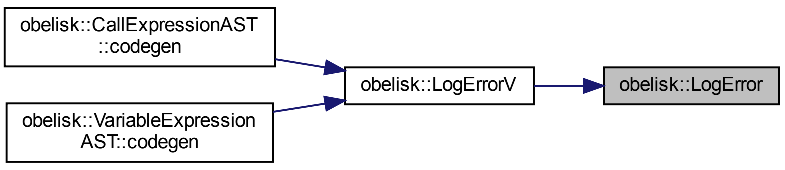
std::unique_ptr<ExpressionAST> Returns the AST expression that caused the error.

Definition at line 3 of file error.cpp.

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

Referenced by LogErrorV().

Here is the caller graph for this function:


obelisk::CallExpressionAST ::codegen
obelisk::VariableExpression AST::codegen
obelisk::LogErrorV
obelisk::LogError

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.

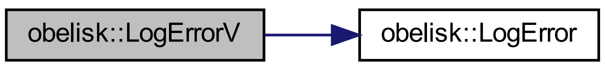
Definition at line 9 of file error.cpp.

```
{
    LogError(Str);
    return nullptr;
}
```

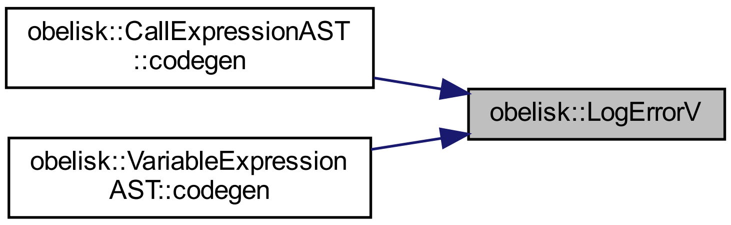
References LogError().

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:


obelisk::CallExpressionAST ::codegen
obelisk::VariableExpression AST::codegen
obelisk::LogErrorV

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

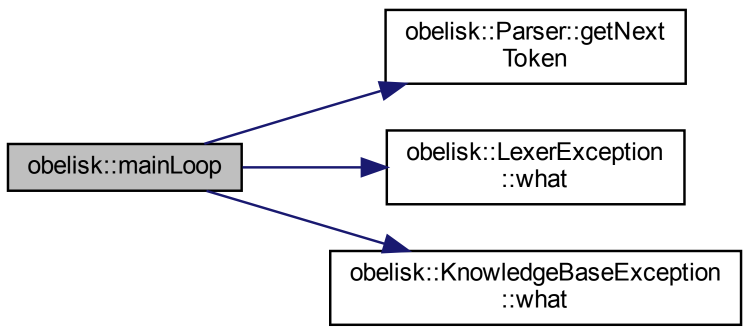
int Returns EXIT_SUCCESS or EXIT_FAILURE.

Definition at line 13 of file main.cpp.

```
{
    std::unique_ptr<obelisk::KnowledgeBase> kb;

    try
    {
        kb = std::unique_ptr<obelisk::KnowledgeBase> {
            new obelisk::KnowledgeBase(kbFile.c_str())};
    }
    catch (obelisk::KnowledgeBaseException& exception)
    {
        std::cout « exception.what() « std::endl;
        return EXIT_FAILURE;
    }

    size_t file = 0;
    std::shared_ptr<obelisk::Lexer> lexer;
    try
    {
        lexer = std::shared_ptr<obelisk::Lexer> {
            new obelisk::Lexer(sourceFiles[file++])};
    }
    catch (obelisk::LexerException& exception)
    {
        std::cout « exception.what() « std::endl;
        return EXIT_FAILURE;
    }
    auto parser = std::unique_ptr<obelisk::Parser> {new obelisk::Parser(lexer)};

    // prime the first token
    try
    {
        parser->getNextToken();
    }
    catch (obelisk::LexerException& exception)
    {
        std::cout « "Error: " « exception.what() « std::endl;
        return EXIT_FAILURE;
    }

    while (true)
    {
        switch (parser->getCurrentToken())
        {
            case obelisk::Lexer::kTokenEof :
                // end of source file found, create a new lexer and pass it to
                // the parser to use
                if (file >= sourceFiles.size())
                {
                    return EXIT_SUCCESS;
                }
                try
                {
                    lexer = std::shared_ptr<obelisk::Lexer> {
                        new obelisk::Lexer(sourceFiles[file++])};
                    parser->setLexer(lexer);
                    // prime the first token in the parser
                    parser->getNextToken();
                }
                catch (obelisk::LexerException& exception)
                {
                    std::cout « exception.what() « std::endl;
                    return EXIT_FAILURE;
                }
                break;
            case ';' :
                // semicolon found, the end of a statement
                try
                {
                    parser->getNextToken();
                }
                catch (obelisk::LexerException& exception)
                {
                    std::cout « "Error: " « exception.what() « std::endl;
                    return EXIT_FAILURE;
                }
                break;
            case obelisk::Lexer::kTokenFact :
                try
                {
                    parser->handleFact(kb);
                }
                catch (obelisk::ParserException& exception)
                {
                    std::cout « "Error: " « exception.what() « std::endl;
                    return EXIT_FAILURE;
                }
                break;
            case obelisk::Lexer::kTokenRule :
                try
                {
                    parser->handleRule(kb);
                }
                catch (obelisk::ParserException& exception)
                {
                    std::cout « "Error: " « exception.what() « std::endl;
                    return EXIT_FAILURE;
                }
                break;
            case obelisk::Lexer::kTokenAction :
                try
                {
                    parser->handleAction(kb);
                }
                catch (obelisk::ParserException& exception)
                {
                    std::cout « "Error: " « exception.what() « std::endl;
                    return EXIT_FAILURE;
                }
                break;
            default :
                parser->getNextToken();
                break;
        }
    }

    return EXIT_SUCCESS;
}
```

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:


obelisk::mainLoop
obelisk::Parser::getNext Token
obelisk::LexerException ::what
obelisk::KnowledgeBaseException ::what

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

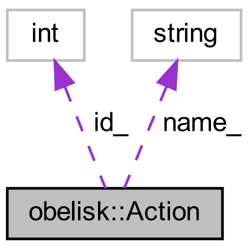
Class Documentation

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.

```
                                 :
            id_(id),
            name_("")
        {
        }
```

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.

```
                                         :
            id_(0),
            name_(name)
        {
        }
```

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.

```
                                         :
                     id_(id),
                     name_(name)
                 {
                 }
```

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

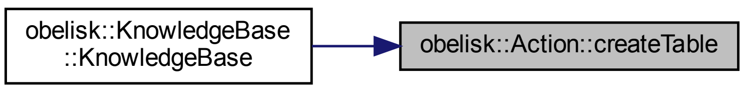
const char* Returns the query used to create the table.

Definition at line 4 of file action.cpp.

```
{
    return R"(
        CREATE TABLE "action" (
            "id"   INTEGER NOT NULL UNIQUE,
            "name" TEXT NOT NULL CHECK(trim(name) != ”) UNIQUE,
            PRIMARY KEY("id" AUTOINCREMENT)
        );
    )";
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::KnowledgeBase
obelisk::Action::createTable

5.1.3.2 getId()

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

Get the ID of the Action.

Returns

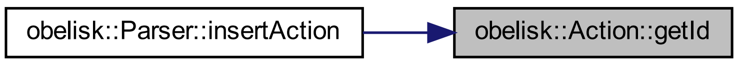
int& Returns the ID.

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

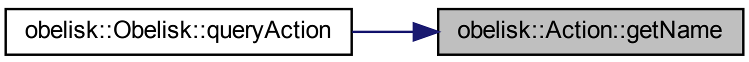
std::string& The Action name.

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.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "INSERT INTO action (name) VALUES (?)",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result
        = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_TRANSIENT);
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            setId((int) sqlite3_last_insert_rowid(dbConnection));
            sqlite3_set_last_insert_rowid(dbConnection, 0);
            break;
        case SQLITE_CONSTRAINT :
            throw obelisk::DatabaseConstraintException(
                sqlite3_errmsg(dbConnection));
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

5.1.3.5 selectByName()

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

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

Parameters

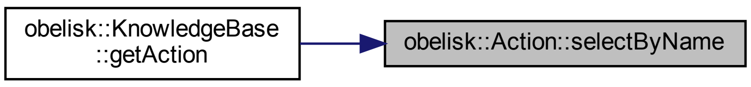
in	*dbConnection*	The database connection to use.

Definition at line 15 of file action.cpp.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "SELECT id, name FROM action WHERE name=?",
        -1,
        &ppStmt,
        nullptr);

    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_STATIC);
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            // no rows in the database
            break;
        case SQLITE_ROW :
            setId(sqlite3_column_int(ppStmt, 0));
            setName((char*) sqlite3_column_text(ppStmt, 1));
            break;
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);

    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::getAction
obelisk::Action::selectByName

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.

```
{
    id_ = id;
}
```

5.1.3.7 setName()

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

Set the name of the Action.

Parameters

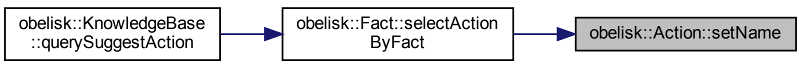
in	*name*	The name of the Action.

Definition at line 165 of file action.cpp.

```
{
    name_ = name;
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::querySuggestAction
obelisk::Fact::selectAction ByFact
obelisk::Action::setName

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

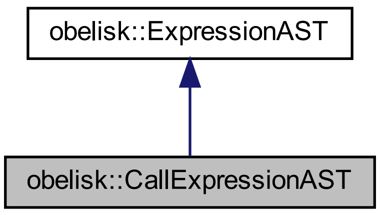
- src/lib/include/models/action.h
- src/lib/models/action.cpp

5.2 obelisk::CallExpressionAST Class Reference

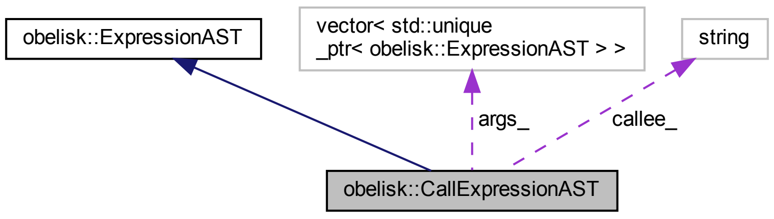
The call AST expression node used to call functions.

```
#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.

```
                                                                    :
                    callee_(callee),
                    args_(std::move(args))
                {
                }
```

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∗

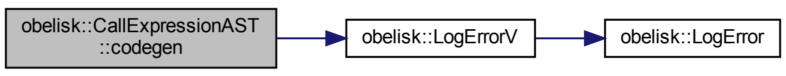
Implements obelisk::ExpressionAST.

Definition at line 5 of file call_expression_ast.cpp.

```
{
    // Look up the name in the global module table.
    llvm::Function *calleeF = TheModule->getFunction(callee_);
    if (!calleeF)
    {
        return LogErrorV("Unknown function referenced");
    }

    // If argument mismatch error.
    if (calleeF->arg_size() != args_.size())
    {
        return LogErrorV("Incorrect # arguments passed");
    }

    std::vector<llvm::Value *> argsV;
    for (unsigned i = 0, e = args_.size(); i != e; ++i)
    {
        argsV.push_back(args_[i]->codegen());
        if (!argsV.back())
        {
            return nullptr;
        }
    }

    return Builder->CreateCall(calleeF, argsV, "calltmp");
}
```

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:

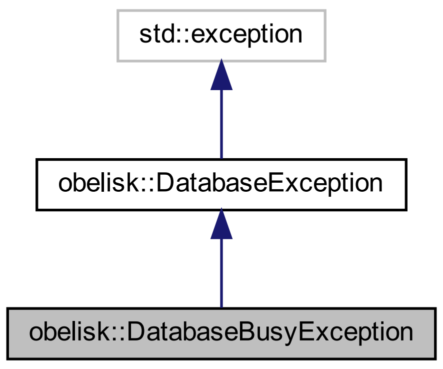
- src/ast/call_expression_ast.h
- src/ast/call_expression_ast.cpp

5.3 obelisk::DatabaseBusyException Class Reference

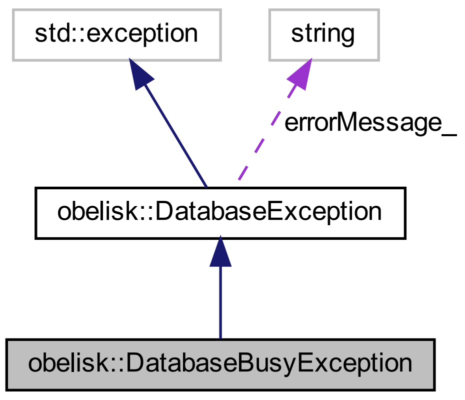
Exception thrown if the database was busy.

```
#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.

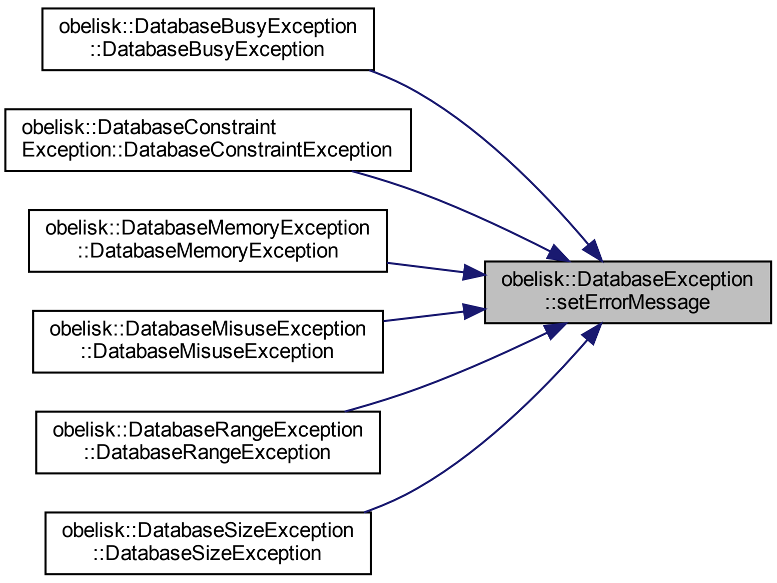
Definition at line 69 of file error.h.

```
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:


obelisk::DatabaseBusyException ::DatabaseBusyException
obelisk::DatabaseConstraint Exception::DatabaseConstraintException
obelisk::DatabaseMemoryException ::DatabaseMemoryException
obelisk::DatabaseMisuseException ::DatabaseMisuseException
obelisk::DatabaseRangeException ::DatabaseRangeException
obelisk::DatabaseSizeException ::DatabaseSizeException
obelisk::DatabaseException ::setErrorMessage

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.

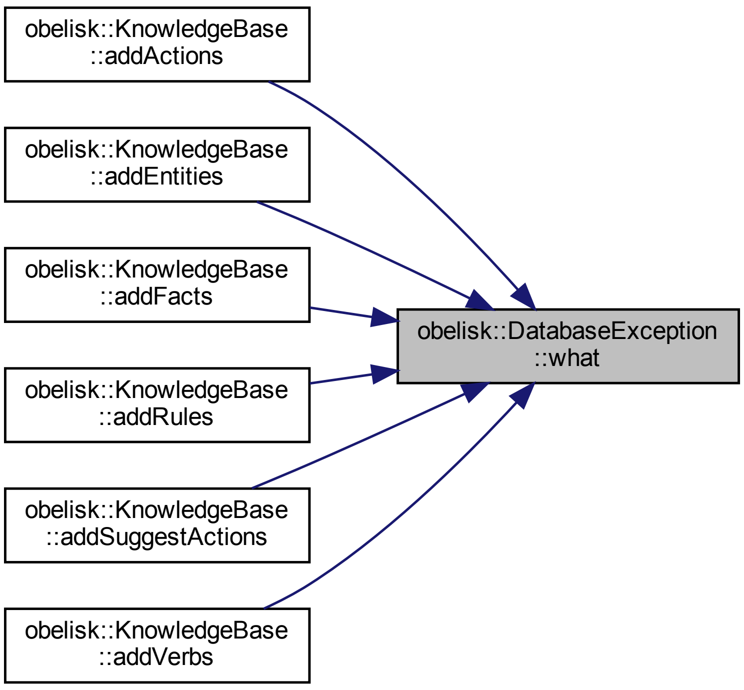
Definition at line 59 of file error.h.

```
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:


obelisk::KnowledgeBase ::addActions
obelisk::KnowledgeBase ::addEntities
obelisk::KnowledgeBase ::addFacts
obelisk::KnowledgeBase ::addRules
obelisk::KnowledgeBase ::addSuggestActions
obelisk::KnowledgeBase ::addVerbs
obelisk::DatabaseException ::what

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

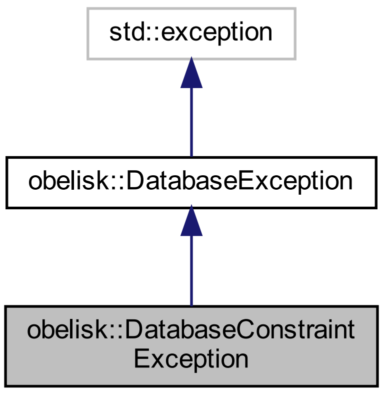
- src/lib/models/error.h

5.4 obelisk::DatabaseConstraintException Class Reference

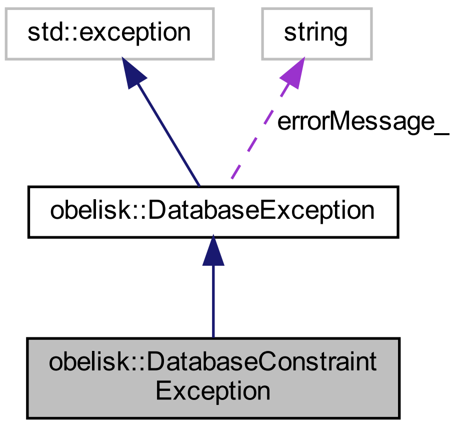
Exception thrown if a constraint was violated.

`#include <error.h>`

Inheritance diagram for obelisk::DatabaseConstraintException:


std::exception
obelisk::DatabaseException
obelisk::DatabaseConstraint Exception

Collaboration diagram for obelisk::DatabaseConstraintException:


std::exception
string
errorMessage_
obelisk::DatabaseException
obelisk::DatabaseConstraint Exception

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

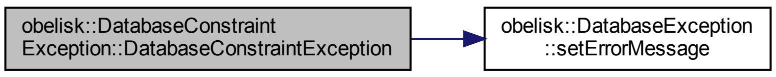
in	*errorMessage*	The error message to send when the constraint is violated.

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:


obelisk::DatabaseConstraint
Exception::DatabaseConstraintException
obelisk::DatabaseException
::setErrorMessage

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.

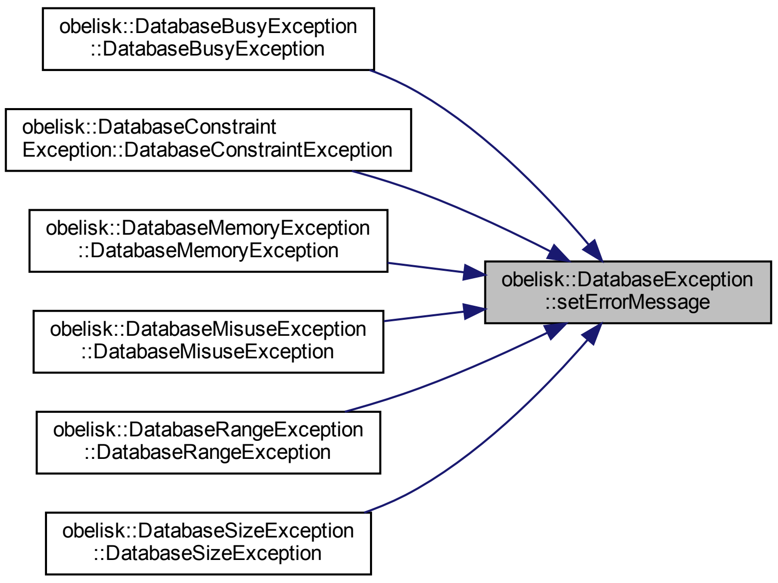
Definition at line 69 of file error.h.

```
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:


obelisk::DatabaseBusyException ::DatabaseBusyException
obelisk::DatabaseConstraint Exception::DatabaseConstraintException
obelisk::DatabaseMemoryException ::DatabaseMemoryException
obelisk::DatabaseMisuseException ::DatabaseMisuseException
obelisk::DatabaseRangeException ::DatabaseRangeException
obelisk::DatabaseSizeException ::DatabaseSizeException
obelisk::DatabaseException ::setErrorMessage

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.

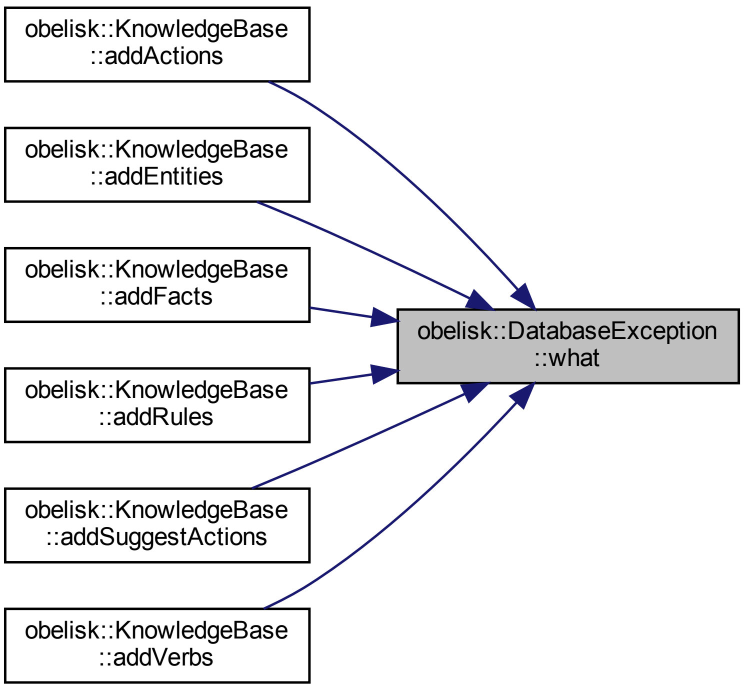
Definition at line 59 of file error.h.

```
            {
                return errorMessage_.c_str();
            }
```

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:


obelisk::KnowledgeBase ::addActions
obelisk::KnowledgeBase ::addEntities
obelisk::KnowledgeBase ::addFacts
obelisk::KnowledgeBase ::addRules
obelisk::KnowledgeBase ::addSuggestActions
obelisk::KnowledgeBase ::addVerbs
obelisk::DatabaseException ::what

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

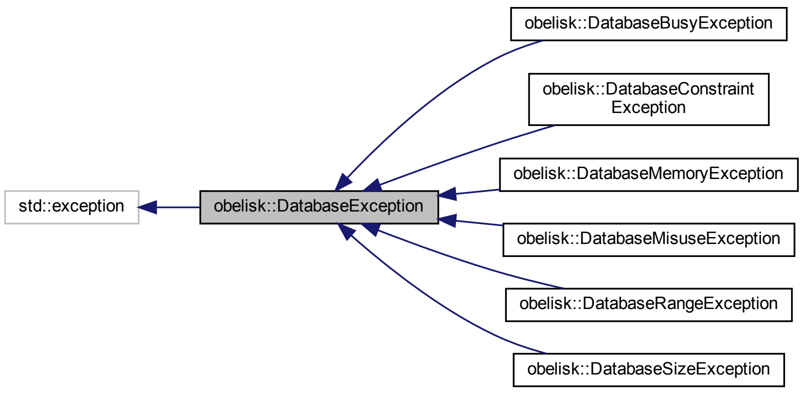
- src/lib/models/error.h

5.5 obelisk::DatabaseException Class Reference

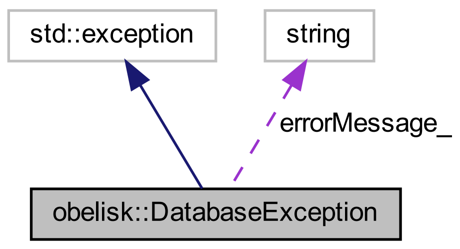
Exception thrown by database models.

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

Inheritance diagram for obelisk::DatabaseException:


obelisk::DatabaseBusyException
obelisk::DatabaseConstraint Exception
obelisk::DatabaseMemoryException
std::exception
obelisk::DatabaseException
obelisk::DatabaseMisuseException
obelisk::DatabaseRangeException
obelisk::DatabaseSizeException

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.

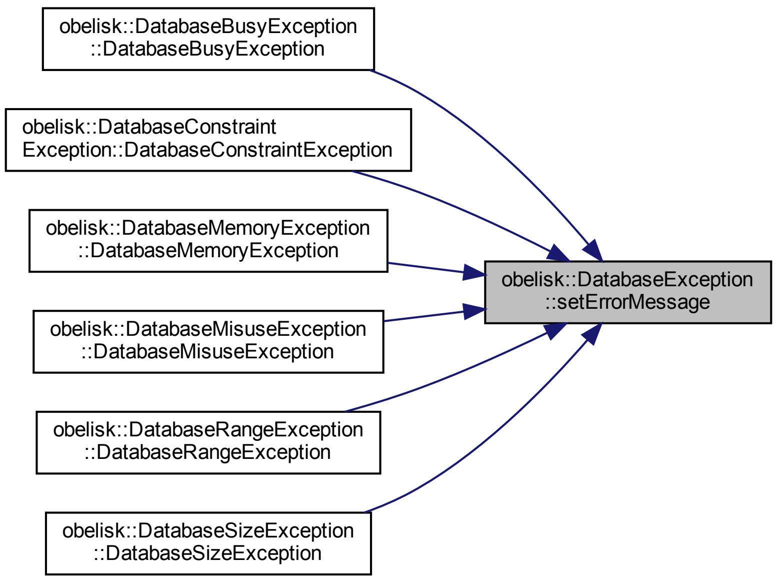
Definition at line 69 of file error.h.

```
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:


obelisk::DatabaseBusyException ::DatabaseBusyException
obelisk::DatabaseConstraint Exception::DatabaseConstraintException
obelisk::DatabaseMemoryException ::DatabaseMemoryException
obelisk::DatabaseMisuseException ::DatabaseMisuseException
obelisk::DatabaseRangeException ::DatabaseRangeException
obelisk::DatabaseSizeException ::DatabaseSizeException
obelisk::DatabaseException ::setErrorMessage

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

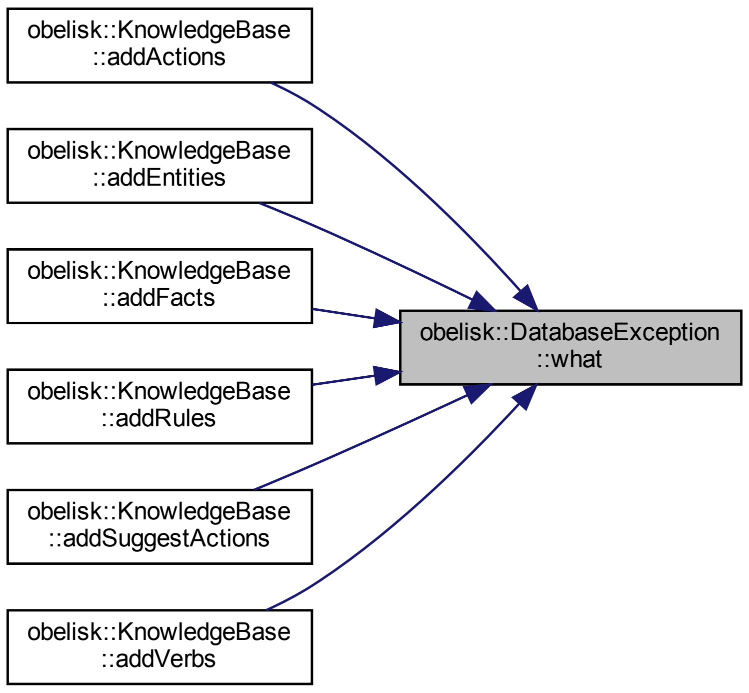
const char* The error message.

Definition at line 59 of file error.h.
```
            {
                return errorMessage_.c_str();
            }
```

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:


obelisk::KnowledgeBase ::addActions
obelisk::KnowledgeBase ::addEntities
obelisk::KnowledgeBase ::addFacts
obelisk::KnowledgeBase ::addRules
obelisk::KnowledgeBase ::addSuggestActions
obelisk::KnowledgeBase ::addVerbs
obelisk::DatabaseException ::what

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

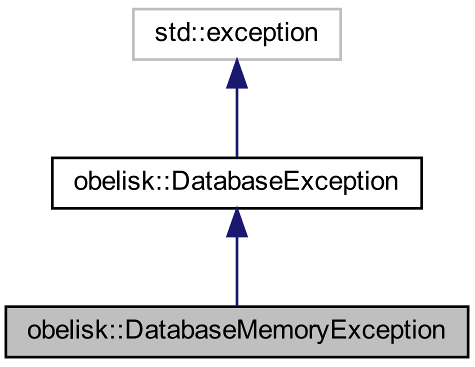
- src/lib/models/error.h

5.6 obelisk::DatabaseMemoryException Class Reference

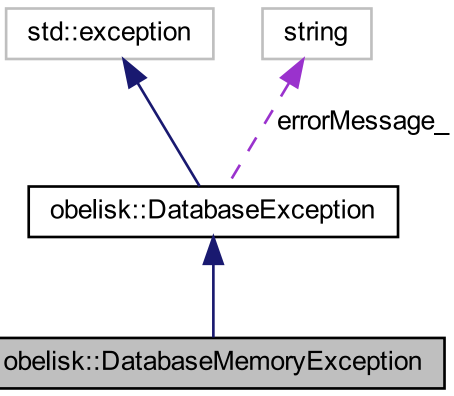
Exception thrown if there is not enough memory to perform the operation.

```
#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.

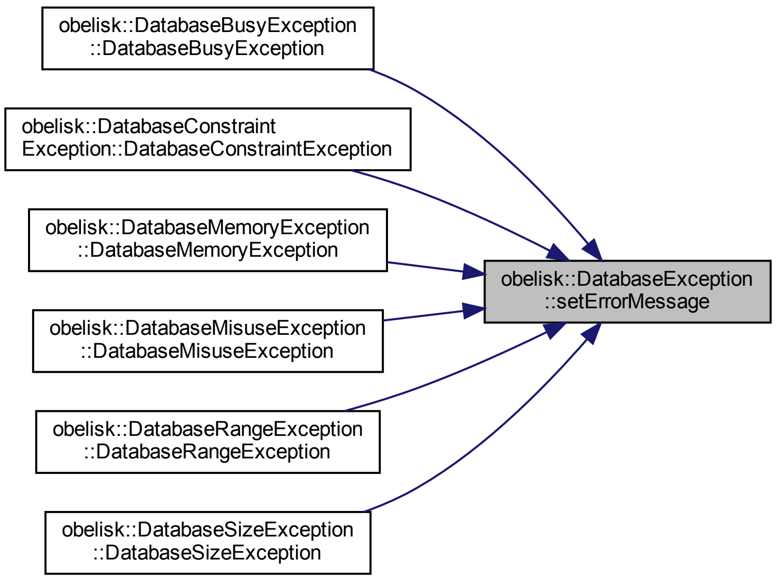
Definition at line 69 of file error.h.

```
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:


obelisk::DatabaseBusyException ::DatabaseBusyException
obelisk::DatabaseConstraint Exception::DatabaseConstraintException
obelisk::DatabaseMemoryException ::DatabaseMemoryException
obelisk::DatabaseMisuseException ::DatabaseMisuseException
obelisk::DatabaseRangeException ::DatabaseRangeException
obelisk::DatabaseSizeException ::DatabaseSizeException
obelisk::DatabaseException ::setErrorMessage

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.

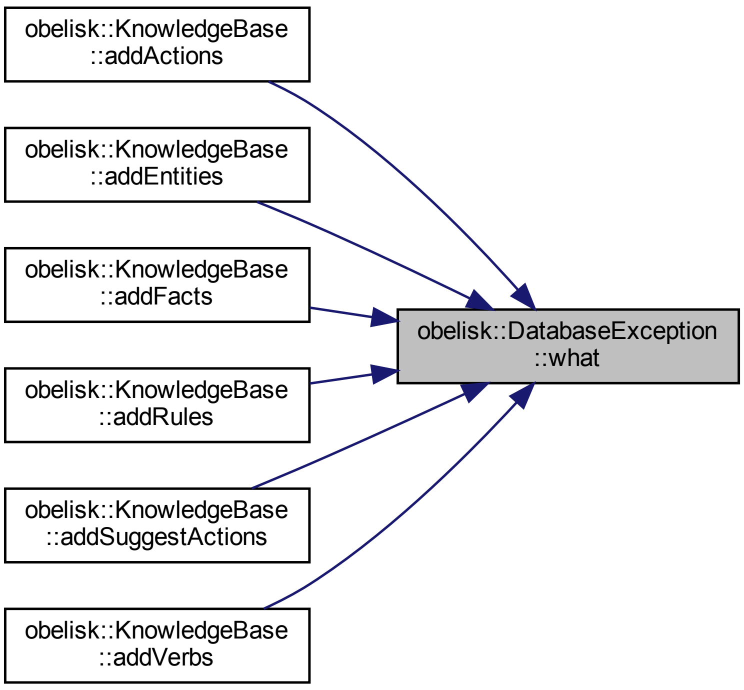
Definition at line 59 of file error.h.

```
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:


obelisk::KnowledgeBase ::addActions
obelisk::KnowledgeBase ::addEntities
obelisk::KnowledgeBase ::addFacts
obelisk::KnowledgeBase ::addRules
obelisk::KnowledgeBase ::addSuggestActions
obelisk::KnowledgeBase ::addVerbs
obelisk::DatabaseException ::what

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

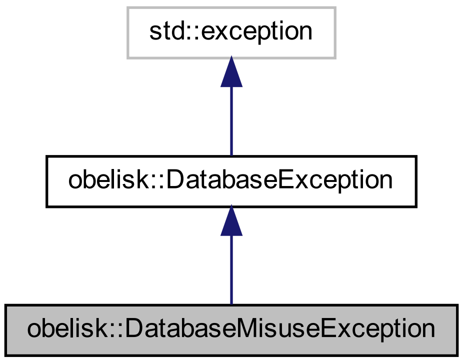
- src/lib/models/error.h

5.7 obelisk::DatabaseMisuseException Class Reference

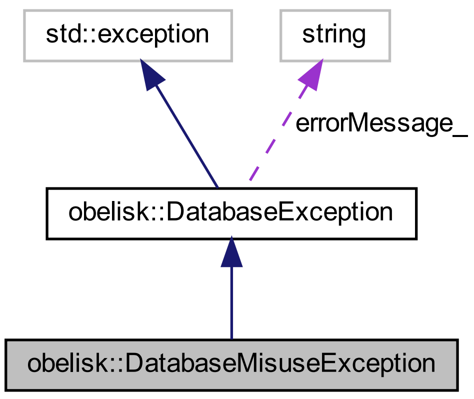
Exception thrown if there is a misuse of the databse.

`#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.

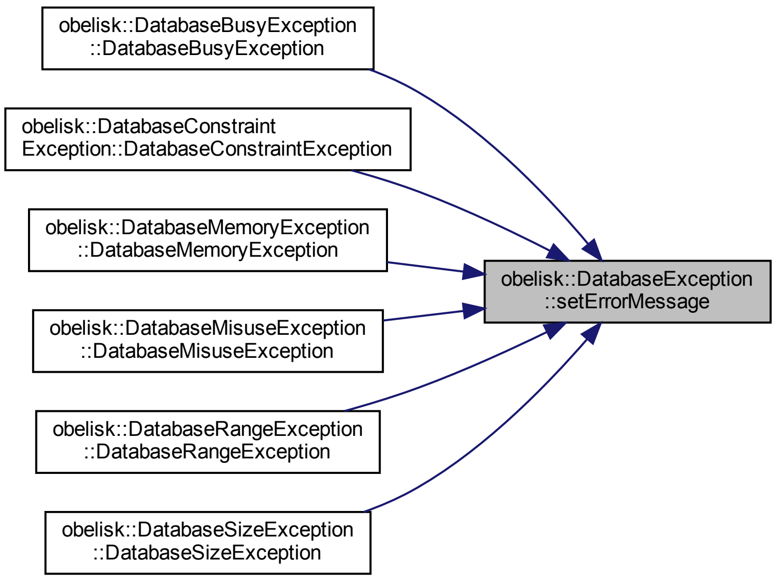
Definition at line 69 of file error.h.

```
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:


obelisk::DatabaseBusyException ::DatabaseBusyException
obelisk::DatabaseConstraint Exception::DatabaseConstraintException
obelisk::DatabaseMemoryException ::DatabaseMemoryException
obelisk::DatabaseMisuseException ::DatabaseMisuseException
obelisk::DatabaseRangeException ::DatabaseRangeException
obelisk::DatabaseSizeException ::DatabaseSizeException
obelisk::DatabaseException ::setErrorMessage

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.

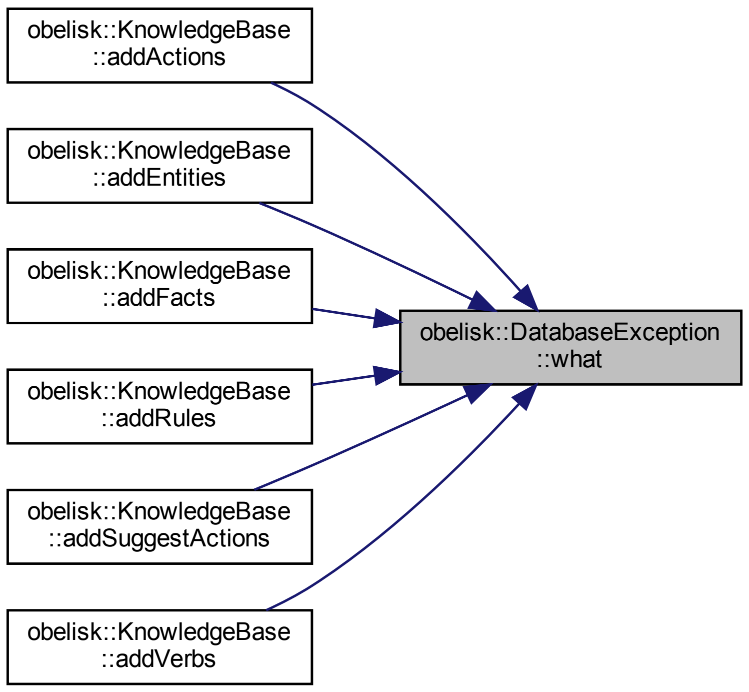
Definition at line 59 of file error.h.

```
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:


obelisk::KnowledgeBase ::addActions
obelisk::KnowledgeBase ::addEntities
obelisk::KnowledgeBase ::addFacts
obelisk::KnowledgeBase ::addRules
obelisk::KnowledgeBase ::addSuggestActions
obelisk::KnowledgeBase ::addVerbs
obelisk::DatabaseException ::what

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

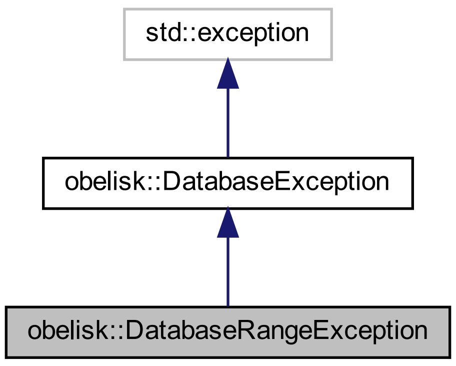
- src/lib/models/error.h

5.8 obelisk::DatabaseRangeException Class Reference

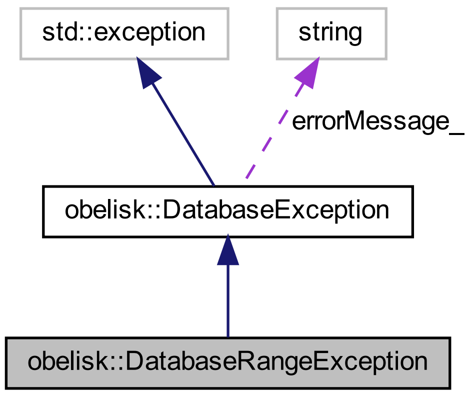
Exception thrown if the index used it out of range.

```
#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.

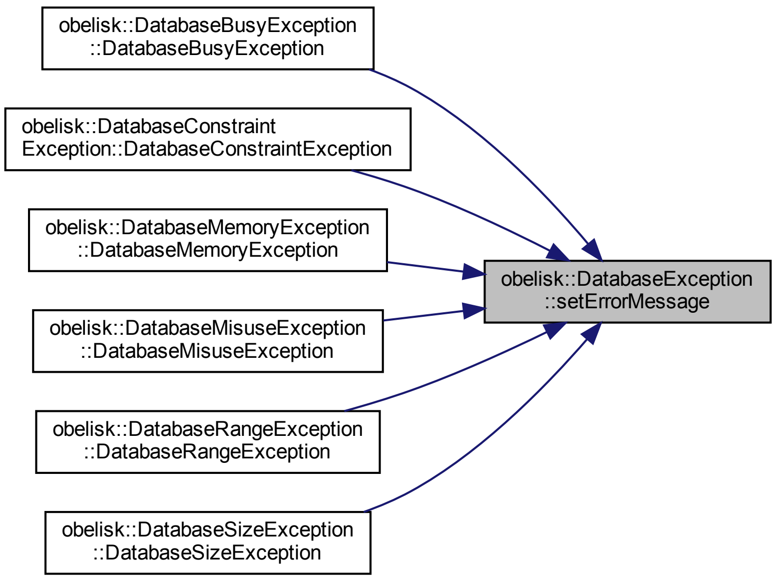
```
                {
                    errorMessage_ = errorMessage;
```

```
               }
```

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:


obelisk::DatabaseBusyException ::DatabaseBusyException
obelisk::DatabaseConstraint Exception::DatabaseConstraintException
obelisk::DatabaseMemoryException ::DatabaseMemoryException
obelisk::DatabaseMisuseException ::DatabaseMisuseException
obelisk::DatabaseRangeException ::DatabaseRangeException
obelisk::DatabaseSizeException ::DatabaseSizeException
obelisk::DatabaseException ::setErrorMessage

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.

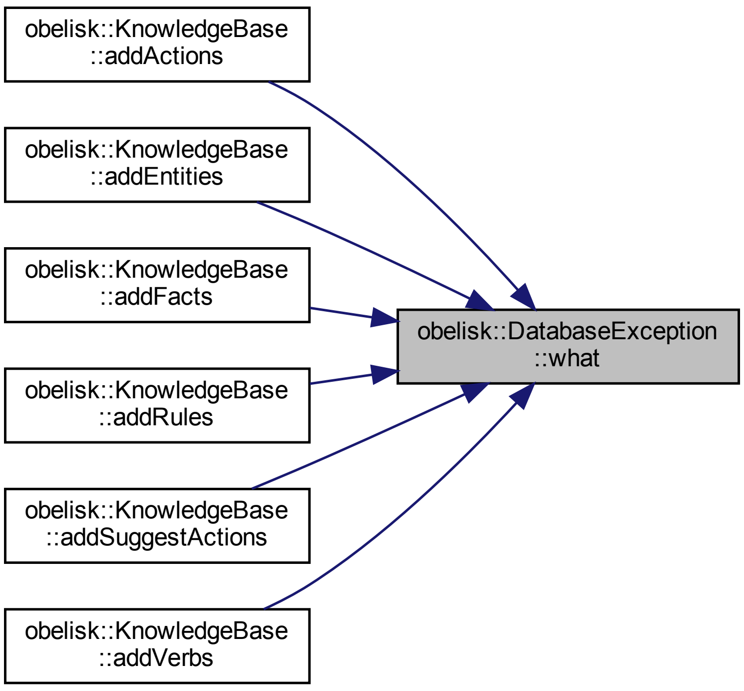
Definition at line 59 of file error.h.

```
               {
                   return errorMessage_.c_str();
               }
```

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:


obelisk::KnowledgeBase ::addActions
obelisk::KnowledgeBase ::addEntities
obelisk::KnowledgeBase ::addFacts
obelisk::KnowledgeBase ::addRules
obelisk::KnowledgeBase ::addSuggestActions
obelisk::KnowledgeBase ::addVerbs
obelisk::DatabaseException ::what

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

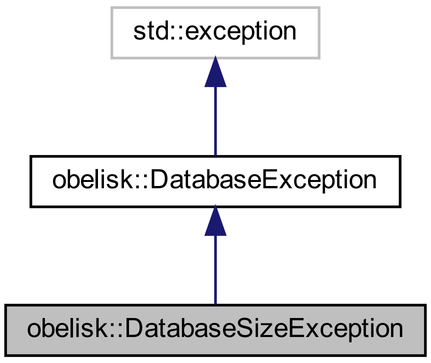
- src/lib/models/error.h

5.9 obelisk::DatabaseSizeException Class Reference

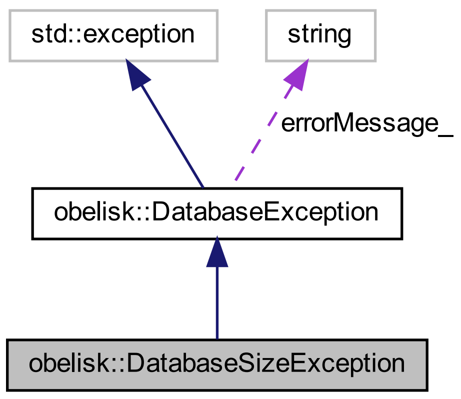
Exception thrown if the string or blob size exceeds sqlite's limits.

```
#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.

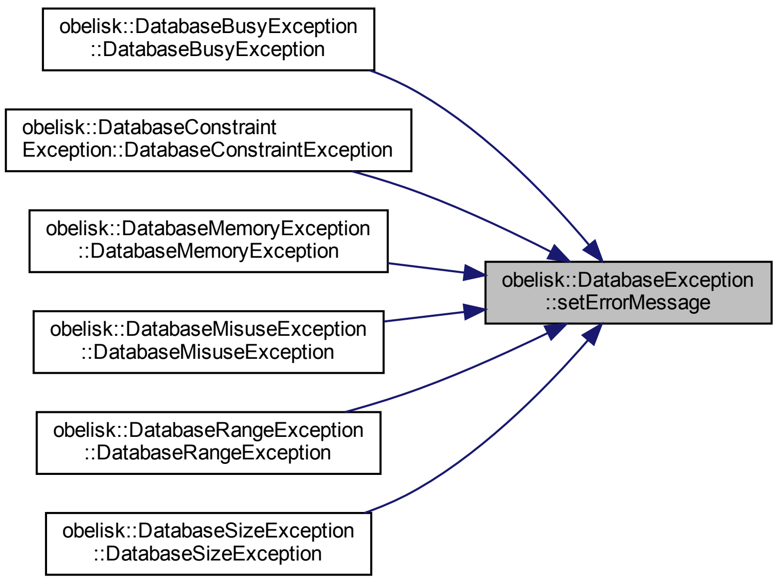
```
00070           {
00071               errorMessage_ = errorMessage;
```

```
        }
```

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:


obelisk::DatabaseBusyException ::DatabaseBusyException
obelisk::DatabaseConstraint Exception::DatabaseConstraintException
obelisk::DatabaseMemoryException ::DatabaseMemoryException
obelisk::DatabaseMisuseException ::DatabaseMisuseException
obelisk::DatabaseRangeException ::DatabaseRangeException
obelisk::DatabaseSizeException ::DatabaseSizeException
obelisk::DatabaseException ::setErrorMessage

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.

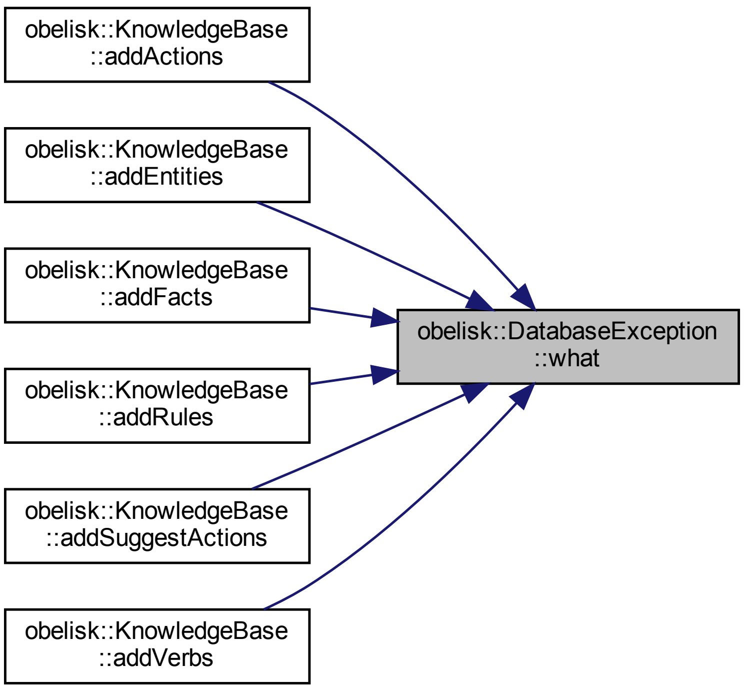
Definition at line 59 of file error.h.

```
        {
            return errorMessage_.c_str();
        }
```

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:


obelisk::KnowledgeBase ::addActions
obelisk::KnowledgeBase ::addEntities
obelisk::KnowledgeBase ::addFacts
obelisk::KnowledgeBase ::addRules
obelisk::KnowledgeBase ::addSuggestActions
obelisk::KnowledgeBase ::addVerbs
obelisk::DatabaseException ::what

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

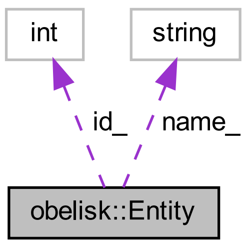
- src/lib/models/error.h

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.

```
                               :
                 id_(0),
                 name_(name)
             {
             }
```

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.

```
                                       :
                 id_(id),
                 name_(name)
             {
             }
```

5.10.3 Member Function Documentation

5.10.3.1 createTable()

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

Create the table in the KnowledgeBase.

Returns

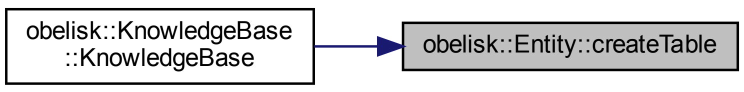
const char∗ Returns the query used to create the table.

Definition at line 4 of file entity.cpp.

```
{
    return R"(
        CREATE TABLE "entity" (
            "id"   INTEGER NOT NULL UNIQUE,
            "name" TEXT NOT NULL CHECK(trim(name) != ”) UNIQUE,
            PRIMARY KEY("id" AUTOINCREMENT)
        );
    )";
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::KnowledgeBase
obelisk::Entity::createTable

5.10.3.2 getId()

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

Get the ID of the Entity.

Returns

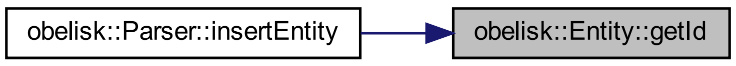
int& Returns the ID.

Definition at line 150 of file entity.cpp.

```
{
    return id_;
}
```

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.

```
{
    return name_;
}
```

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.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "INSERT INTO entity (name) VALUES (?)",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result
        = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_TRANSIENT);
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            setId((int) sqlite3_last_insert_rowid(dbConnection));
            sqlite3_set_last_insert_rowid(dbConnection, 0);
            break;
        case SQLITE_CONSTRAINT :
            throw obelisk::DatabaseConstraintException(
                sqlite3_errmsg(dbConnection));
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

5.10.3.5 selectByName()

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

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

Parameters

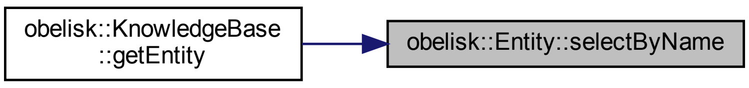
in	*dbConnection*	The database connection to use.

Definition at line 15 of file entity.cpp.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "SELECT id, name FROM entity WHERE name=?",
        -1,
        &ppStmt,
        nullptr);

    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_STATIC);
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            // no rows in the database
            break;
        case SQLITE_ROW :
            setId(sqlite3_column_int(ppStmt, 0));
            setName((char*) sqlite3_column_text(ppStmt, 1));
            break;
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);

    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::getEntity
obelisk::Entity::selectByName

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.

```
{
    id_ = id;
}
```

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.

```
{
    name_ = name;
}
```

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

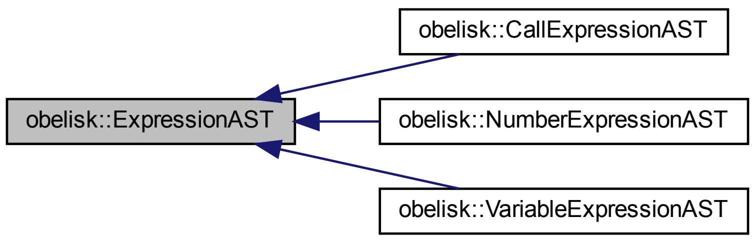
- src/lib/include/models/entity.h
- src/lib/models/entity.cpp

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:


obelisk::CallExpressionAST
obelisk::ExpressionAST
obelisk::NumberExpressionAST
obelisk::VariableExpressionAST

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:

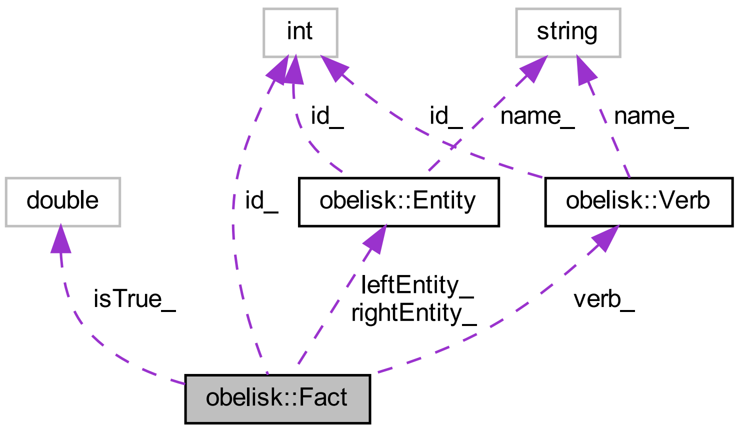
- src/ast/expression_ast.h

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

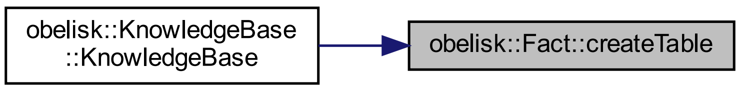
const char∗ Returns the query used to create the table.

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:


obelisk::KnowledgeBase ::KnowledgeBase
obelisk::Fact::createTable

5.12.3.2 getId()

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

Get the ID of the Fact.

Returns

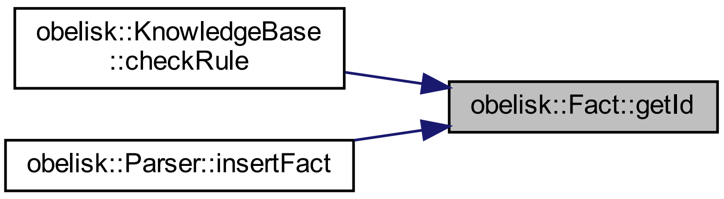
int& Returns the ID.

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:


obelisk::KnowledgeBase ::checkRule
obelisk::Parser::insertFact
obelisk::Fact::getId

5.12.3.3 getIsTrue()

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

Gets the isTrue value.

Returns

true If the Fact is considered true.

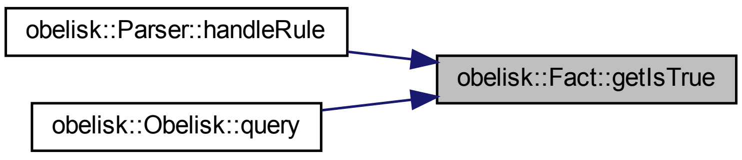
false If the Fact is considered false.

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

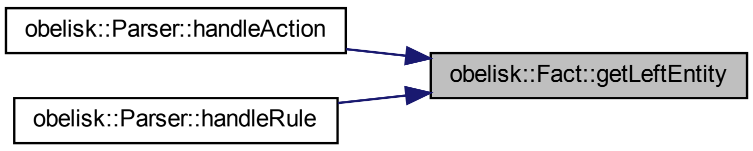
Entity& The left Entity.

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:


obelisk::Parser::handleAction
obelisk::Parser::handleRule
obelisk::Fact::getLeftEntity

5.12.3.5 getRightEntity()

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

Get the right Entity object.

Returns

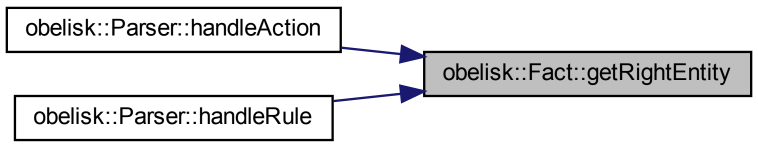
Entity& The right Entity.

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:


obelisk::Parser::handleAction
obelisk::Parser::handleRule
obelisk::Fact::getRightEntity

5.12.3.6 getVerb()

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

Get the Verb object.

Returns

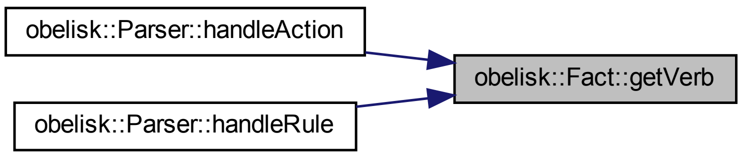
Verb& The Verb.

Definition at line 582 of file fact.cpp.

```
{
    return verb_;
}
```

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.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "INSERT INTO fact (left_entity, right_entity, verb, is_true) VALUES (?, ?, ?, ?)",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_int(ppStmt, 1, getLeftEntity().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_int(ppStmt, 2, getRightEntity().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_int(ppStmt, 3, getVerb().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_int(ppStmt, 4, getIsTrue());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            setId((int) sqlite3_last_insert_rowid(dbConnection));
            sqlite3_set_last_insert_rowid(dbConnection, 0);
            break;
        case SQLITE_CONSTRAINT :
            throw obelisk::DatabaseConstraintException(
                sqlite3_errmsg(dbConnection));
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

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.

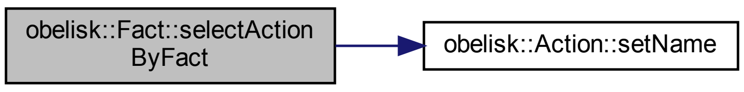
Definition at line 279 of file fact.cpp.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "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 = ?)",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_int(ppStmt, 1, getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            // no rows in the database
            break;
        case SQLITE_ROW :
            action.setName((char*) sqlite3_column_text(ppStmt, 0));
            break;
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

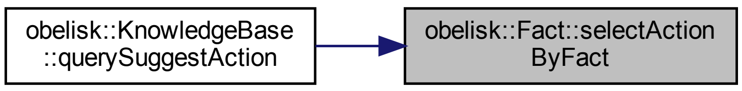
References obelisk::Action::setName().

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

Here is the call graph for this function:



Here is the caller graph for this function:


obelisk::KnowledgeBase ::querySuggestAction
obelisk::Fact::selectAction ByFact

5.12.3.9 selectById()

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

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

Parameters

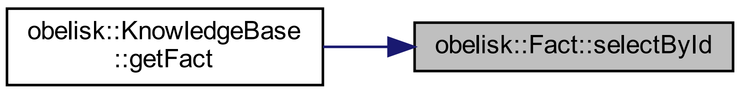
in	*dbConnection*	The database connection to use.

Definition at line 22 of file fact.cpp.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    const char* query;
    if (getId() == 0)
    {
        query
            = "SELECT id, left_entity, right_entity, verb, is_true FROM fact WHERE (left_entity=? AND
right_entity=? AND verb=?)";
    }
    else
    {
        query
            = "SELECT id, left_entity, right_entity, verb, is_true FROM fact WHERE (id=?)";
    }
    auto result = sqlite3_prepare_v2(dbConnection, query, -1, &ppStmt, nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    if (getId() == 0)
    {
        result = sqlite3_bind_int(ppStmt, 1, getLeftEntity().getId());
        switch (result)
        {
            case SQLITE_OK :
                break;
            case SQLITE_TOOBIG :
                throw obelisk::DatabaseSizeException();
                break;
            case SQLITE_RANGE :
                throw obelisk::DatabaseRangeException();
                break;
            case SQLITE_NOMEM :
                throw obelisk::DatabaseMemoryException();
                break;
            default :
                throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
                break;
        }

        result = sqlite3_bind_int(ppStmt, 2, getRightEntity().getId());
        switch (result)
        {
            case SQLITE_OK :
                break;
            case SQLITE_TOOBIG :
                throw obelisk::DatabaseSizeException();
                break;
            case SQLITE_RANGE :
                throw obelisk::DatabaseRangeException();
                break;
            case SQLITE_NOMEM :
                throw obelisk::DatabaseMemoryException();
                break;
            default :
                throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
                break;
        }

        result = sqlite3_bind_int(ppStmt, 3, getVerb().getId());
        switch (result)
        {
            case SQLITE_OK :
                break;
            case SQLITE_TOOBIG :
                throw obelisk::DatabaseSizeException();
                break;
            case SQLITE_RANGE :
                throw obelisk::DatabaseRangeException();
                break;
            case SQLITE_NOMEM :
                throw obelisk::DatabaseMemoryException();
                break;
            default :
                throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
                break;
        }
    }
    else
    {
        result = sqlite3_bind_int(ppStmt, 1, getId());
        switch (result)
        {
            case SQLITE_OK :
                break;
            case SQLITE_TOOBIG :
                throw obelisk::DatabaseSizeException();
                break;
            case SQLITE_RANGE :
                throw obelisk::DatabaseRangeException();
                break;
            case SQLITE_NOMEM :
                throw obelisk::DatabaseMemoryException();
                break;
            default :
                throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
                break;
        }
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            // no rows in the database
            break;
        case SQLITE_ROW :
            setId(sqlite3_column_int(ppStmt, 0));
            getLeftEntity().setId(sqlite3_column_int(ppStmt, 1));
            getRightEntity().setId(sqlite3_column_int(ppStmt, 2));
            getVerb().setId(sqlite3_column_int(ppStmt, 3));
            setIsTrue(sqlite3_column_int(ppStmt, 4));
            break;
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

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

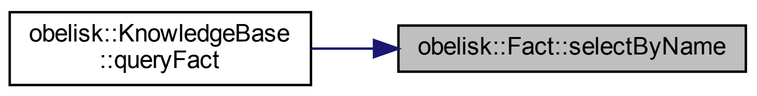
in	*dbConnection*	The database connection to use.

Definition at line 160 of file fact.cpp.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "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=?)",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_text(ppStmt,
        1,
        getLeftEntity().getName().c_str(),
        -1,
        SQLITE_STATIC);
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_text(ppStmt,
        2,
        getVerb().getName().c_str(),
        -1,
        SQLITE_STATIC);
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_text(ppStmt,
        3,
        getRightEntity().getName().c_str(),
        -1,
        SQLITE_STATIC);
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            // no rows in the database
            break;
        case SQLITE_ROW :
            setId(sqlite3_column_int(ppStmt, 0));
            getLeftEntity().setId(sqlite3_column_int(ppStmt, 1));
            getRightEntity().setId(sqlite3_column_int(ppStmt, 2));
            getVerb().setId(sqlite3_column_int(ppStmt, 3));
            setIsTrue(sqlite3_column_int(ppStmt, 4));
            break;
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::queryFact
obelisk::Fact::selectByName

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.

```
{
    id_ = id;
}
```

5.12.3.12 setIsTrue()

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

Set the Fact as true or false.

Parameters

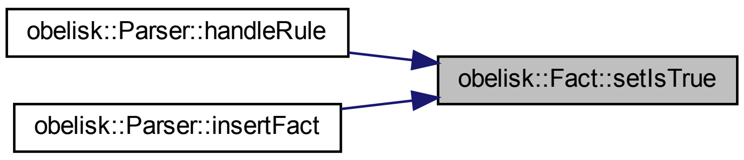
in	*isTrue*	Whether or not the Fact is true.

Definition at line 597 of file fact.cpp.

```
{
    isTrue_ = isTrue;
}
```

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.

```
{
    verb_ = verb;
}
```

5.12.3.16 updateIsTrue()

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

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

Parameters

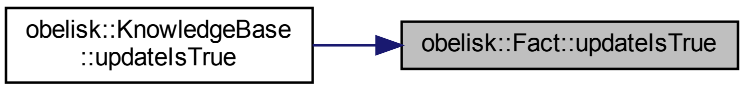
in	*dbConnection*	The database connection.

Definition at line 468 of file fact.cpp.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "UPDATE fact SET is_true=? WHERE id=?",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_int(ppStmt, 1, getIsTrue());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_int(ppStmt, 2, getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            // Row updated
            break;
        case SQLITE_CONSTRAINT :
            throw obelisk::DatabaseConstraintException(
                sqlite3_errmsg(dbConnection));
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::updateIsTrue
obelisk::Fact::updateIsTrue

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

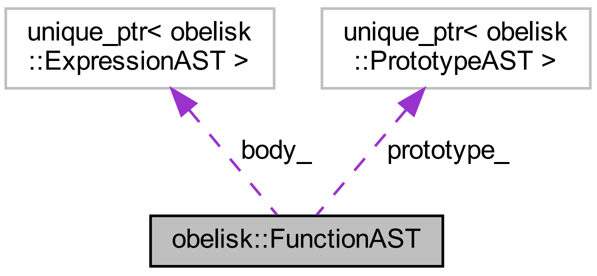
- src/lib/include/models/fact.h
- src/lib/models/fact.cpp

5.13 obelisk::FunctionAST Class Reference

A Funcion AST node.

`#include <function_ast.h>`

Collaboration diagram for obelisk::FunctionAST:


unique_ptr< obelisk ::ExpressionAST >
unique_ptr< obelisk ::PrototypeAST >
body_
prototype_
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.

```
                                                    :
                    prototype_(std::move(prototype)),
                    body_(std::move(body))
                {
                }
```

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.

```
{
    llvm::Function *theFunction = TheModule->getFunction(prototype_->getName());

    if (!theFunction)
    {
        theFunction = prototype_->codegen();
    }

    if (!theFunction)
    {
        return nullptr;
    }

    llvm::BasicBlock *bB
```

```
        = llvm::BasicBlock::Create(*TheContext, "entry", theFunction);
    Builder->SetInsertPoint(bB);

    NamedValues.clear();
    for (auto &arg : theFunction->args())
    {
        NamedValues[std::string(arg.getName())] = &arg;
    }

    if (llvm::Value *retVal = body_->codegen())
    {
        Builder->CreateRet(retVal);
        llvm::verifyFunction(*theFunction);
        return theFunction;
    }

    theFunction->eraseFromParent();
    return nullptr;
}
```

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:

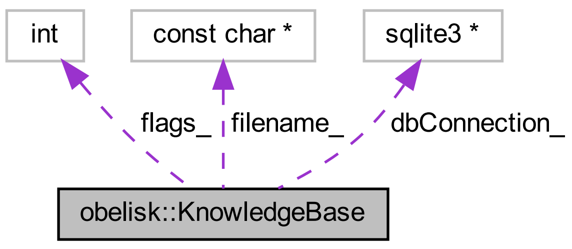
- src/ast/function_ast.h
- src/ast/function_ast.cpp

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

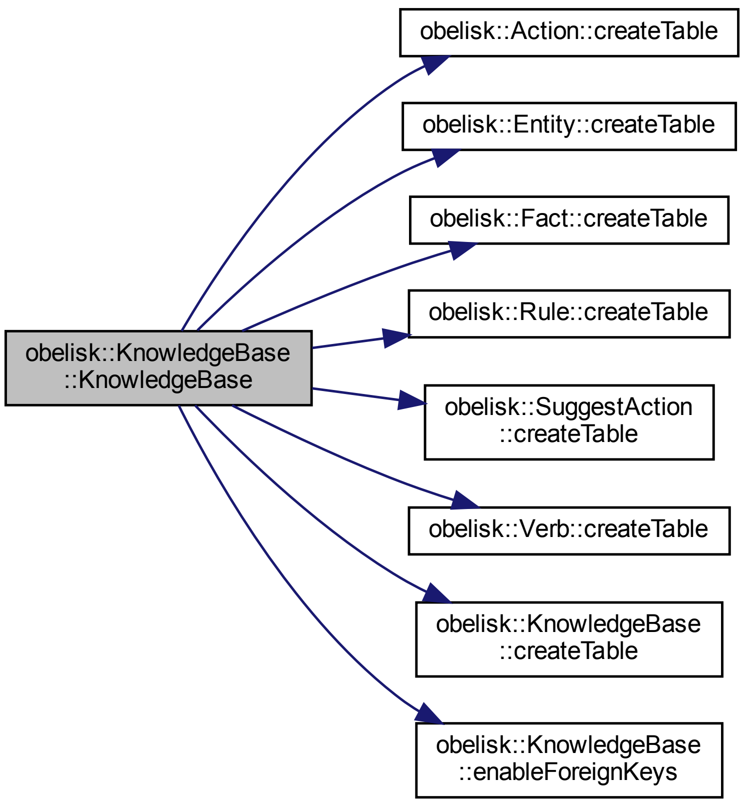
in	*filename*	The name of the file to save the knowledge base as.
in	*flags*	The flags to open the KnowledgeBase with.

Definition at line 9 of file knowledge_base.cpp.

```
{
    filename_ = std::move(filename);
    flags_    = std::move(flags);

    std::filesystem::path path {filename};
    auto dbExists = std::filesystem::exists(path);

    auto result = sqlite3_open_v2(filename, &dbConnection_, flags, NULL);
    if (result != SQLITE_OK)
    {
        throw new KnowledgeBaseException("database could not be opened");
    }

    enableForeignKeys();

    if (!dbExists)
    {
        createTable(obelisk::Action::createTable);
        createTable(obelisk::Entity::createTable);
        createTable(obelisk::Verb::createTable);
        createTable(obelisk::Fact::createTable);
        createTable(obelisk::Rule::createTable);
        createTable(obelisk::SuggestAction::createTable);
    }
}
```

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:


obelisk::KnowledgeBase ::KnowledgeBase
obelisk::Action::createTable
obelisk::Entity::createTable
obelisk::Fact::createTable
obelisk::Rule::createTable
obelisk::SuggestAction ::createTable
obelisk::Verb::createTable
obelisk::KnowledgeBase ::createTable
obelisk::KnowledgeBase ::enableForeignKeys

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.

```
                                                :
                  KnowledgeBase(filename,
                      SQLITE_OPEN_READWRITE | SQLITE_OPEN_CREATE)
              {
              }
```

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.

```
{
    if (dbConnection_)
    {
        sqlite3_close_v2(dbConnection_);
    }
}
```

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

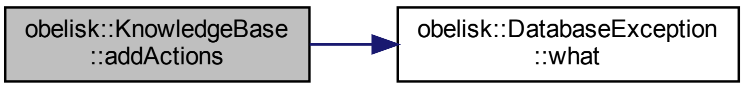
in,out	*actions*	The actions to add. If the insert is successful it will have a row ID, if nto the ID will be 0.

Definition at line 124 of file knowledge_base.cpp.

```
{
    for (auto& action : actions)
    {
        try
        {
            action.insert(dbConnection_);
        }
        catch (obelisk::DatabaseConstraintException& exception)
        {
            // ignore unique constraint error
            if (std::strcmp(exception.what(),
                    "UNIQUE constraint failed: action.name")
                != 0)
            {
                throw;
            }
        }
    }
}
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:


obelisk::KnowledgeBase ::addActions
obelisk::DatabaseException ::what

5.14.3.2 addEntities()

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

Add entities to the KnowledgeBase.

Parameters

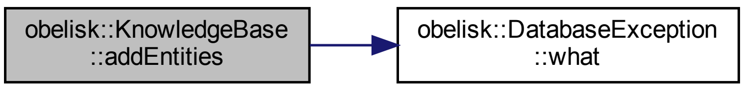
in,out	*entities*	The entities to add. If the insert is successful it will have a row ID, if not the ID will be 0.

Definition at line 82 of file knowledge_base.cpp.

```
{
    for (auto& entity : entities)
    {
        try
        {
            entity.insert(dbConnection_);
        }
        catch (obelisk::DatabaseConstraintException& exception)
        {
            // ignore unique constraint error
            if (std::strcmp(exception.what(),
                    "UNIQUE constraint failed: entity.name")
                != 0)
            {
                throw;
            }
        }
    }
}
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:


obelisk::KnowledgeBase ::addEntities
obelisk::DatabaseException ::what

5.14.3.3 addFacts()

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

Add facts to the KnowledgeBase.

Parameters

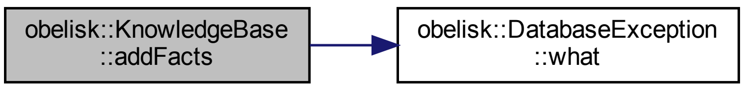
in,out	*facts*	The facts to add. If the insert is successful it will have a row ID, if not the ID will be 0.

Definition at line 145 of file knowledge_base.cpp.

```
{
    for (auto& fact : facts)
    {
        try
        {
            fact.insert(dbConnection_);
        }
        catch (obelisk::DatabaseConstraintException& exception)
        {
            // ignore unique constraint error
            if (std::strcmp(exception.what(),
                    "UNIQUE constraint failed: fact.left_entity, fact.right_entity, fact.verb")
                != 0)
            {
                throw;
            }
        }
    }
}
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:


obelisk::KnowledgeBase ::addFacts
obelisk::DatabaseException ::what

5.14.3.4 addRules()

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

Add rules to the KnowledgeBase.

Parameters

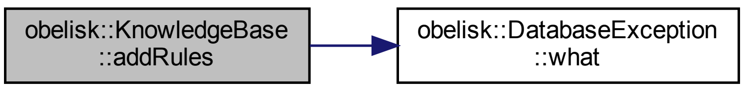
in,out	*rules*	The rules to add. If the insert is successful it will have a row ID, if not the ID will be 0.

Definition at line 188 of file knowledge_base.cpp.

```
{
    for (auto& rule : rules)
    {
        try
        {
            rule.insert(dbConnection_);
        }
        catch (obelisk::DatabaseConstraintException& exception)
        {
            // ignore unique constraint error
            if (std::strcmp(exception.what(),
                    "UNIQUE constraint failed: rule.fact, rule.reason")
                != 0)
            {
                throw;
            }
        }
    }
}
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:


obelisk::KnowledgeBase ::addRules
obelisk::DatabaseException ::what

5.14.3.5 addSuggestActions()

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

Add suggested actions to the KnowledgeBase.

Parameters

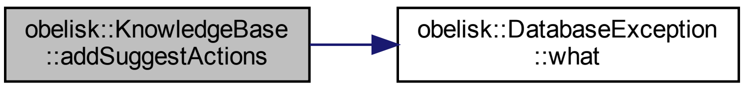
in,out	*suggestActions*	The suggested actions to add. If the insert is successful it will have a row ID, if not the ID will be 0.

Definition at line 166 of file knowledge_base.cpp.

```
{
    for (auto& suggestAction : suggestActions)
    {
        try
        {
            suggestAction.insert(dbConnection_);
        }
        catch (obelisk::DatabaseConstraintException& exception)
        {
            // ignore unique constraint error
            if (std::strcmp(exception.what(),
                    "UNIQUE constraint failed: suggest_action.fact, suggest_action.true_action,
      suggest_action.false_action")
                != 0)
            {
                throw;
            }
        }
    }
}
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:


obelisk::KnowledgeBase ::addSuggestActions
obelisk::DatabaseException ::what

5.14.3.6 addVerbs()

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

Add verbs to the KnowledgeBase.

Parameters

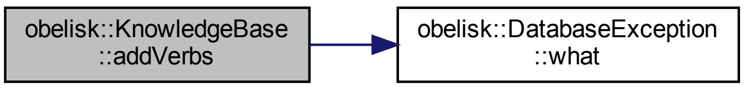
in,out	*verbs*	The verbs to add. If the insert is successful it will have a row ID, if not the ID will be 0.

Definition at line 103 of file knowledge_base.cpp.

```
{
    for (auto& verb : verbs)
    {
        try
        {
            verb.insert(dbConnection_);
        }
        catch (obelisk::DatabaseConstraintException& exception)
        {
            // ignore unique constraint error
            if (std::strcmp(exception.what(),
                    "UNIQUE constraint failed: verb.name")
                != 0)
            {
                throw;
            }
        }
    }
}
```

References obelisk::DatabaseException::what().

Here is the call graph for this function:


obelisk::KnowledgeBase ::addVerbs
obelisk::DatabaseException ::what

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

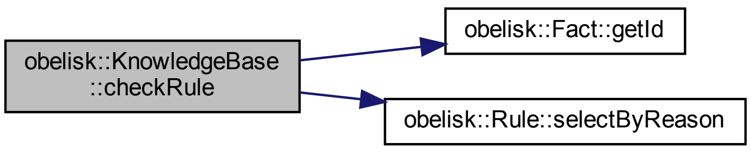
in,out	*fact*	The Fact to check for existing rules.

Definition at line 240 of file knowledge_base.cpp.

```
{
    std::vector<obelisk::Rule> rules;
    obelisk::Rule::selectByReason(dbConnection_, fact.getId(), rules);
    for (auto& rule : rules)
    {
        auto reason = rule.getReason();
        getFact(reason);
        if (reason.getIsTrue() > 0)
        {
            auto updateFact = rule.getFact();
            updateFact.setIsTrue(1.0);
            updateFact.updateIsTrue(dbConnection_);
            checkRule(updateFact);
        }
    }
}
```

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

Here is the call graph for this function:


obelisk::KnowledgeBase ::checkRule
obelisk::Fact::getId
obelisk::Rule::selectByReason

5.14.3.8 createTable()

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

Create the tables in the database.

Parameters

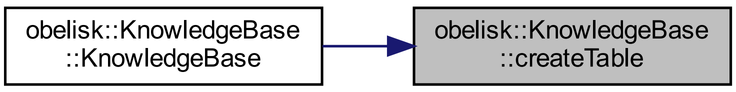
in	*function*	This function is called to create the table.

Definition at line 65 of file knowledge_base.cpp.

```
{
    char* errmsg;
    int result = sqlite3_exec(dbConnection_, function(), NULL, NULL, &errmsg);
    if (result != SQLITE_OK)
    {
        if (errmsg)
        {
            throw obelisk::KnowledgeBaseException(errmsg);
        }
        else
        {
            throw obelisk::KnowledgeBaseException();
        }
    }
}
```

Referenced by KnowledgeBase().

Here is the caller graph for this function:


obelisk::KnowledgeBase ::KnowledgeBase
obelisk::KnowledgeBase ::createTable

5.14.3.9 enableForeignKeys()

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

Enable foreign key functionality in the open database.

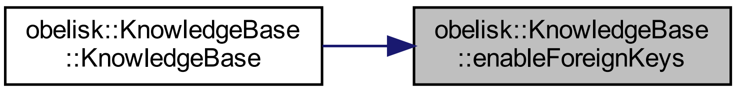
This must always be done when the connection is opened or it will not enforce the foreign key constraints.

Definition at line 44 of file knowledge_base.cpp.

```
{
    char* errmsg;
    int result = sqlite3_exec(dbConnection_,
        "PRAGMA foreign_keys = ON;",
        NULL,
        NULL,
        &errmsg);
    if (result != SQLITE_OK)
    {
        if (errmsg)
        {
            throw obelisk::KnowledgeBaseException(errmsg);
        }
        else
        {
            throw obelisk::KnowledgeBaseException();
        }
    }
}
```

Referenced by KnowledgeBase().

Here is the caller graph for this function:


obelisk::KnowledgeBase ::KnowledgeBase
obelisk::KnowledgeBase ::enableForeignKeys

5.14.3.10 getAction()

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

Get an Action based on the ID it contains.

Parameters

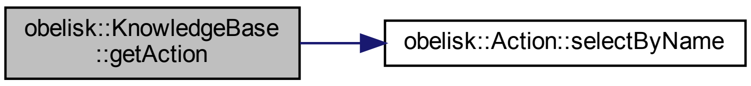
in	*action*	The Action object should contain just the ID and the rest will be filled in.

Definition at line 219 of file knowledge_base.cpp.

```
{
    action.selectByName(dbConnection_);
}
```

References obelisk::Action::selectByName().

Here is the call graph for this function:


obelisk::KnowledgeBase ::getAction
obelisk::Action::selectByName

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.

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

5.14.3.12 getEntity()

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

Get an Entity object based on the ID it contains.

Parameters

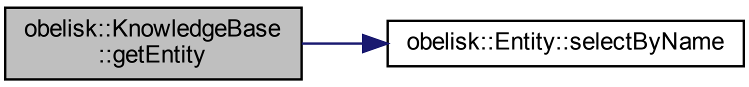
in,out	*entity*	The Entity object should contain just the ID and the rest will be filled in.

Definition at line 209 of file knowledge_base.cpp.

```
{
    entity.selectByName(dbConnection_);
}
```

References obelisk::Entity::selectByName().

Here is the call graph for this function:


obelisk::KnowledgeBase ::getEntity
obelisk::Entity::selectByName

5.14.3.13 getFact()

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

Get a Fact object based on the ID it contains.

Parameters

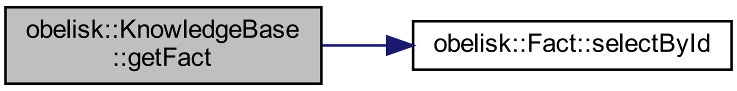
`in,out`	*fact*	The Fact object should contain just the ID and the rest will be filled in.

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.

```
{
    result1 = (float) var;
    result2 = (float) (var - (double) result1);
}
```

5.14.3.15 getRule()

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

Get a Rule based on the ID it contains.

Parameters

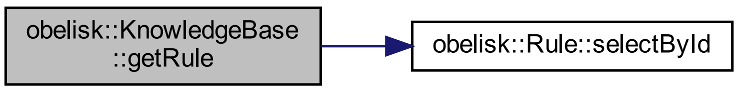
in,out	*rule*	The Rule object should contain just the ID and the rest will be filled in.

Definition at line 235 of file knowledge_base.cpp.

```
{
    rule.selectById(dbConnection_);
}
```

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

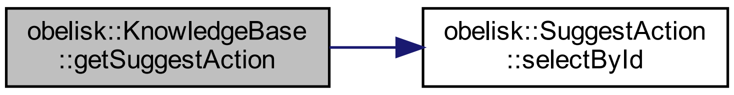
in,out	*suggestAction*	The SuggestAction object should contain just the ID and the rest will be filled in.

Definition at line 229 of file knowledge_base.cpp.

```
{
    suggestAction.selectById(dbConnection_);
}
```

References obelisk::SuggestAction::selectById().

Here is the call graph for this function:


obelisk::KnowledgeBase ::getSuggestAction
obelisk::SuggestAction ::selectById

5.14.3.17 getVerb()

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

Get a Verb object based on the ID it contains.

Parameters

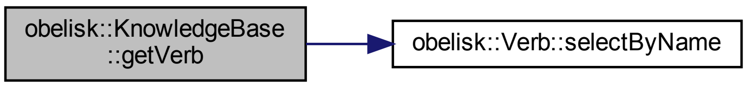
in,out	*verb*	The Verb object should contain just the ID and the rest will be filled in.

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

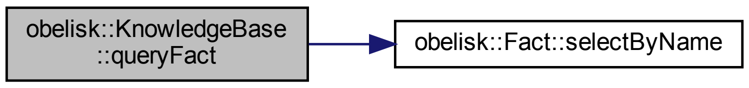
in	*fact*	The Fact to check.

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:


obelisk::KnowledgeBase ::queryFact
obelisk::Fact::selectByName

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

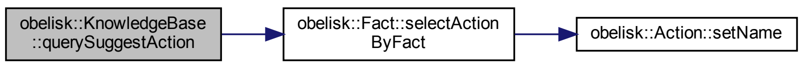
in	*fact*	The Fact to search for.
out	*action*	The Action that is suggested to take.

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:


obelisk::KnowledgeBase ::querySuggestAction
obelisk::Fact::selectAction ByFact
obelisk::Action::setName

5.14.3.20 updateIsTrue()

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

Update the is true field in the KnowledgeBase.

Parameters

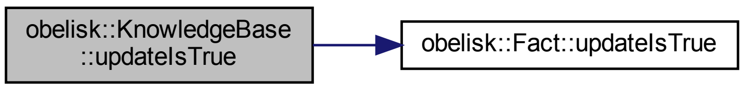
in,out	*fact*	The fact to update.

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:


obelisk::KnowledgeBase ::updateIsTrue
obelisk::Fact::updateIsTrue

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

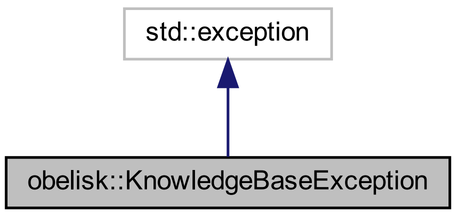
- src/lib/include/knowledge_base.h
- src/lib/knowledge_base.cpp

5.15 obelisk::KnowledgeBaseException Class Reference

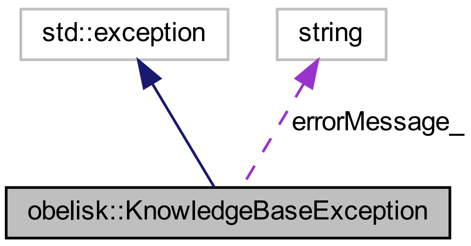
Exception thrown by the KnowledgeBase.

`#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.

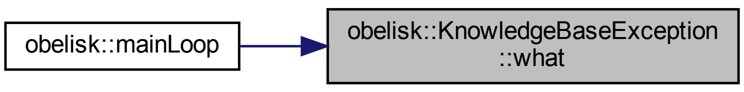
Definition at line 285 of file knowledge_base.h.

```
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:

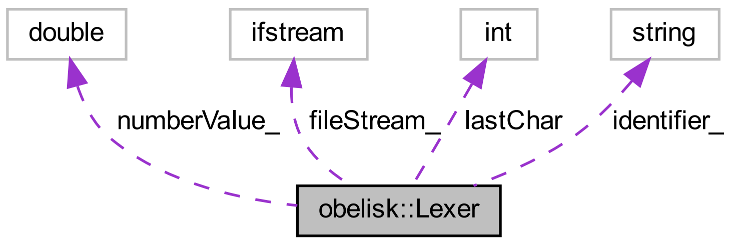
- src/lib/include/knowledge_base.h

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.

```
            {
                kTokenEof = -1,

                kTokenFact = -2,

                kTokenRule = -3,

                kTokenAction = -4,

                kTokenDef = -5,

                kTokenExtern = -6,

                kTokenIdentifier = -7,

                kTokenNumber = -8,

                kTokenString = -9
            };
```

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.

```
{
    fileStream_.open(sourceFile, std::ifstream::in);
    if (!fileStream_)
    {
        throw obelisk::LexerException(
            "could not open source file " + sourceFile);
    }
}
```

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.

```
{
    identifier_ += lastChar;
}
```

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.

```
{
    do
    {
        *lastChar = fileStream_.get();
    }
    while (*lastChar != EOF && *lastChar != '\n' && *lastChar != '\r');
}
```

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.

```
{
    return identifier_;
}
```

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.

```
{
    return numberValue_;
}
```

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.

```
{
    while (isspace(lastChar))
    {
        lastChar = fileStream_.get();
    }

    if (isalpha(lastChar))
    {
        eraseIdentifier();
        appendIdentifier(lastChar);
        while (isalnum((lastChar = fileStream_.get())))
        {
            appendIdentifier(lastChar);
        }

        if (getIdentifier() == "fact")
        {
            return Token::kTokenFact;
        }

        if (getIdentifier() == "rule")
        {
            return Token::kTokenRule;
        }

        if (getIdentifier() == "action")
        {
            return Token::kTokenAction;
        }

        if (getIdentifier() == "def")
        {
            return Token::kTokenDef;
        }

        if (getIdentifier() == "extern")
        {
            return Token::kTokenExtern;
        }

        return Token::kTokenIdentifier;
    }

    if (isdigit(lastChar))
    {
        bool firstPeriod = false;
        std::string numberStr;
        do
        {
            if (firstPeriod && lastChar == '.')
            {
                throw obelisk::LexerException("invalid double value");
            }
            else if (!firstPeriod && lastChar == '.')
            {
                firstPeriod = true;
            }
            numberStr += lastChar;
            lastChar = fileStream_.get();
        }
        while (isdigit(lastChar) || lastChar == '.');

        setNumberValue(strtod(numberStr.c_str(), nullptr));

        return kTokenNumber;
    }

    if (lastChar == '#')
    {
        commentLine(&lastChar);

        if (lastChar != EOF)
        {
            return getToken();
        }
    }
    else if (lastChar == '/')
    {
        lastChar = fileStream_.get();
        if (lastChar == '/')
        {
            commentLine(&lastChar);

            if (lastChar != EOF)
            {
                return getToken();
            }
        }
    }

    if (lastChar == EOF)
    {
        return kTokenEof;
    }

    int thisChar = lastChar;
    lastChar     = fileStream_.get();
    return thisChar;
}
```

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.

```
{
    identifier_ = identifier;
}
```

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:

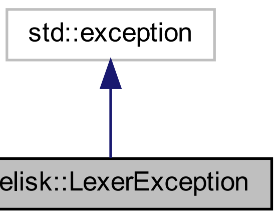
- src/lexer.h
- src/lexer.cpp

5.17 obelisk::LexerException Class Reference

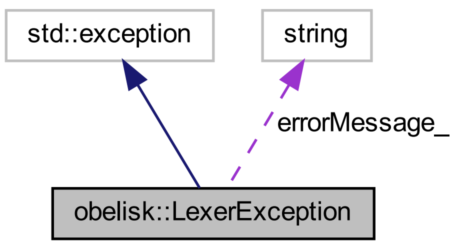
Lexer exception class.

`#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.

```
                                                            :
                errorMessage_(errorMessage)
            {
            }
```

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.

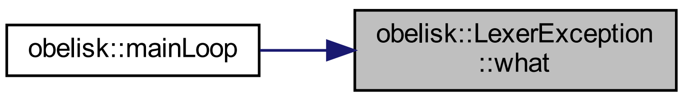
Definition at line 206 of file lexer.h.

```
            {
                return errorMessage_.c_str();
            }
```

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:

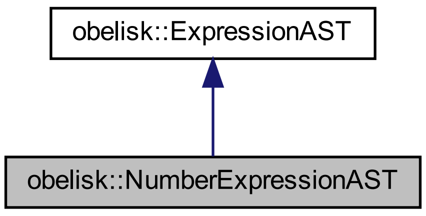
- src/lexer.h

5.18 obelisk::NumberExpressionAST Class Reference

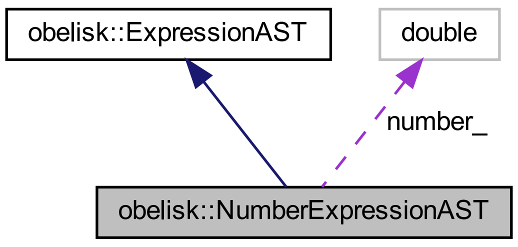
A number expression AST node.

`#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:

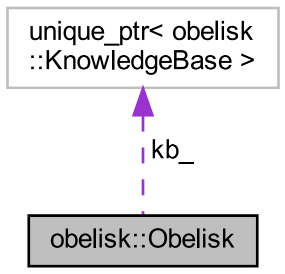
- src/ast/number_expression_ast.h
- src/ast/number_expression_ast.cpp

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.

```
{
    return obelisk::soVersion;
}
```

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.

```
{
    return obelisk::version;
}
```

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

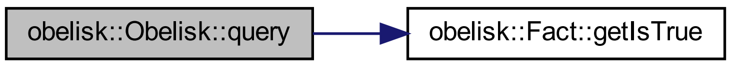
double Returns whether or not the Fact is true.

Definition at line 20 of file obelisk.cpp.

```
{
    obelisk::Fact fact = obelisk::Fact(obelisk::Entity(leftEntity),
        obelisk::Entity(rightEntity),
        obelisk::Verb(verb));

    kb_->queryFact(fact);

    return fact.getIsTrue();
}
```

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

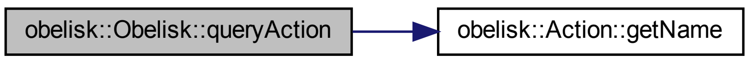
std::string Returns the suggested action.

Definition at line 33 of file obelisk.cpp.

```
{
    obelisk::Fact fact = obelisk::Fact(obelisk::Entity(leftEntity),
        obelisk::Entity(rightEntity),
        obelisk::Verb(verb));

    kb_->queryFact(fact);

    obelisk::Action action;
    kb_->querySuggestAction(fact, action);

    return action.getName();
}
```

References obelisk::Action::getName().

Here is the call graph for this function:



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

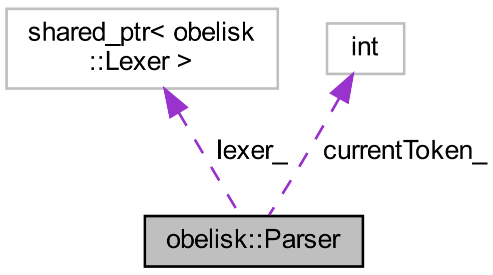
- src/lib/include/obelisk.h
- src/lib/obelisk.cpp

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

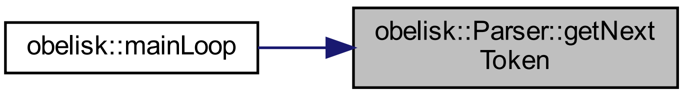
int Returns the next token.

Definition at line 22 of file parser.cpp.

```
{
    try
    {
        setCurrentToken(getLexer()->getToken());
    }
    catch (obelisk::LexerException& exception)
    {
        throw;
    }
    return getCurrentToken();
}
```

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

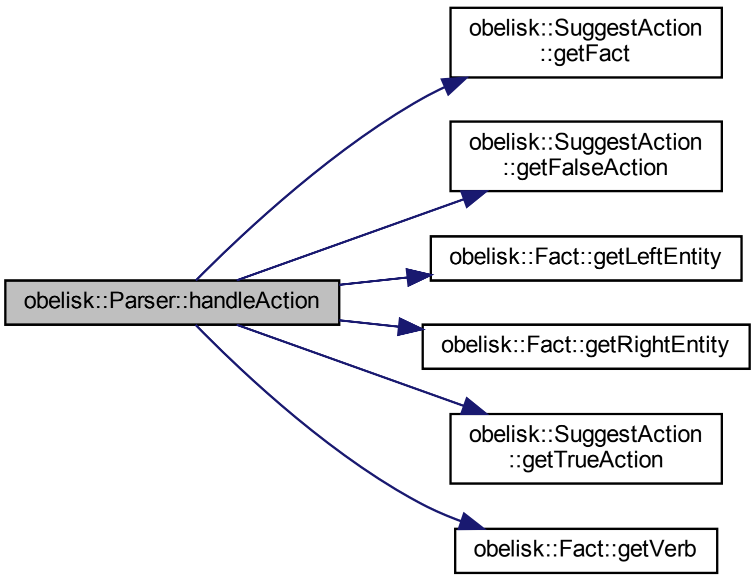
in	*kb*	The KnowledgeBase to insert the SuggestAction into.

Definition at line 774 of file parser.cpp.

```
{
    obelisk::SuggestAction suggestAction;

    try
    {
        parseAction(suggestAction);
        insertEntity(kb, suggestAction.getFact().getLeftEntity());
        insertEntity(kb, suggestAction.getFact().getRightEntity());
        insertVerb(kb, suggestAction.getFact().getVerb());
        insertFact(kb, suggestAction.getFact());
        insertAction(kb, suggestAction.getTrueAction());
        insertAction(kb, suggestAction.getFalseAction());
        insertSuggestAction(kb, suggestAction);
    }
    catch (obelisk::ParserException& exception)
    {
        throw;
    }
}
```

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:


obelisk::SuggestAction ::getFact
obelisk::SuggestAction ::getFalseAction
obelisk::Fact::getLeftEntity
obelisk::Parser::handleAction
obelisk::Fact::getRightEntity
obelisk::SuggestAction ::getTrueAction
obelisk::Fact::getVerb

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.

```
{
    std::vector<obelisk::Fact> facts;
    try
    {
        parseFact(facts);
    }
    catch (obelisk::ParserException& exception)
    {
        throw;
    }

    int verbId = 0;
    for (auto& fact : facts)
    {
        try
        {
            insertEntity(kb, fact.getLeftEntity());
            insertEntity(kb, fact.getRightEntity());
        }
        catch (obelisk::ParserException& exception)
        {
            throw;
        }

        if (verbId == 0)
        {
            try
            {
                insertVerb(kb, fact.getVerb());
            }
            catch (obelisk::ParserException& exception)
            {
                throw;
            }
            verbId = fact.getVerb().getId();
        }
        else
        {
            fact.getVerb().setId(verbId);
        }

        try
        {
            insertFact(kb, fact, true);
        }
        catch (obelisk::ParserException& exception)
        {
            throw;
        }

        kb->checkRule(fact);
    }
}
```

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

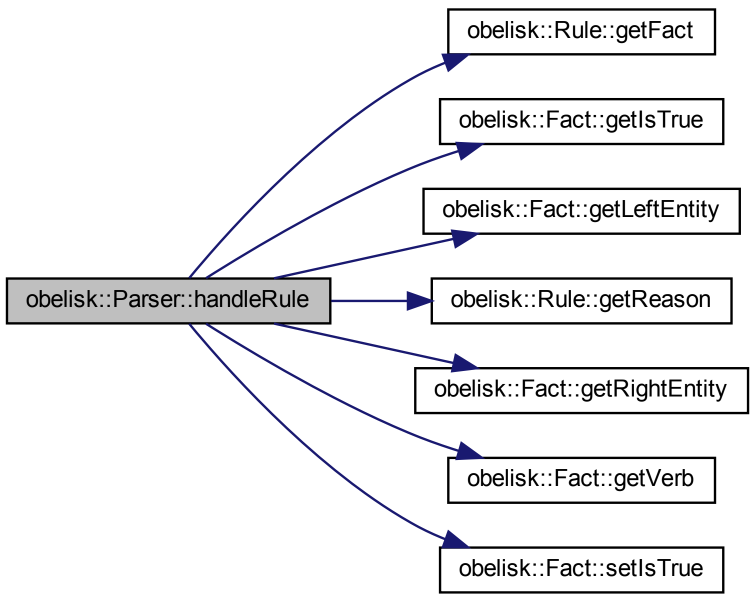
in	*kb*	The KnowledgeBase to insert the Rule into.

Definition at line 795 of file parser.cpp.

```
{
    obelisk::Rule rule;

    try
    {
        parseRule(rule);

        insertEntity(kb, rule.getReason().getLeftEntity());
        insertEntity(kb, rule.getReason().getRightEntity());
        insertVerb(kb, rule.getReason().getVerb());
        insertFact(kb, rule.getReason());

        // The rule is true, so the fact must be true to.
        if (rule.getReason().getIsTrue() > 0)
        {
            rule.getFact().setIsTrue(1.0);
        }

        insertEntity(kb, rule.getFact().getLeftEntity());
        insertEntity(kb, rule.getFact().getRightEntity());
        insertVerb(kb, rule.getFact().getVerb());
        insertFact(kb, rule.getFact());

        insertRule(kb, rule);
    }
    catch (obelisk::ParserException& exception)
    {
        throw;
    }
}
```

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

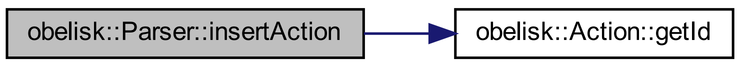
in	*kb*	The KnowledgeBase to use.
in,out	*action*	The Action to insert. It will contain the ID of the Action after inserting it.

Definition at line 920 of file parser.cpp.

```
{
    std::vector<obelisk::Action> actions {action};
    kb->addActions(actions);
    action = std::move(actions.front());

    // the id was not inserted, so check if it exists in the database
    if (action.getId() == 0)
    {
        kb->getAction(action);
        if (action.getId() == 0)
        {
            throw obelisk::ParserException(
                "action could not be inserted into the database");
        }
    }
}
```

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

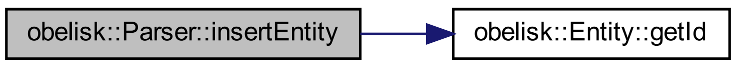
in	*kb*	The KnowledgeBase to use.
in,out	*entity*	The Entity to insert. It will contain the ID of the Entity after inserting it.

Definition at line 882 of file parser.cpp.

```
{
    std::vector<obelisk::Entity> entities {entity};
    kb->addEntities(entities);
    entity = std::move(entities.front());

    // the id was not inserted, so check if it exists in the database
    if (entity.getId() == 0)
    {
        kb->getEntity(entity);
        if (entity.getId() == 0)
        {
            throw obelisk::ParserException(
                "entity could not be inserted into the database");
        }
    }
}
```

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

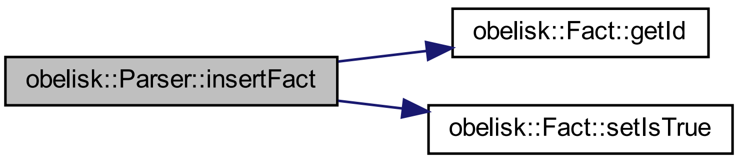
in	*kb*	The KnowledgeBase to use.
in,out	*fact*	The Fact to insert. It will contain the ID of the Fact after inserting it.
in	*updateIsTrue*	If true, it will update the value of is_true in the KnowledgeBase if the Fact already exists.

Definition at line 939 of file parser.cpp.

```
{
    std::vector<obelisk::Fact> facts {fact};
    kb->addFacts(facts);
    fact = std::move(facts.front());

    // the id was not inserted, so check if it exists in the database
    if (fact.getId() == 0)
    {
        kb->getFact(fact);
        if (fact.getId() == 0)
        {
            throw obelisk::ParserException(
                "fact could not be inserted into the database");
        }
        else
        {
            if (updateIsTrue)
            {
                fact.setIsTrue(1.0);
                kb->updateIsTrue(fact);
            }
        }
    }
}
```

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

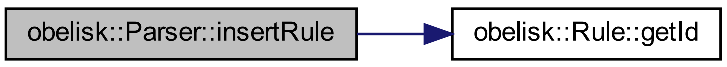
in	*kb*	The KnowledgeBase to use.
in,out	*rule*	The Rule to insert. It will contain the ID of the Rule after inserting it.

Definition at line 987 of file parser.cpp.

```
{
    std::vector<obelisk::Rule> rules {rule};
    kb->addRules(rules);
    rule = std::move(rules.front());

    // the id was not inserted, so check if it exists in the database
    if (rule.getId() == 0)
    {
        kb->getRule(rule);
        if (rule.getId() == 0)
        {
            throw obelisk::ParserException(
                "rule could not be inserted into the database");
        }
    }
}
```

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

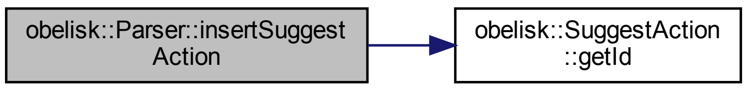
in	*kb*	The KnowledgeBase to use.
in,out	*suggestAction*	The SuggestAction to insert. It will contain the ID of the SuggestAction after inserting it.

Definition at line 967 of file parser.cpp.

```
{
    std::vector<obelisk::SuggestAction> suggestActions {suggestAction};
    kb->addSuggestActions(suggestActions);
    suggestAction = std::move(suggestActions.front());

    // the id was not inserted, so check if it exists in the database
    if (suggestAction.getId() == 0)
    {
        kb->getSuggestAction(suggestAction);
        if (suggestAction.getId() == 0)
        {
            throw obelisk::ParserException(
                "suggest_action could not be inserted into the database");
        }
    }
}
```

References obelisk::SuggestAction::getId().

Here is the call graph for this function:


obelisk::Parser::insertSuggest Action
obelisk::SuggestAction ::getId

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

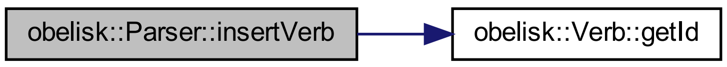
in	*kb*	The KnowledegeBase to use.
in,out	*verb*	The Verb to insert. It will contain the ID of the Verb after inserting it.

Definition at line 901 of file parser.cpp.

```
{
    std::vector<obelisk::Verb> verbs {verb};
    kb->addVerbs(verbs);
    verb = std::move(verbs.front());

    // the id was not inserted, so check if it exists in the database
    if (verb.getId() == 0)
    {
        kb->getVerb(verb);
        if (verb.getId() == 0)
        {
            throw obelisk::ParserException(
                "verb could not be inserted into the database");
        }
    }
}
```

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.

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

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.

```
{
    logError(str);
    return nullptr;
}
```

5.20.3.15 parseAction()

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

Parse a SuggestAction.

Parameters

out	*suggestAction*	The parsed SuggestAction.

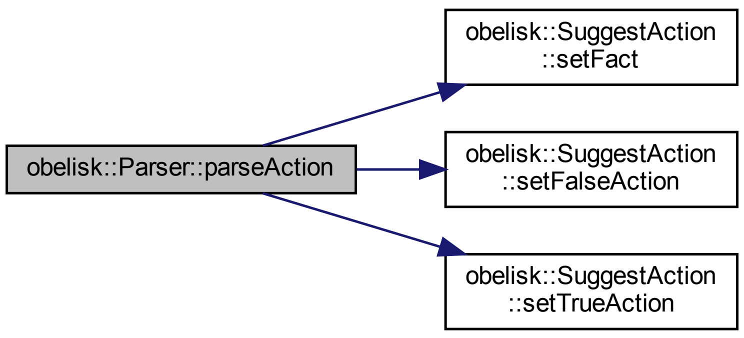
Definition at line 223 of file parser.cpp.

```
{
    std::stack<char> syntax;

    getNextToken();
    if (getCurrentToken() != '(')
    {
        throw obelisk::ParserException(
            "expected '(' but got '" + std::to_string(getCurrentToken()) + "'");
    }

    syntax.push('(');

    getNextToken();
    if (getLexer()->getIdentifier() != "if")
    {
        throw obelisk::ParserException(
            "expected 'if' but got '" + getLexer()->getIdentifier() + "'");
    }

    bool getEntity {true};
    std::string leftEntity {""};
    std::string rightEntity {""};
    std::string trueAction {""};
    std::string falseAction {""};
    std::string entityName {""};
    std::string verb {""};
    getNextToken();

    // get the entity side of statement
    while (true)
    {
        if (getEntity)
        {
            if (getCurrentToken() == '"')
            {
                if (syntax.top() != '"')
                {
                    // open a double quote
                    syntax.push('"');
                    getNextToken();
                }
                else if (syntax.top() == '"')
                {
                    // close a double quote
                    syntax.pop();
                    if (verb == "")
                    {
                        leftEntity = std::move(entityName);
                    }
                    else
                    {
                        rightEntity = std::move(entityName);
                    }
                    getEntity = false;
                    getNextToken();
                    continue;
                }
            }

            if (syntax.top() == '"')
```

```
                {
                    if (entityName != "")
                    {
                        entityName += " ";
                    }
                    entityName += getLexer()->getIdentifier();
                }
                getNextToken();
            }
            else
            {
                if (getCurrentToken() == ')')
                {
                    throw obelisk::ParserException("unexpected ')'");
                }

                if (getCurrentToken() == '"')
                {
                    throw obelisk::ParserException("unexpected '\"'");
                }

                if (getLexer()->getIdentifier() == "then")
                {
                    break;
                }
                else
                {
                    verb = getLexer()->getIdentifier();
                    for (const auto& letter : verb)
                    {
                        if (!isalpha(letter))
                        {
                            throw new obelisk::ParserException(
                                "non alphabetic symbol in verb");
                        }
                    }
                    getEntity = true;
                    continue;
                }
            }
        }

        // get the action side of statement
        bool getAction {true};
        while (true)
        {
            if (getAction)
            {
                if (getCurrentToken() == '"')
                {
                    if (syntax.top() != '"')
                    {
                        // open a double quote
                        syntax.push('"');
                        getNextToken();
                    }
                    else if (syntax.top() == '"')
                    {
                        // close a double quote
                        syntax.pop();
                        if (trueAction == "")
                        {
                            trueAction = std::move(entityName);
                        }
                        else
                        {
                            falseAction = std::move(entityName);
                        }
                        getAction = false;
                        getNextToken();
                        continue;
                    }
                }

                if (syntax.top() == '"')
                {
                    if (entityName != "")
                    {
                        entityName += " ";
                    }
                    entityName += getLexer()->getIdentifier();
                }
                getNextToken();
            }
            else
            {
                if (getCurrentToken() == ')')
                {
                    // closing parenthesis found, make sure we have everything
                    // needed
                    if (syntax.top() != '(')
                    {
                        throw obelisk::ParserException("unexpected ')'");
                    }
                    else
                    {
                        syntax.pop();
                    }

                    if (leftEntity == "")
                    {
                        throw obelisk::ParserException("missing left entity");
                    }

                    if (rightEntity == "")
                    {
                        throw obelisk::ParserException("missing left entity");
                    }

                    if (verb == "")
                    {
                        throw obelisk::ParserException("missing verb");
                    }

                    if (trueAction == "")
                    {
                        throw obelisk::ParserException("missing true action");
                    }

                    if (falseAction == "")
                    {
                        throw obelisk::ParserException("missing false action");
                    }

                    getNextToken();
                    if (getCurrentToken() != ';')
                    {
                        throw obelisk::ParserException("missing ';'");
                    }

                    break;
                }

                if (getCurrentToken() == '"')
                {
                    throw obelisk::ParserException("unexpected '\"'");
                    break;
                }

                if (getLexer()->getIdentifier() == "else")
                {
                    getNextToken();
                    getAction = true;
                    continue;
                }
                else
                {
                    getAction = true;
                    continue;
                }
            }
        }

        suggestAction.setFact(obelisk::Fact(obelisk::Entity(leftEntity),
            obelisk::Entity(rightEntity),
            obelisk::Verb(verb)));
        suggestAction.setTrueAction(obelisk::Action(trueAction));
        suggestAction.setFalseAction(obelisk::Action(falseAction));
    }
```

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

Here is the call graph for this function:


obelisk::Parser::parseAction
obelisk::SuggestAction ::setFact
obelisk::SuggestAction ::setFalseAction
obelisk::SuggestAction ::setTrueAction

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.

```
{
    getNextToken();
    auto prototype = parsePrototype();
    if (!prototype)
    {
        return nullptr;
    }

    if (auto expression = parseExpression())
    {
        return std::make_unique<FunctionAST>(std::move(prototype),
            std::move(expression));
    }

    return nullptr;
}
```

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.

```
{
    auto LHS = parsePrimary();
    if (!LHS)
    {
        return nullptr;
    }

    return LHS;
}
```

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.

```
{
    getNextToken();
    return parsePrototype();
}
```

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.

```
{
    std::stack<char> syntax;

    getNextToken();
    if (getCurrentToken() != '(')
    {
        throw obelisk::ParserException(
            "expected '(' but got '" + std::to_string(getCurrentToken()) + "'");
    }

    syntax.push('(');

    bool getEntity {true};
    std::vector<std::string> leftEntities;
    std::vector<std::string> rightEntities;
    std::string entityName {""};
    std::string verb {""};
    getNextToken();
    while (true)
    {
        if (getEntity)
        {
            if (getCurrentToken() == '"')
            {
                if (syntax.top() != '"')
                {
                    // open a double quote
                    syntax.push('"');
                    getNextToken();
                }
                else if (syntax.top() == '"')
                {
                    // close a double quote
                    syntax.pop();
                    if (verb == "")
                    {
                        leftEntities.push_back(entityName);
                    }
                    else
                    {
                        rightEntities.push_back(entityName);
                    }
                    entityName = "";
                    getEntity  = false;
                    getNextToken();
                    continue;
                }
            }

            if (syntax.top() == '"')
            {
                if (entityName != "")
                {
                    entityName += " ";
                }
                entityName += getLexer()->getIdentifier();
            }
            getNextToken();
        }
        else
        {
            if (getCurrentToken() == ')')
            {
                // closing parenthesis found, make sure we have everything
                // needed
                if (syntax.top() != '(')
                {
                    throw obelisk::ParserException("unexpected ')'");
                }
                else
                {
                    syntax.pop();
                }

                if (verb == "")
                {
                    throw obelisk::ParserException("verb is empty");
                }

                if (leftEntities.size() == 0)
                {
                    throw obelisk::ParserException(
                        "missing left side entities");
                }

                if (rightEntities.size() == 0)
                {
                    throw obelisk::ParserException(
                        "missing right side entities");
                }

                getNextToken();
                if (getCurrentToken() != ';')
                {
                    throw obelisk::ParserException("missing ';'");
                }

                break;
            }
            if (getCurrentToken() == '"')
            {
                throw obelisk::ParserException("unexpected '\"'");
            }

            if (getLexer()->getIdentifier() == "and")
            {
                getNextToken();
                getEntity = true;
                continue;
            }
            else
            {
                verb = getLexer()->getIdentifier();
                for (const auto& letter : verb)
                {
                    if (!isalpha(letter))
                    {
                        throw new obelisk::ParserException(
                            "non alphabetic symbol in verb");
                    }
                }
                getEntity = true;
                continue;
            }
        }
    }

    for (auto& leftEntity : leftEntities)
    {
        for (auto& rightEntity : rightEntities)
        {
            facts.push_back(obelisk::Fact(obelisk::Entity(leftEntity),
                obelisk::Entity(rightEntity),
                obelisk::Verb(verb),
                true));
        }
    }
}
```

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.

```
{
    std::string idName = getLexer()->getIdentifier();
    getNextToken();
    if (getCurrentToken() != '(')
    {
        return std::make_unique<obelisk::VariableExpressionAST>(idName);
    }

    getNextToken();
    std::vector<std::unique_ptr<obelisk::ExpressionAST» args;
    if (getCurrentToken() != ')')
    {
        while (true)
        {
            if (auto arg = parseExpression())
            {
                args.push_back(std::move(arg));
            }
            else
            {
                return nullptr;
            }

            if (getCurrentToken() == ')')
            {
                break;
            }

            if (getCurrentToken() != ',')
            {
                return logError("Expected ')' or ',' in argument list");
            }

            getNextToken();
        }
    }

    getNextToken();
    return std::make_unique<CallExpressionAST>(idName, std::move(args));
}
```

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.

```
{
    auto result = std::make_unique<obelisk::NumberExpressionAST>(
        getLexer()->getNumberValue());
    getNextToken();
    return result;
}
```

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.

```
{
    getNextToken();
    auto v = parseExpression();
    if (!v)
    {
        return nullptr;
    }

    if (getCurrentToken() != ')')
    {
        return logError("expected ')'");
    }
    getNextToken();
    return v;
}
```

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.

```
{
    switch (getCurrentToken())
    {
        case obelisk::Lexer::kTokenIdentifier :
            return parseIdentifierExpression();
        case obelisk::Lexer::kTokenNumber :
            return parseNumberExpression();
        case '(' :
            return parseParenthesisExpression();
        default :
            return logError("unknown token when expecting and expression");
    }
}
```

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.

```
{
    if (getCurrentToken() != obelisk::Lexer::kTokenIdentifier)
    {
        return logErrorPrototype("Expected function name in prototype");
    }

    std::string functionName = getLexer()->getIdentifier();
    getNextToken();

    if (getCurrentToken() != '(')
    {
        return logErrorPrototype("Expected '(' in prototype");
    }

    std::vector<std::string> argNames;
    while (getNextToken() == obelisk::Lexer::kTokenIdentifier)
    {
        argNames.push_back(getLexer()->getIdentifier());
    }

    if (getCurrentToken() != ')')
    {
        return logErrorPrototype("Expected ')' in prototype");
    }

    getNextToken();

    return std::make_unique<obelisk::PrototypeAST>(functionName,
        std::move(argNames));
}
```

References obelisk::Lexer::kTokenIdentifier.

5.20.3.25 parseRule()

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

Parse a Rule.

Parameters

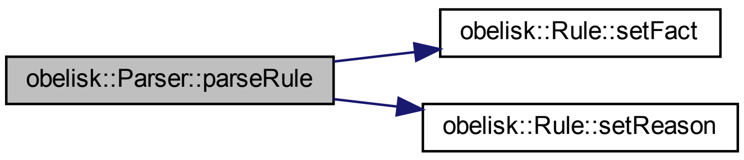
out	*rule*	The parsed Rule.

Definition at line 444 of file parser.cpp.

```
{
    std::stack<char> syntax;

    getNextToken();
    if (getCurrentToken() != '(')
    {
        throw obelisk::ParserException(
            "expected '(' but got '" + std::to_string(getCurrentToken()) + "'");
    }

    syntax.push('(');

    bool getEntity {true};
    bool getReason {false};
    std::string leftEntity {""};
    std::string rightEntity {""};
    std::string verb {""};
    std::string leftReasonEntity {""};
    std::string rightReasonEntity {""};
    std::string reasonVerb {""};
    std::string entityName {""};
    getNextToken();

    // get the entity side of statement
    while (true)
    {
        if (getEntity)
        {
            if (getCurrentToken() == '"')
            {
                if (syntax.top() != '"')
                {
                    // open a double quote
                    syntax.push('"');
                    getNextToken();
                }
                else if (syntax.top() == '"')
                {
                    // close a double quote
                    syntax.pop();
                    if (!getReason)
                    {
                        if (verb == "")
                        {
                            leftEntity = std::move(entityName);
                        }
                        else
                        {
                            rightEntity = std::move(entityName);
                        }
                    }
                    else
                    {
                        if (reasonVerb == "")
                        {
                            leftReasonEntity = std::move(entityName);
                        }
                        else
                        {
                            rightReasonEntity = std::move(entityName);
                        }
                    }
                    getEntity = false;
                    getNextToken();
                    continue;
                }
            }

            if (syntax.top() == '"')
            {
                if (entityName != "")
                {
                    entityName += " ";
                }
                entityName += getLexer()->getIdentifier();
            }
            getNextToken();
        }
        else
        {
            if (getCurrentToken() == ')')
            {
                // closing parenthesis found, make sure we have everything
                // needed
                if (syntax.top() != '(')
                {
                    throw obelisk::ParserException("unexpected ')'");
                }
                else
                {
                    syntax.pop();
                }

                if (leftEntity == "")
                {
                    throw obelisk::ParserException("missing left entity");
                }

                if (rightEntity == "")
                {
                    throw obelisk::ParserException("missing right entity");
                }

                if (verb == "")
                {
                    throw obelisk::ParserException("missing verb");
                }

                if (leftReasonEntity == "")
                {
                    throw obelisk::ParserException(
                        "missing left reason entity");
                }

                if (rightReasonEntity == "")
                {
                    throw obelisk::ParserException(
                        "missing right reason entity");
                }

                if (reasonVerb == "")
                {
                    throw obelisk::ParserException("missing reason verb");
                }

                getNextToken();
                if (getCurrentToken() != ';')
                {
                    throw obelisk::ParserException("missing ';'");
                }

                break;
            }

            if (getCurrentToken() == '"')
            {
                throw obelisk::ParserException("unexpected '\"'");
            }

            if (getLexer()->getIdentifier() == "if")
            {
                getReason = true;
                getEntity = true;
                getNextToken();
                continue;
            }
            else
            {
                if (!getReason)
                {
                    verb = getLexer()->getIdentifier();
                    for (const auto& letter : verb)
                    {
                        if (!isalpha(letter))
                        {
                            throw new obelisk::ParserException(
                                "non alphabetic symbol in verb");
                        }
                    }
                    getEntity = true;
                    continue;
                }
                else
                {
                    reasonVerb = getLexer()->getIdentifier();
                    for (const auto& letter : reasonVerb)
                    {
                        if (!isalpha(letter))
                        {
                            throw new obelisk::ParserException(
                                "non alphabetic symbol in verb");
                        }
                    }
                    getEntity = true;
                    continue;
                }
            }
        }
    }

    rule.setFact(obelisk::Fact(obelisk::Entity(leftEntity),
        obelisk::Entity(rightEntity),
        obelisk::Verb(verb)));
    rule.setReason(obelisk::Fact(obelisk::Entity(leftReasonEntity),
        obelisk::Entity(rightReasonEntity),
        obelisk::Verb(reasonVerb)));
}
```

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.

```
{
    if (auto expression = parseExpression())
    {
        // Make an anonymous prototype
        auto prototype = std::make_unique<obelisk::PrototypeAST>("__anon_expr",
            std::vector<std::string>());
        return std::make_unique<obelisk::FunctionAST>(std::move(prototype),
            std::move(expression));
    }
    return nullptr;
}
```

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:

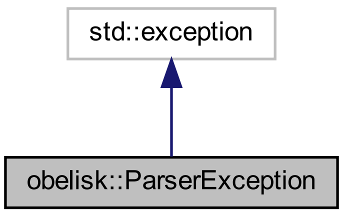
- src/parser.h
- src/parser.cpp

5.21 obelisk::ParserException Class Reference

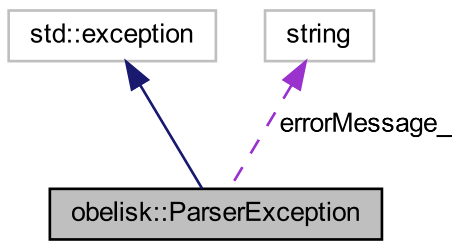
The exceptions thrown by the Parser.

```
#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:

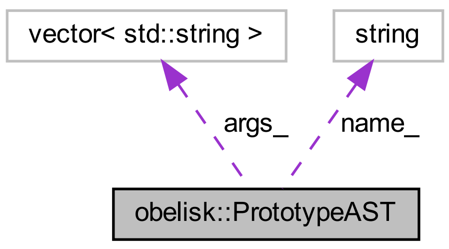
- src/parser.h

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.

```
                                           :
                 name_(name),
                 args_(std::move(args))
             {
             }
```

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.

```
{
    std::vector<llvm::Type *> doubles(args_.size(),
        llvm::Type::getDoubleTy(*TheContext));
    llvm::FunctionType *FT
        = llvm::FunctionType::get(llvm::Type::getDoubleTy(*TheContext),
            doubles,
            false);

    llvm::Function *F = llvm::Function::Create(FT,
        llvm::Function::ExternalLinkage,
        name_,
        obelisk::TheModule.get());

    unsigned idx = 0;
    for (auto &arg : F->args())
    {
        arg.setName(args_[idx++]);
    }

    return F;
}
```

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.

```
            {
                return name_;
            }
```

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:

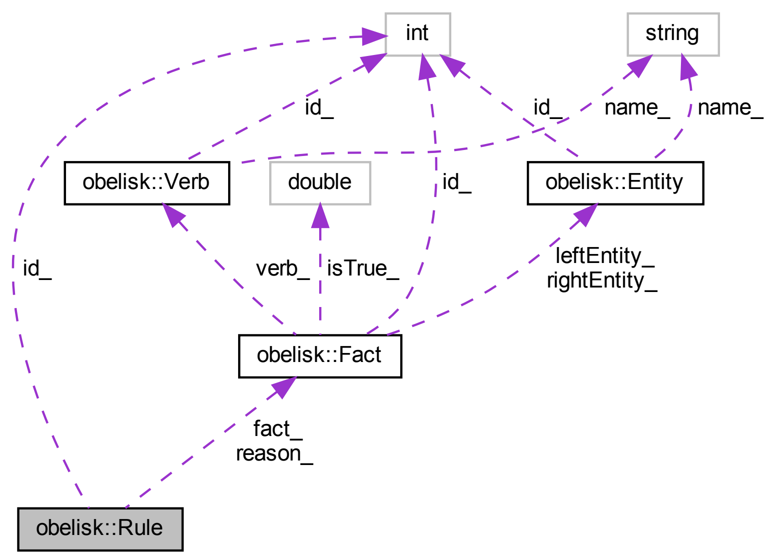
- src/ast/prototype_ast.h
- src/ast/prototype_ast.cpp

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.

```
                            :
                id_(id),
                fact_(),
                reason_()
            {
            }
```

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.

```
                                                        :
                id_(0),
                fact_(fact),
                reason_(reason)
            {
            }
```

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.

```
                                                                :
                id_(id),
                fact_(fact),
                reason_(reason)
            {
            }
```

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

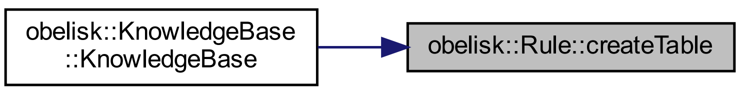
const char∗ Returns the query used to create the table.

Definition at line 4 of file rule.cpp.

```
{
    return R"(
        CREATE TABLE "rule" (
            "id"     INTEGER NOT NULL UNIQUE,
            "fact"   INTEGER NOT NULL,
            "reason" INTEGER NOT NULL CHECK("reason" != "fact"),
            PRIMARY KEY("id" AUTOINCREMENT),
            UNIQUE("fact", "reason"),
            FOREIGN KEY("fact") REFERENCES "fact"("id") ON DELETE RESTRICT,
            FOREIGN KEY("reason") REFERENCES "fact"("id") ON DELETE RESTRICT
        );
    )";
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::KnowledgeBase
obelisk::Rule::createTable

5.23.3.2 getFact()

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

Get the Fact object.

Returns

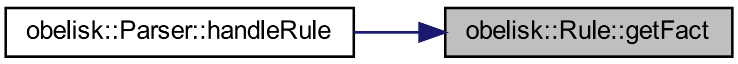
obelisk::Fact& The Fact.

Definition at line 268 of file rule.cpp.

```
{
    return fact_;
}
```

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

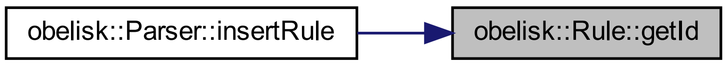
int& The ID.

Definition at line 258 of file rule.cpp.

```
{
    return id_;
}
```

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

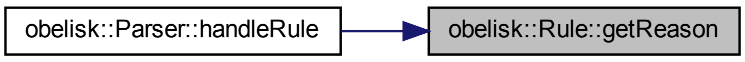
obelisk::Fact& The reason Fact.

Definition at line 278 of file rule.cpp.

```
{
    return reason_;
}
```

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.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "INSERT INTO rule (fact, reason) VALUES (?, ?)",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_int(ppStmt, 1, getFact().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_int(ppStmt, 2, getReason().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            setId((int) sqlite3_last_insert_rowid(dbConnection));
            sqlite3_set_last_insert_rowid(dbConnection, 0);
            break;
        case SQLITE_CONSTRAINT :
            throw obelisk::DatabaseConstraintException(
                sqlite3_errmsg(dbConnection));
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

5.23.3.6 selectById()

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

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

Parameters

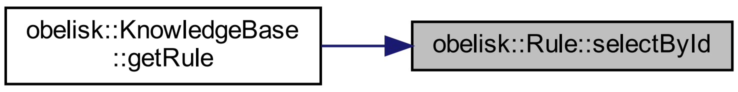
in	*dbConnection*	The database connection to use.

Definition at line 19 of file rule.cpp.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "SELECT id, fact, reason FROM rule WHERE (fact=? AND reason=?)",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_int(ppStmt, 1, getFact().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_int(ppStmt, 2, getReason().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            // no rows in the database
            break;
        case SQLITE_ROW :
            setId(sqlite3_column_int(ppStmt, 0));
            getFact().setId(sqlite3_column_int(ppStmt, 1));
            getReason().setId(sqlite3_column_int(ppStmt, 2));
            break;
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

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

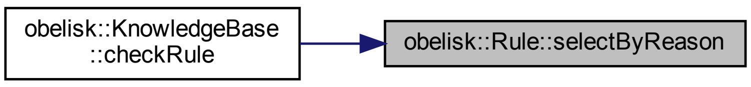
in	*dbConnection*	The database connection to use.
out	*rules*	The rules to fill in from the database.

Definition at line 190 of file rule.cpp.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "SELECT id, fact, reason FROM rule WHERE (reason=?)",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_int(ppStmt, 1, reasonId);
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    while ((result = sqlite3_step(ppStmt)) != SQLITE_DONE)
    {
        switch (result)
        {
            case SQLITE_ROW :
                rules.push_back(obelisk::Rule(sqlite3_column_int(ppStmt, 0),
                    obelisk::Fact(sqlite3_column_int(ppStmt, 1)),
                    obelisk::Fact(sqlite3_column_int(ppStmt, 2))));
                break;
            case SQLITE_BUSY :
                throw obelisk::DatabaseBusyException();
                break;
            case SQLITE_MISUSE :
                throw obelisk::DatabaseMisuseException();
                break;
            default :
                throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
                break;
        }
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::checkRule
obelisk::Rule::selectByReason

5.23.3.8 setFact()

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

Set the Fact object.

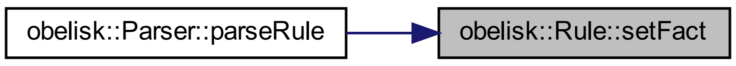
Parameters

in	*fact*	The Fact.

Definition at line 273 of file rule.cpp.
```
{
    fact_ = fact;
}
```

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.
```
{
    id_ = id;
}
```

5.23.3.10 setReason()

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

Set the reason Fact object.

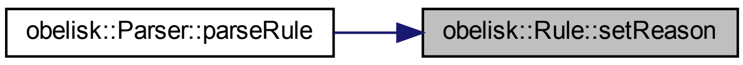
Parameters

in	*reason*	The reason Fact.

Definition at line 283 of file rule.cpp.
```
{
    reason_ = reason;
}
```

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

Here is the caller graph for this function:



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

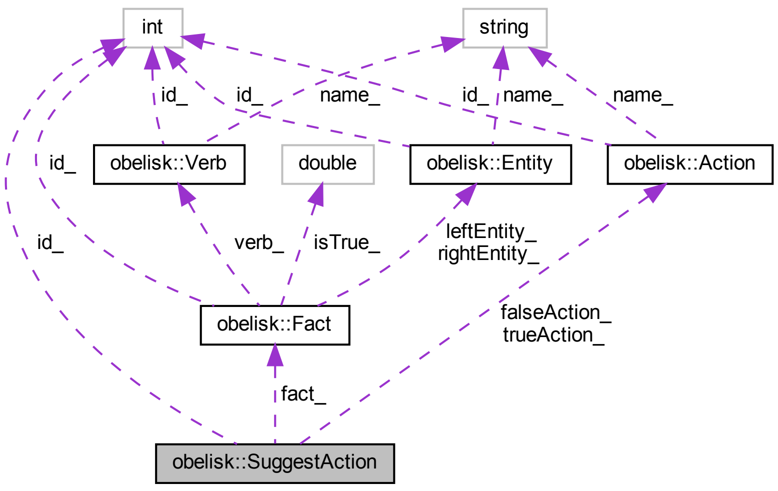
- src/lib/include/models/rule.h
- src/lib/models/rule.cpp

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.

```
                                  :
                 id_(id),
                 fact_(),
                 trueAction_(),
                 falseAction_()
             {
             }
```

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.

```
                                               :
                  id_(0),
                  fact_(fact),
                  trueAction_(trueAction),
                  falseAction_(falseAction)
              {
              }
```

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.

```
                                               :
                  id_(id),
                  fact_(fact),
                  trueAction_(trueAction),
                  falseAction_(falseAction)
              {
              }
```

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

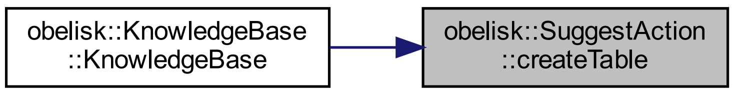
const char∗ Returns the query used to create the table.

Definition at line 4 of file suggest_action.cpp.

```
{
    return R"(
        CREATE TABLE "suggest_action" (
            "id"           INTEGER NOT NULL UNIQUE,
            "fact"         INTEGER NOT NULL,
            "true_action"  INTEGER NOT NULL,
            "false_action" INTEGER NOT NULL,
            PRIMARY KEY("id" AUTOINCREMENT),
            UNIQUE("fact", "true_action", "false_action"),
            FOREIGN KEY("fact") REFERENCES "fact"("id") ON DELETE RESTRICT,
            FOREIGN KEY("true_action") REFERENCES "action"("id") ON DELETE RESTRICT,
            FOREIGN KEY("false_action") REFERENCES "action"("id") ON DELETE RESTRICT
        );
    )";
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::KnowledgeBase
obelisk::SuggestAction ::createTable

5.24.3.2 getFact()

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

Get the Fact object.

Returns

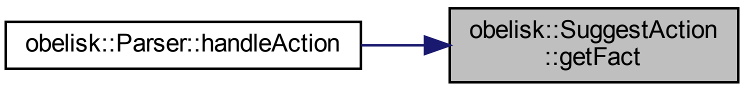
obelisk::Fact& Returns the Fact.

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:


obelisk::Parser::handleAction
obelisk::SuggestAction ::getFact

5.24.3.3 getFalseAction()

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

Get the false Action object.

Returns

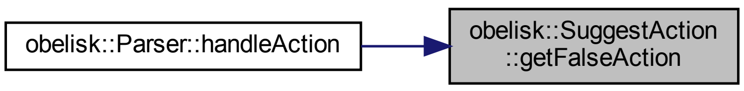
obelisk::Action& Returns the false Action.

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:


obelisk::Parser::handleAction
obelisk::SuggestAction ::getFalseAction

5.24.3.4 getId()

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

Get the ID of the SuggestAction.

Returns

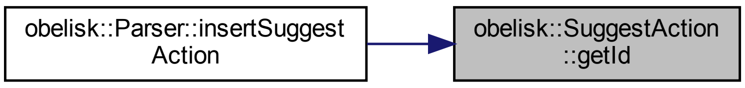
int& Returns the ID.

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

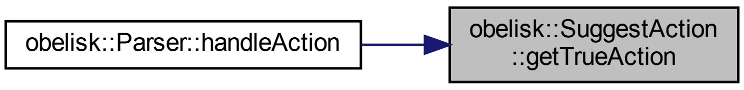
obelisk::Action& Returns the true Action.

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.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "INSERT INTO suggest_action (fact, true_action, false_action) VALUES (?, ?, ?)",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_int(ppStmt, 1, getFact().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_int(ppStmt, 2, getTrueAction().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_int(ppStmt, 3, getFalseAction().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            setId((int) sqlite3_last_insert_rowid(dbConnection));
            sqlite3_set_last_insert_rowid(dbConnection, 0);
            break;
        case SQLITE_CONSTRAINT :
            throw obelisk::DatabaseConstraintException(
                sqlite3_errmsg(dbConnection));
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

5.24.3.7 selectById()

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

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

Parameters

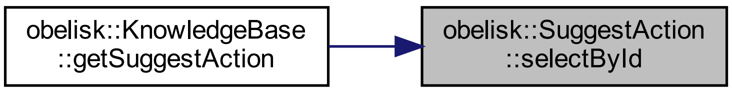
in	*dbConnection*	The database connection to use.

Definition at line 21 of file suggest_action.cpp.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "SELECT id, fact, true_action, false_action FROM suggest_action WHERE (fact=? AND true_action=?
AND false_action=?)",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_int(ppStmt, 1, getFact().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_int(ppStmt, 2, getTrueAction().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_bind_int(ppStmt, 3, getFalseAction().getId());
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            // no rows in the database
            break;
        case SQLITE_ROW :
            setId(sqlite3_column_int(ppStmt, 0));
            getFact().setId(sqlite3_column_int(ppStmt, 1));
            getTrueAction().setId(sqlite3_column_int(ppStmt, 2));
            getFalseAction().setId(sqlite3_column_int(ppStmt, 3));
            break;
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::getSuggestAction
obelisk::SuggestAction ::selectById

5.24.3.8 setFact()

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

Set the Fact object.

Parameters

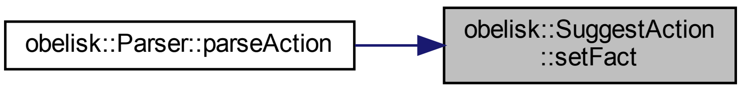
in	*fact*	The new Fact.

Definition at line 246 of file suggest_action.cpp.

```
{
    fact_ = fact;
}
```

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

Here is the caller graph for this function:


obelisk::Parser::parseAction
obelisk::SuggestAction ::setFact

5.24.3.9 setFalseAction()

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

Set the false Action object.

Parameters

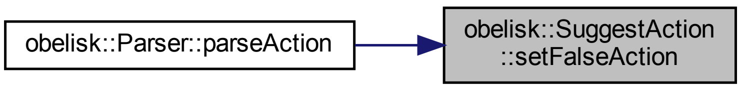
in	*falseAction*	The new false Action.

Definition at line 266 of file suggest_action.cpp.

```
{
    falseAction_ = falseAction;
}
```

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

Here is the caller graph for this function:


obelisk::Parser::parseAction
obelisk::SuggestAction ::setFalseAction

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.

```
{
    id_ = id;
}
```

5.24.3.11 setTrueAction()

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

Set the true Action object.

Parameters

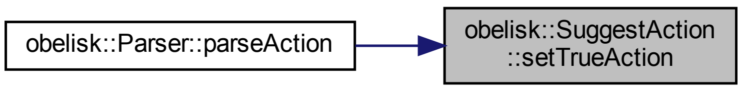
in	*trueAction*	The new true Action.

Definition at line 256 of file suggest_action.cpp.

```
{
    trueAction_ = trueAction;
00259 }
```

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

Here is the caller graph for this function:


obelisk::Parser::parseAction
obelisk::SuggestAction ::setTrueAction

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

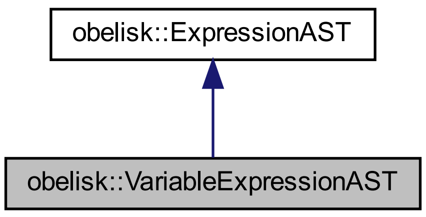
- src/lib/include/models/suggest_action.h
- src/lib/models/suggest_action.cpp

5.25 obelisk::VariableExpressionAST Class Reference

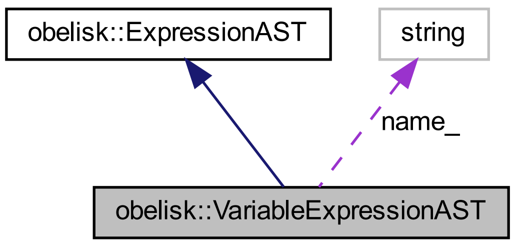
The variable expression AST node.

`#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.

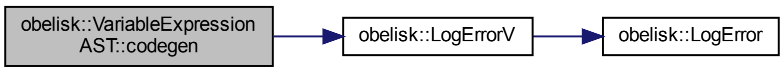
Definition at line 5 of file variable_expression_ast.cpp.

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

```
    }
    return V;
}
```

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:

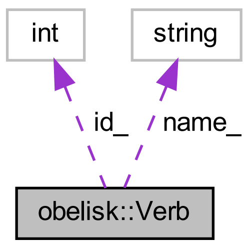
- src/ast/variable_expression_ast.h
- src/ast/variable_expression_ast.cpp

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.

```
                        :
            id_(id),
            name_("")
        {
        }
```

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.

```
                                  :
            id_(0),
            name_(name)
        {
        }
```

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.

```
                                          :
            id_(id),
            name_(name)
        {
        }
```

5.26.3 Member Function Documentation

5.26.3.1 createTable()

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

Create the Verb table in the KnowledgeBase.

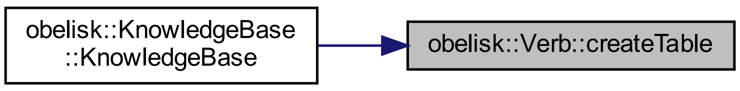
Returns

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

Definition at line 6 of file verb.cpp.
```
{
    return R"(
        CREATE TABLE "verb" (
            "id"   INTEGER NOT NULL UNIQUE,
            "name" TEXT NOT NULL CHECK(trim(name) != "") UNIQUE,
            PRIMARY KEY("id" AUTOINCREMENT)
        );
    )";
}
```

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

Here is the caller graph for this function:


obelisk::KnowledgeBase ::KnowledgeBase
obelisk::Verb::createTable

5.26.3.2 getId()

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

Get the ID of the Verb.

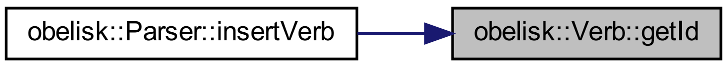
Returns

int& Returns the ID.

Definition at line 150 of file verb.cpp.
```
{
    return id_;
}
```

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.
```
{
    return name_;
}
```

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.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "INSERT INTO verb (name) VALUES (?)",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result
        = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_TRANSIENT);
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            setId((int) sqlite3_last_insert_rowid(dbConnection));
            sqlite3_set_last_insert_rowid(dbConnection, 0);
            break;
        case SQLITE_CONSTRAINT :
            throw obelisk::DatabaseConstraintException(
                sqlite3_errmsg(dbConnection));
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

5.26.3.5 selectByName()

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

Select a verb by name from the KnowledgeBase.

Parameters

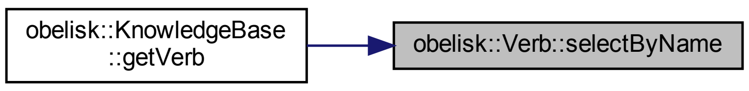
in	*dbConnection*	The database connection to use.

Definition at line 17 of file verb.cpp.

```
{
    if (dbConnection == nullptr)
    {
        throw obelisk::DatabaseException("database isn't open");
    }

    sqlite3_stmt* ppStmt = nullptr;

    auto result = sqlite3_prepare_v2(dbConnection,
        "SELECT id, name FROM verb WHERE name=?",
        -1,
        &ppStmt,
        nullptr);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }

    result = sqlite3_bind_text(ppStmt, 1, getName().c_str(), -1, SQLITE_STATIC);
    switch (result)
    {
        case SQLITE_OK :
            break;
        case SQLITE_TOOBIG :
            throw obelisk::DatabaseSizeException();
            break;
        case SQLITE_RANGE :
            throw obelisk::DatabaseRangeException();
            break;
        case SQLITE_NOMEM :
            throw obelisk::DatabaseMemoryException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_step(ppStmt);
    switch (result)
    {
        case SQLITE_DONE :
            // no rows in the database
            break;
        case SQLITE_ROW :
            setId(sqlite3_column_int(ppStmt, 0));
            setName((char*) sqlite3_column_text(ppStmt, 1));
            break;
        case SQLITE_BUSY :
            throw obelisk::DatabaseBusyException();
            break;
        case SQLITE_MISUSE :
            throw obelisk::DatabaseMisuseException();
            break;
        default :
            throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
            break;
    }

    result = sqlite3_finalize(ppStmt);
    if (result != SQLITE_OK)
    {
        throw obelisk::DatabaseException(sqlite3_errmsg(dbConnection));
    }
}
```

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.

```
{
    id_ = id;
}
```

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.

```
{
    name_ = name;
}
```

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

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

Index