

My Project

Generated by Doxygen 1.9.8

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 CompareHashable< HashableObject > Struct Template Reference	7
4.2 ExplanationSystem Class Reference	7
4.3 Hashable Class Reference	8
4.4 HashBinding< HashableObject > Struct Template Reference	8
4.5 Identifiable Class Reference	9
4.6 IrrationalNumber Struct Reference	9
4.6.1 Member Function Documentation	10
4.6.1.1 Rehash()	10
4.7 IrrationalPart Struct Reference	10
4.7.1 Member Function Documentation	12
4.7.1.1 Rehash()	12
4.8 Lexer Class Reference	12
4.8.1 Detailed Description	12
4.9 LexiconToken Struct Reference	12
4.10 MathExpression Struct Reference	13
4.10.1 Member Function Documentation	14
4.10.1.1 Rehash()	14
4.11 MathNode Struct Reference	14
4.12 Parser Class Reference	15
4.13 ProfileInformation Struct Reference	16
4.14 Profiler Class Reference	16
4.15 ProfilerSession Struct Reference	16
4.16 RandomEngine Class Reference	17
4.17 RationalNumber Struct Reference	17
4.18 RealNumber Struct Reference	18
4.19 Solver Class Reference	18
4.20 Timer Class Reference	18
5 File Documentation	19
5.1 ExplanationSystem.h	19
5.2 Hashable.h	19
5.3 Identifiable.h	20
5.4 Lexer.h	20
5.5 LexiconToken.h	21

5.6 Integer.h	21
5.7 Number.h	22
5.8 Operations.h	24
5.9 Rational.h	24
5.10 Real.h	24
5.11 MathNode.h	25
5.12 Parser.h	25
5.13 Solver.h	26
5.14 Conversions.h	26
5.15 Profiler.h	27
5.16 Random.h	27
5.17 Timer.h	28
5.18 Types.h	28
5.19 Utils.h	29
Index	31

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

CompareHashable< HashableObject >	7
ExplanationSystem	7
Hashable	8
IrrationalNumber	9
IrrationalPart	10
MathExpression	13
HashBinding< HashableObject >	8
Identifiable	9
LexiconToken	12
MathNode	14
Lexer	12
Parser	15
ProfileInformation	16
Profiler	16
ProfilerSession	16
RandomEngine	17
RationalNumber	17
RealNumber	18
Solver	18
Timer	18
Vector	
IrrationalPart	10

Chapter 2

Class Index

2.1 Class List

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

CompareHashable< HashableObject >	7
ExplanationSystem	7
Hashable	8
HashBinding< HashableObject >	8
Identifiable	9
IrrationalNumber	9
IrrationalPart	10
Lexer	12
LexiconToken	12
MathExpression	13
MathNode	14
Parser	15
ProfileInformation	16
Profiler	16
ProfilerSession	16
RandomEngine	17
RationalNumber	17
RealNumber	18
Solver	18
Timer	18

Chapter 3

File Index

3.1 File List

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

Build/Include/Core/ ExplanationSystem.h	19
Build/Include/Core/ Hashable.h	19
Build/Include/Core/ Identifiable.h	20
Build/Include/Core/ Lexer.h	20
Build/Include/Core/ LexiconToken.h	21
Build/Include/Core/ MathNode.h	25
Build/Include/Core/ Parser.h	25
Build/Include/Core/ Solver.h	26
Build/Include/Core/Math/ Integer.h	21
Build/Include/Core/Math/ Number.h	22
Build/Include/Core/Math/ Operations.h	24
Build/Include/Core/Math/ Rational.h	24
Build/Include/Core/Math/ Real.h	24
Build/Include/Core/Utility/ Conversions.h	26
Build/Include/Core/Utility/ Profiler.h	27
Build/Include/Core/Utility/ Random.h	27
Build/Include/Core/Utility/ Timer.h	28
Build/Include/Core/Utility/ Types.h	28
Build/Include/Core/Utility/ Utils.h	29

Chapter 4

Class Documentation

4.1 CompareHashable< HashableObject > Struct Template Reference

Public Member Functions

- bool **operator()** (const HashableObject &first, const HashableObject &second) const

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

- Build/Include/Core/Hashable.h

4.2 ExplanationSystem Class Reference

Public Member Functions

- void **Explain** (Ref< [MathNode](#) > tree)
- Vector< String > **GetSolution** () const
- void **Flush** ()

Static Public Member Functions

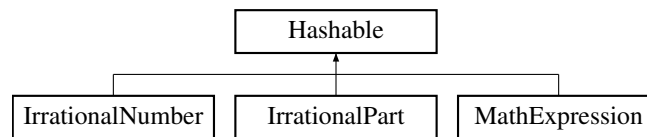
- static [ExplanationSystem](#) & **Get** ()

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

- Build/Include/Core/ExplanationSystem.h
- Build/Source/Core/ExplanationSystem.cpp

4.3 Hashable Class Reference

Inheritance diagram for Hashable:



Public Member Functions

- UInt32 **GetHash** () const
- virtual void **Rehash** ()=0
- template<> void **HashField** (UInt32 field)
- template<> void **HashField** (String field)
- template<> void **HashField** (Ref< [MathNode](#) > field)
- template<> void **HashField** ([RationalNumber](#) field)
- template<> void **HashField** ([IrrationalNumber](#) field)

Static Public Member Functions

- static bool **Compare** (const [Hashable](#) &first, const [Hashable](#) &second)

Protected Member Functions

- template<typename T >
void **HashField** (T field)
- void **AddCachedHash** (const UInt32 &cached)
- void **ResetHash** ()

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

- Build/Include/Core/Hashable.h
- Build/Source/Core/Hashable.cpp
- Build/Include/Core/Math/Number.h

4.4 HashBinding< HashableObject > Struct Template Reference

Public Member Functions

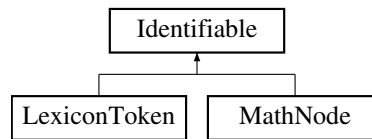
- UInt64 **operator()** (const HashableObject &first) const

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

- Build/Include/Core/Hashable.h

4.5 Identifiable Class Reference

Inheritance diagram for Identifiable:



Public Member Functions

- String **GetUUID** () const
- String **GetShortUUID** () const
- bool **operator==** ([Identifiable](#) other)
- bool **operator!=** ([Identifiable](#) other)

Protected Attributes

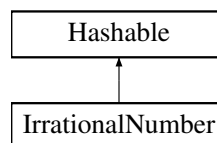
- String **mId**

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

- Build/Include/Core/Identifiable.h
- Build/Source/Core/Identifiable.cpp

4.6 IrrationalNumber Struct Reference

Inheritance diagram for IrrationalNumber:



Public Member Functions

- **IrrationalNumber** (Ref< [MathNode](#) > num=Mathematica::MakeRef< [MathNode](#) >(), Ref< [MathNode](#) > den=Mathematica::MakeRef< [MathNode](#) >())
- **IrrationalNumber** (const String &constantName)
- bool **operator==** (const [IrrationalNumber](#) &other)
- virtual void **Rehash** () override
- Float32 **RawNumerical** ()

Public Member Functions inherited from Hashable

- UInt32 **GetHash** () const
- template<> void **HashField** (UInt32 field)
- template<> void **HashField** (String field)
- template<> void **HashField** (Ref< [MathNode](#) > field)
- template<> void **HashField** ([RationalNumber](#) field)
- template<> void **HashField** ([IrrationalNumber](#) field)

Public Attributes

- Ref< [MathNode](#) > **numerator**
- Ref< [MathNode](#) > **denominator**

Additional Inherited Members

Static Public Member Functions inherited from Hashable

- static bool **Compare** (const [Hashable](#) &first, const [Hashable](#) &second)

Protected Member Functions inherited from Hashable

- template<typename T >
void **HashField** (T field)
- void **AddCachedHash** (const UInt32 &cached)
- void **ResetHash** ()

4.6.1 Member Function Documentation

4.6.1.1 Rehash()

```
void IrrationalNumber::Rehash ( ) [override], [virtual]
```

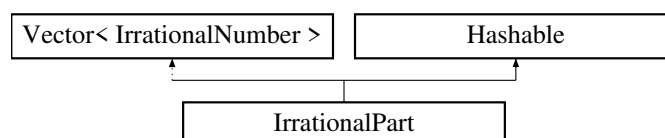
Implements [Hashable](#).

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

- Build/Include/Core/Math/Number.h
- Build/Source/Core/Math/Number.cpp

4.7 IrrationalPart Struct Reference

Inheritance diagram for IrrationalPart:



Public Types

- using **Super** = Vector< [IrrationalNumber](#) >
- using **IteratorType** = Super::iterator
- using **ConstIteratorType** = Super::const_iterator

Public Member Functions

- **IrrationalPart** (size_t count, const [IrrationalNumber](#) &val)
- template<typename ... Args>
decltype(auto) **EmplaceBack** (Args &&... args)
- IteratorType **begin** ()
- IteratorType **end** ()
- ConstIteratorType **begin** () const
- ConstIteratorType **end** () const
- [IrrationalNumber](#) **operator[]** (const UInt64 &where)
- const [IrrationalNumber](#) & **operator[]** (const UInt64 &where) const
- void **PushBack** (const [IrrationalNumber](#) &what)
- void **PushBack** ([IrrationalNumber](#) &&what)
- void **PopBack** ()
- UInt64 **Size** () const
- virtual void [Rehash](#) () override

Public Member Functions inherited from [Hashable](#)

- UInt32 **GetHash** () const
- template<> void **HashField** (UInt32 field)
- template<> void **HashField** (String field)
- template<> void **HashField** (Ref< [MathNode](#) > field)
- template<> void **HashField** ([RationalNumber](#) field)
- template<> void **HashField** ([IrrationalNumber](#) field)

Additional Inherited Members

Static Public Member Functions inherited from [Hashable](#)

- static bool **Compare** (const [Hashable](#) &first, const [Hashable](#) &second)

Protected Member Functions inherited from [Hashable](#)

- template<typename T >
void **HashField** (T field)
- void **AddCachedHash** (const UInt32 &cached)
- void **ResetHash** ()

4.7.1 Member Function Documentation

4.7.1.1 Rehash()

```
void IrrationalPart::Rehash ( ) [override], [virtual]
```

Implements [Hashable](#).

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

- Build/Include/Core/Math/Number.h
- Build/Source/Core/Math/Number.cpp

4.8 Lexer Class Reference

```
#include <Lexer.h>
```

Public Member Functions

- void **GenerateTokens** (String equation)
- Vector< [LexiconToken](#) > **GetTokens** () const
- Map< UInt32, HashMap< EPriority, Vector< UInt32 > > > **GetOperationIndex** () const
- HashMap< UInt32, Pair< UInt32, Vector< Pair< UInt32, UInt32 > > > > **GetScopeCounter** () const

4.8.1 Detailed Description

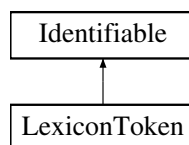
This class is used to generate an array of [LexiconToken](#) objects given a user input. It generates also metadata for each array, used during the parsing and solving steps.

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

- Build/Include/Core/Lexer.h
- Build/Source/Core/Lexer.cpp

4.9 LexiconToken Struct Reference

Inheritance diagram for LexiconToken:



Public Member Functions

- **LexiconToken** (String __data, ELexiconTokenType __type)
- String **GetTokenRichInformation** ()

Public Member Functions inherited from [Identifiable](#)

- String **GetUUID** () const
- String **GetShortUUID** () const
- bool **operator==** ([Identifiable](#) other)
- bool **operator!=** ([Identifiable](#) other)

Public Attributes

- String **data**
- ELexiconTokenType **type**

Additional Inherited Members**Protected Attributes inherited from [Identifiable](#)**

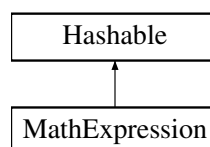
- String **mId**

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

- Build/Include/Core/LexiconToken.h
- Build/Source/Core/LexiconToken.cpp

4.10 MathExpression Struct Reference

Inheritance diagram for MathExpression:

**Public Member Functions**

- **MathExpression** (Vector< [RealNumber](#) > num={}, Vector< [RealNumber](#) > den={ {} })
- **MathExpression** ([RealNumber](#) real)
- virtual void **Rehash** () override
- void **CollapseTransformers** ()

Public Member Functions inherited from Hashable

- UInt32 **GetHash** () const
- template<> void **HashField** (UInt32 field)
- template<> void **HashField** (String field)
- template<> void **HashField** (Ref< [MathNode](#) > field)
- template<> void **HashField** ([RationalNumber](#) field)
- template<> void **HashField** ([IrrationalNumber](#) field)

Public Attributes

- Vector< [RealNumber](#) > **numerator**
- Vector< [RealNumber](#) > **denominator**
- Vector< Pair< ETransformer, [MathExpression](#) > > **transformers**

Additional Inherited Members

Static Public Member Functions inherited from Hashable

- static bool **Compare** (const [Hashable](#) &first, const [Hashable](#) &second)

Protected Member Functions inherited from Hashable

- template<typename T >
void **HashField** (T field)
- void **AddCachedHash** (const UInt32 &cached)
- void **ResetHash** ()

4.10.1 Member Function Documentation

4.10.1.1 Rehash()

```
void MathExpression::Rehash ( ) [override], [virtual]
```

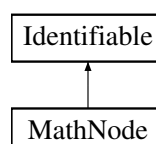
Implements [Hashable](#).

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

- Build/Include/Core/Math/Number.h
- Build/Source/Core/Math/Number.cpp

4.11 MathNode Struct Reference

Inheritance diagram for MathNode:



Public Types

- using **PointerType** = Ref< [MathNode](#) >
- using **ChildrenType** = Vector< PointerType >

Public Member Functions

- **MathNode** (PointerType __parent=nullptr, ChildrenType __children={}, Any __data={}, EMathNodeType __type=EMathNodeType::None, Int32 __scope=-1)

Public Member Functions inherited from [Identifiable](#)

- String **GetUUID** () const
- String **GetShortUUID** () const
- bool **operator==** ([Identifiable](#) other)
- bool **operator!=** ([Identifiable](#) other)

Public Attributes

- Int32 **scope**
- PointerType **parent**
- ChildrenType **children**
- Any **data**
- EMathNodeType **type**

Additional Inherited Members

Protected Attributes inherited from [Identifiable](#)

- String **mId**

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

- Build/Include/Core/MathNode.h
- Build/Source/Core/MathNode.cpp

4.12 Parser Class Reference

Public Member Functions

- void **InitParser** (const Vector< [LexiconToken](#) > &tokens, const Map< UInt32, HashMap< EPriority, Vector< UInt32 > > > &opIndexes, const HashMap< UInt32, Pair< UInt32, Vector< Pair< UInt32, UInt32 > > > &scopeCounter)
- Map< UInt32, Vector< Ref< [MathNode](#) > > > **GetExecutionFlow** () const
- Ref< [MathNode](#) > **GenerateTree** ()

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

- Build/Include/Core/Parser.h
- Build/Source/Core/Parser.cpp

4.13 ProfileInformation Struct Reference

Public Attributes

- String **name**
- Int32 **start**
- Int32 **end**
- Int32 **duration**
- Int32 **processId**
- Int32 **threadId**

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

- Build/Include/Core/Utility/Profiler.h

4.14 Profiler Class Reference

Public Member Functions

- void **BeginProfile** (const String &name, String outFilePath="")
- void **WriteProfile** (const [ProfileInformation](#) &information)
- void **EndProfile** ()

Static Public Member Functions

- static [Profiler](#) & **Get** ()

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

- Build/Include/Core/Utility/Profiler.h
- Build/Source/Core/Utility/Profiler.cpp

4.15 ProfilerSession Struct Reference

Public Attributes

- String **name**

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

- Build/Include/Core/Utility/Profiler.h

4.16 RandomEngine Class Reference

Static Public Member Functions

- static void **Init** ()
- static Int32 **Int** (Int32 minRange=0, Int32 maxRange=0x7fffffff)
- static UInt32 **UnsignedInt** (UInt32 minRange=0, UInt32 maxRange=0xffffffff)
- static Float32 **Float** (Float32 minRange=0.0, Float32 maxRange=1.0)
- static Float64 **Double** (Float64 minRange=0.0, Float64 maxRange=1.0)

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

- Build/Include/Core/Utility/Random.h
- Build/Source/Core/Utility/Random.cpp

4.17 RationalNumber Struct Reference

Public Member Functions

- **RationalNumber** (Int32 num=1, Int32 den=1)
- **RationalNumber** (const String &strNumber)
- **RationalNumber operator+** ([RationalNumber](#) other)
- **RationalNumber operator-** ([RationalNumber](#) other)
- **RationalNumber operator*** ([RationalNumber](#) other)
- **RationalNumber operator/** ([RationalNumber](#) other)
- **RationalNumber operator-** ()
- const [RationalNumber](#) & **operator+** ([RationalNumber](#) other) const
- const [RationalNumber](#) & **operator-** ([RationalNumber](#) other) const
- const [RationalNumber](#) & **operator*** ([RationalNumber](#) other) const
- const [RationalNumber](#) & **operator/** ([RationalNumber](#) other) const
- void **operator+=** ([RationalNumber](#) other)
- void **operator-=** ([RationalNumber](#) other)
- void **operator*=** ([RationalNumber](#) other)
- void **operator/=** ([RationalNumber](#) other)
- bool **operator==** ([RationalNumber](#) other)
- bool **operator!=** ([RationalNumber](#) other)
- bool **operator>=** ([RationalNumber](#) other)
- bool **operator<=** ([RationalNumber](#) other)
- bool **operator>** ([RationalNumber](#) other)
- bool **operator<** ([RationalNumber](#) other)
- Float32 **RawNumerical** ()
- [RationalNumber](#) **LowestTerms** (Int32 num, Int32 den)
- void **LowestTerms** ()

Public Attributes

- Int32 **numerator**
- Int32 **denominator**
- EnumerType **type**

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

- Build/Include/Core/Math/Number.h
- Build/Source/Core/Math/Number.cpp

4.18 RealNumber Struct Reference

Public Member Functions

- **RealNumber** ([RationalNumber](#) rational={}, [IrrationalPart](#) irrational={})
- **RealNumber** (const String &strNumber)
- [RealNumber](#) **operator-** ()
- Float32 **RawNumerical** ()

Public Attributes

- [RationalNumber](#) **rational**
- [IrrationalPart](#) **irrational**
- ESubset **type**

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

- Build/Include/Core/Math/Number.h
- Build/Source/Core/Math/Number.cpp

4.19 Solver Class Reference

Public Member Functions

- void **InitSolver** (const Ref< [MathNode](#) > &tree)
- void **InitSolver** (const Ref< [MathNode](#) > &tree, const Map< UInt32, Vector< Ref< [MathNode](#) > > > &executionFlow)
- [RationalNumber](#) **SolveTree** ()

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

- Build/Include/Core/Solver.h
- Build/Source/Core/Solver.cpp

4.20 Timer Class Reference

Public Member Functions

- **Timer** (const String &name)
- void **Start** ()
- void **Stop** ()

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

- Build/Include/Core/Utility/Timer.h
- Build/Source/Core/Utility/Timer.cpp

Chapter 5

File Documentation

5.1 ExplanationSystem.h

```
00001 #pragma once
00002
00003 #include "Core/Utility/Types.h"
00004
00005 #include "Core/MathNode.h"
00006
00007 class ExplanationSystem
00008 {
00009 public:
00010     void Explain(Ref<MathNode> tree);
00011
00012     Vector<String> GetSolution() const { return mSolution; }
00013
00014     static ExplanationSystem& Get();
00015     void Flush();
00016
00017 private:
00018     void RecursiveExplain(Ref<MathNode> node);
00019
00020     String mCurrentStep;
00021     Vector<String> mSolution;
00022 };
```

5.2 Hashable.h

```
00001 #pragma once
00002
00003 #include "Core/MathNode.h"
00004
00005 #include "Core/Utility/Utils.h"
00006 #include "Core/Utility/Types.h"
00007
00008 class Hashable
00009 {
00010 public:
00011     Hashable();
00012     UInt32 GetHash() const { return mHash; }
00013
00014     virtual void Rehash() = 0;
00015
00016     static bool Compare(const Hashable& first, const Hashable& second);
00017
00018 protected:
00019     template<typename T>
00020     void HashField(T field);
00021
00022     void AddCachedHash(const UInt32& cached) { mHash *= cached; }
00023     void ResetHash() { mHash = 1; }
00024
00025 private:
00026     void Hash(UInt32 seed);
00027
00028     UInt32 mHash;
00029 };
```

```

00030
00031 template<typename HashableObject>
00032 struct HashBinding
00033 {
00034     UInt64 operator()(const HashableObject& first) const
00035     {
00036         return Mathematica::Cast<UInt64>(first.GetHash());
00037     }
00038 };
00039
00040 template<typename HashableObject>
00041 struct CompareHashable
00042 {
00043     bool operator()(const HashableObject& first, const HashableObject& second) const
00044     {
00045         return Hashable::Compare(first, second);
00046     }
00047 };
00048
00049 template<typename T>
00050 inline void Hashable::HashField(T field)
00051 {
00052     MTH_ASSERT(false, "HashError: To use Hashable with this type, you have to implement your own hash
function!");
00053 }
00054
00055 template<>
00056 inline void Hashable::HashField(UInt32 field)
00057 {
00058     Hash(field);
00059 }
00060
00061 template<>
00062 inline void Hashable::HashField(String field)
00063 {
00064     UInt32 hashNumber = 0;
00065
00066     const UInt16 magicHigh = 0xaf13;
00067     const UInt16 magicLow = 0x4b71;
00068
00069     for (auto c : field) hashNumber += c * magicHigh - magicLow / c;
00070
00071     Hash(hashNumber);
00072 }
00073
00074 template<>
00075 inline void Hashable::HashField(Ref<MathNode> field)
00076 {
00077     Hash(*MTH_UINT_ADDRESS_OF(field->data));
00078
00079     for (Ref<MathNode> child : field->children) HashField(child);
00080 }

```

5.3 Identifiable.h

```

00001 #pragma once
00002
00003 #include "Core/Utility/Types.h"
00004
00005 class Identifiable
00006 {
00007 public:
00008     Identifiable();
00009     String GetUUID() const { return mId; }
00010     String GetShortUUID() const { return String(mId.begin(), mId.begin() + 7); }
00011
00012     bool operator==(Identifiable other);
00013     bool operator!=(Identifiable other);
00014
00015 protected:
00016     String mId;
00017
00018 private:
00019     String GenerateUUID();
00020 };

```

5.4 Lexer.h

```

00001 #pragma once

```



```

00002
00003 #include "Core/LexiconToken.h"
00004
00005 #include "Core/Utility/Types.h"
00006
00007 enum class EPriority : Int32
00008 {
00009     Low,
00010     Medium,
00011     High,
00012     Macro
00013 };
00014
00023 class Lexer
00024 {
00025 public:
00026     Lexer();
00027
00028     void GenerateTokens(String equation);
00029
00030     Vector<LexiconToken> GetTokens() const { return mTokens; }
00031     Map<UInt32, HashMap<EPriority, Vector<UInt32>>> GetOperationIndex() const { return
mOperationIndexes; }
00032     HashMap<UInt32, Pair<UInt32, Vector<Pair<UInt32, UInt32>>> GetScopeCounter() const { return
mScopeCounter; }
00033
00034 private:
00035     String StringifyNumberToken(String string, bool invertSign);
00036     String PreliminaryProcess(String equation);
00037
00038     Vector<LexiconToken> mTokens;
00039
00040     // Token Metadata
00041     Map<UInt32, HashMap<EPriority, Vector<UInt32>>> mOperationIndexes;
00042     HashMap<UInt32, Pair<UInt32, Vector<Pair<UInt32, UInt32>>> mScopeCounter;
00043     Map<UInt32, Pair<UInt32, String> mFunctionMacros;
00044 };
00045
00046 // NOTE : Making Lexer and Parser friends is potentially useful. Maybe in the future I'll change that.

```

5.5 LexiconToken.h

```

00001 #pragma once
00002
00003 #include "Core/Identifiable.h"
00004
00005 #include "Core/Utility/Utils.h"
00006
00007 enum class ELexiconTokenType : Int32
00008 {
00009     Number,
00010     BinaryFunction,
00011     WrapperStart,
00012     WrapperEnd,
00013     Macro,
00014
00015     Unknown
00016 };
00017
00018 struct LexiconToken : public Identifiable
00019 {
00020     String data;
00021     ELexiconTokenType type;
00022
00023     LexiconToken(String __data, ELexiconTokenType __type);
00024     String GetTokenRichInformation();
00025 };

```

5.6 Integer.h

```

00001 #pragma once
00002
00003 #include "Core/Utility/Utils.h"
00004
00005 namespace Mathematica
00006 {
00007     namespace Integer
00008     {
00009         // Integer functions

```

```

00010         Int32 LeastCommonMultiple(Int32 a, Int32 b);
00011         Int32 GreatestCommonDivisor(Int32 a, Int32 b);
00012
00013         // Prime numbers
00014         Map<Int32, Int32> Factorize(Int32 n);
00015         Map<Int32, Int32> Factorize(RationalNumber n);
00016
00017         bool IsPrime(Int32 n);
00018         bool IsPrime(RationalNumber n);
00019
00020         Int32 Prime(Int32 n);
00021         Int32 Prime(RationalNumber n);
00022
00023         // Sieve of Erathosthenes
00024         Vector<Int32> SoE (Int32 max);
00025     }
00026 }

```

5.7 Number.h

```

00001 #pragma once
00002
00003 #include "Core/Identifiable.h"
00004 #include "Core/Hashable.h"
00005 #include "Core/MathNode.h"
00006
00007 #include "Core/Utility/Utils.h"
00008
00009 // NOTE : This definition might vary.
00010 enum class ENumberType : Int32
00011 {
00012     Integer,
00013     Rational,
00014 };
00015
00016 enum class ESubset : Int32
00017 {
00018     Rational,
00019     Irrational,
00020     Real,
00021 };
00022
00023 enum class ETransformer : UInt32
00024 {
00025     Exponentiate
00026 };
00027
00028 struct RationalNumber
00029 {
00030     Int32 numerator;
00031     Int32 denominator;
00032     ENumberType type;
00033
00034     RationalNumber(Int32 num = 1, Int32 den = 1);
00035     RationalNumber(const String& strNumber);
00036
00037     RationalNumber operator+(RationalNumber other);
00038     RationalNumber operator-(RationalNumber other);
00039     RationalNumber operator*(RationalNumber other);
00040     RationalNumber operator/(RationalNumber other);
00041
00042     RationalNumber operator-();
00043
00044     const RationalNumber& operator+(RationalNumber other) const;
00045     const RationalNumber& operator-(RationalNumber other) const;
00046     const RationalNumber& operator*(RationalNumber other) const;
00047     const RationalNumber& operator/(RationalNumber other) const;
00048
00049     void operator+=(RationalNumber other);
00050     void operator-=(RationalNumber other);
00051     void operator*=(RationalNumber other);
00052     void operator/=(RationalNumber other);
00053
00054     bool operator==(RationalNumber other);
00055     bool operator!=(RationalNumber other);
00056     bool operator>=(RationalNumber other);
00057     bool operator<=(RationalNumber other);
00058     bool operator> (RationalNumber other);
00059     bool operator< (RationalNumber other);
00060
00061     Float32 RawNumerical();
00062
00063     RationalNumber LowestTerms(Int32 num, Int32 den);

```

```

00064     void LowestTerms();
00065 };
00066
00067 struct IrrationalNumber : public Hashable
00068 {
00069     Ref<MathNode> numerator;
00070     Ref<MathNode> denominator;
00071
00072     IrrationalNumber(Ref<MathNode> num = Mathematica::MakeRef<MathNode>(), Ref<MathNode> den =
Mathematica::MakeRef<MathNode>());
00073     IrrationalNumber(const String& constantName);
00074
00075     bool operator==(const IrrationalNumber& other);
00076
00077     virtual void Rehash() override;
00078
00079     Float32 RawNumerical();
00080 };
00081
00082 // REFACTOR : Create HashableVector Data Structure
00083 struct IrrationalPart : private Vector<IrrationalNumber>, public Hashable
00084 {
00085     using Super = Vector<IrrationalNumber>;
00086     using IteratorType = Super::iterator;
00087     using ConstIteratorType = Super::const_iterator;
00088
00089     IrrationalPart() {}
00090     IrrationalPart(size_t count, const IrrationalNumber& val) : Vector<IrrationalNumber>(count, val) {
Rehash(); }
00091
00092     template<typename ...Args>
00093     inline decltype(auto) EmplaceBack(Args&&... args)
00094     {
00095         Super::emplace_back(std::forward<Args>(args)...);
00096         Rehash();
00097     }
00098
00099     IteratorType begin() { return Super::begin(); }
00100     IteratorType end() { return Super::end(); }
00101
00102     ConstIteratorType begin() const { return Super::begin(); }
00103     ConstIteratorType end() const { return Super::end(); }
00104
00105     IrrationalNumber operator[](const UInt64& where) { return Super::operator[](where); }
00106     const IrrationalNumber& operator[](const UInt64& where) const { return Super::operator[](where); }
00107
00108     void PushBack(const IrrationalNumber& what);
00109     void PushBack(IrrationalNumber&& what);
00110
00111     void PopBack();
00112     UInt64 Size() const { return Super::size(); }
00113
00114     virtual void Rehash() override;
00115 };
00116
00117 struct RealNumber
00118 {
00119     RationalNumber rational;
00120     IrrationalPart irrational;
00121     ESubset type;
00122
00123     RealNumber(RationalNumber rational = {}, IrrationalPart irrational = {});
00124     RealNumber(const String& strNumber);
00125
00126     RealNumber operator-();
00127
00128     Float32 RawNumerical();
00129 };
00130
00131 struct MathExpression : public Hashable
00132 {
00133     Vector<RealNumber> numerator;
00134     Vector<RealNumber> denominator;
00135
00136     Vector<Pair<ETransformer, MathExpression>> transformers;
00137
00138     MathExpression(Vector<RealNumber> num = {}, Vector<RealNumber> den = { {} });
00139     MathExpression(RealNumber real);
00140
00141     virtual void Rehash() override;
00142
00143     void CollapseTransformers();
00144 };
00145
00146 // TODO : Move these functions somewhere else.
00147 namespace Mathematica
00148 {

```

```

00149     RationalNumber Absolute(RationalNumber number);
00150     Int32 Sign(RationalNumber number);
00151 }
00152
00153 template<>
00154 inline void Hashable::HashField(RationalNumber field)
00155 {
00156     Hash(field.numerator);
00157     Hash(field.denominator);
00158 }
00159
00160 template<>
00161 inline void Hashable::HashField(IrrationalNumber field)
00162 {
00163     HashField(field.numerator);
00164     HashField(field.denominator);
00165 }

```

5.8 Operations.h

```

00001 #pragma once
00002
00003 #include "Core/Utility/Types.h"
00004
00005 namespace Mathematica
00006 {
00007     namespace Operation
00008     {
00009         RationalNumber Add      (const RationalNumber& a, const RationalNumber& b);
00010         RationalNumber Subtract (const RationalNumber& a, const RationalNumber& b);
00011         RationalNumber Multiply (const RationalNumber& a, const RationalNumber& b);
00012         RationalNumber Divide   (const RationalNumber& a, const RationalNumber& b);
00013         RealNumber Exponentiate (const RationalNumber& a, RationalNumber b);
00014
00015         RationalNumber Raise     (const RationalNumber& a, const Int32& b);
00016
00017         RationalNumber Mod       (const RationalNumber& a, const RationalNumber& b);
00018     }
00019 }

```

5.9 Rational.h

```

00001 #pragma once
00002
00003 #include "Core/Utility/Types.h"
00004
00005 namespace Mathematica
00006 {
00007     namespace Rational
00008     {
00009         RationalNumber Average(Vector<RationalNumber> numbers);
00010         RationalNumber Average(RationalNumber first, RationalNumber second);
00011
00012         RationalNumber Between(RationalNumber first, RationalNumber second);
00013
00014         // TODO : Move this function somewhere else.
00015         Int32 Sign(Float32 number);
00016
00017         RationalNumber Farey(Float32 number);
00018
00019         // Rationality
00020         bool IsRootRational(RationalNumber arg, Int32 index);
00021     }
00022 }

```

5.10 Real.h

```

00001 #pragma once
00002
00003 #include "Core/Math/Number.h"
00004
00005 namespace Mathematica
00006 {
00007     namespace Real
00008     {

```

```

00009     namespace Operation
00010     {
00011         MathExpression Add(const MathExpression& first, const MathExpression& second);
00012         MathExpression Subtract(const MathExpression& first, const MathExpression& second);
00013         MathExpression Multiply(const MathExpression& first, const MathExpression& second);
00014         MathExpression Divide(const MathExpression& first, const MathExpression& second);
00015     }
00016
00017     Vector<RealNumber> ExecuteSimplify(const Vector<RealNumber>& expression);
00018     Vector<RealNumber> ExecuteMultiply(const Vector<RealNumber>& first, const Vector<RealNumber>&
second);
00019 }
00020
00021     namespace Irrational
00022     {
00023         RealNumber ProcessIrrationalSubTree(const Ref<MathNode>& subTree, const UInt32& counter);
00024     }
00025 }

```

5.11 MathNode.h

```

00001 #pragma once
00002
00003 #include "Core/Identifiable.h"
00004
00005 #include "Core/Utility/Types.h"
00006
00007 enum class EMathNodeType : Int32
00008 {
00009     Number,
00010     BinaryFunction,
00011     NamedFunction,
00012
00013     WrapStart,
00014     WrapEnd,
00015
00016     Wrapper,
00017
00018     Macro,
00019
00020     None
00021 };
00022
00023 // This struct behaves like a node of a tree.
00024 struct MathNode : public Identifiable
00025 {
00026     using PointerType = Ref<MathNode>;
00027     using ChildrenType = Vector<PointerType>;
00028
00029     Int32 scope;
00030
00031     PointerType parent;
00032     ChildrenType children;
00033
00034     Any data;
00035     EMathNodeType type;
00036
00037     MathNode(
00038         PointerType __parent = nullptr,
00039         ChildrenType __children = {},
00040         Any __data = {},
00041         EMathNodeType __type = EMathNodeType::None,
00042         Int32 __scope = -1
00043     );
00044 };

```

5.12 Parser.h

```

00001 #pragma once
00002
00003 #include "Core/LexiconToken.h"
00004 #include "Core/MathNode.h"
00005 #include "Core/Lexer.h"
00006
00007 #include "Core/Utility/Types.h"
00008
00009 class Parser
00010 {
00011 public:

```

```

00012     Parser();
00013
00014     void InitParser(const Vector<LexiconToken>& tokens, const Map<UInt32, HashMap<EPriority,
Vector<UInt32>>& opIndexes, const HashMap<UInt32, Pair<UInt32, Vector<Pair<UInt32, UInt32>>&
scopeCounter);
00015
00016     Map<UInt32, Vector<Ref<MathNode>> GetExecutionFlow() const { return mExecutionFlow; }
00017
00018     Ref<MathNode> GenerateTree();
00019
00020 private:
00021     void GenerateWrappedNodes(HashMap<EPriority, Vector<UInt32>& scopeData, EPriority priority);
00022     void WrapMacros(HashMap<EPriority, Vector<UInt32>& scopeData);
00023     void GenerateNodes(const Vector<LexiconToken>& tokens);
00024
00025     HashMap<UInt32, Pair<UInt32, Vector<Pair<UInt32, UInt32>>> mScopeCounter;
00026     Map<UInt32, HashMap<EPriority, Vector<UInt32>> mOperationIndexes;
00027     Vector<Ref<MathNode> mNodes;
00028
00029     Map<UInt32, Vector<Ref<MathNode>> mExecutionFlow;
00030
00031     Ref<MathNode> mTree;
00032
00033     UInt32 mFirstIndex;
00034     UInt32 mCurrentScope;
00035     UInt32 mExecutionIndex;
00036 };

```

5.13 Solver.h

```

00001 #pragma once
00002
00003 #include "Core/Utility/Types.h"
00004
00005 #include "Core/Math/Number.h"
00006
00007 #include "Core/ExplanationSystem.h"
00008 #include "Core/MathNode.h"
00009
00010 class Solver
00011 {
00012 public:
00013     Solver();
00014     void InitSolver(const Ref<MathNode>& tree);
00015     void InitSolver(const Ref<MathNode>& tree, const Map<UInt32, Vector<Ref<MathNode>>&
executionFlow);
00016
00017     RationalNumber SolveTree();
00018
00019 private:
00020     RationalNumber RecursiveSolve(const Ref<MathNode>& node);
00021     RationalNumber ExecutionSolve();
00022
00023     Map<UInt32, Vector<Ref<MathNode>> mExecutionFlow;
00024     ExplanationSystem& mExplanationSystem;
00025     Ref<MathNode> mTree;
00026 };

```

5.14 Conversions.h

```

00001 #pragma once
00002
00003 #include "Core/Utility/Types.h"
00004
00005 namespace Mathematica
00006 {
00007     // TODO : Add more conversions.
00008     namespace Convert
00009     {
00010         String UInt32ToHexString(UInt32 number, UInt32 length);
00011         String UInt32ToOctString(UInt32 number, UInt32 length);
00012
00013         String UInt32ToBaseString(UInt32 number, UInt32 base, UInt32 length);
00014
00015         Int32 StringToInt32(String string);
00016
00017         String Int32ToString(Int32 number);
00018
00019         Float32 StringToFloat32(String string);

```

```

00020         Float64 StringToFloat64(String string);
00021
00022         String Float32ToString(Float32 number);
00023         String Float64ToString(Float64 number);
00024     }
00025 }

```

5.15 Profiler.h

```

00001 #pragma once
00002
00003 #include "Core/Utility/Types.h"
00004 #include "Core/Utility/Timer.h"
00005
00006 #ifdef MTH_ENABLE_PROFILER
00007 #define MTH_PROFILE_SCOPE(name) Timer timer##__LINE__(name)
00008 #define MTH_PROFILE_FUNCTION() MTH_PROFILE_SCOPE(__FUNCSIG__)
00009 #define MTH_PROFILE_BEGIN(...) Profiler::Get().BeginProfile(__VA_ARGS__);
00010 #define MTH_PROFILE_END() Profiler::Get().EndProfile();
00011 #else
00012 #define MTH_PROFILE_SCOPE(name)
00013 #define MTH_PROFILE_FUNCTION()
00014 #define MTH_PROFILE_BEGIN(...)
00015 #define MTH_PROFILE_END()
00016 #endif
00017
00018 struct ProfileInformation
00019 {
00020     String name;
00021     Int32 start;
00022     Int32 end;
00023     Int32 duration;
00024
00025     Int32 processId;
00026     Int32 threadId;
00027 };
00028
00029 struct ProfilerSession
00030 {
00031     String name;
00032 };
00033
00034 class Profiler
00035 {
00036 public:
00037     Profiler();
00038     void BeginProfile(const String& name, String outFilePath = "");
00039     void WriteProfile(const ProfileInformation& information);
00040     void EndProfile();
00041
00042     static Profiler& Get();
00043
00044 private:
00045     void WriteHeader();
00046     void WriteFooter();
00047
00048     Scope<ProfilerSession> mCurrentSession;
00049     OutFile mOutputFile;
00050     Int32 mProfileCount;
00051 };

```

5.16 Random.h

```

00001 #pragma once
00002
00003 class RandomEngine
00004 {
00005 public:
00006     static void Init();
00007
00008     static Int32 Int(Int32 minRange = 0, Int32 maxRange = 0x7fffffff);
00009     static UInt32 UnsignedInt(UInt32 minRange = 0, UInt32 maxRange = 0xffffffff);
00010     static Float32 Float(Float32 minRange = 0.0, Float32 maxRange = 1.0);
00011     static Float64 Double(Float64 minRange = 0.0, Float64 maxRange = 1.0);
00012
00013     ~RandomEngine() = default;
00014
00015 private:
00016

```

```

00017     RandomEngine();
00018
00019     static RandomDevice mRandomDevice;
00020     static MersenneTwister mMersenneTwister;
00021     static RandomEngine* sInstance;
00022 };

```

5.17 Timer.h

```

00001 #pragma once
00002
00003 #include "Core/Utility/Types.h"
00004
00005 class Timer
00006 {
00007 public:
00008     Timer(const String& name);
00009     ~Timer();
00010
00011     void Start();
00012     void Stop();
00013
00014 private:
00015     String mName;
00016     TimePoint<SteadyClock> mStartTimepoint;
00017     bool mRunning;
00018 };

```

5.18 Types.h

```

00001 #pragma once
00002
00003 // Forward declarations
00004 struct RationalNumber;
00005 struct RealNumber;
00006
00007 // Data structures
00008 using String = std::string;
00009 using StringStream = std::stringstream;
00010 using Any = std::any;
00011
00012 template<typename T>
00013 using Vector = std::vector<T>;
00014
00015 template<typename T, size_t S>
00016 using Array = std::array<T, S>;
00017
00018 template<typename... Args>
00019 using HashMap = std::unordered_map<Args...>;
00020
00021 template<typename K, typename V>
00022 using Map = std::map<K, V>;
00023
00024 template<typename F, typename S>
00025 using Pair = std::pair<F, S>;
00026
00027 // Files
00028 using OutFile = std::ofstream;
00029 using InFile = std::ifstream;
00030 using File = std::fstream;
00031
00032 // Chrono
00033 template<typename D>
00034 using TimePoint = std::chrono::time_point<D>;
00035
00036 using SteadyClock = std::chrono::steady_clock;
00037
00038 using Nanoseconds = std::chrono::nanoseconds;
00039 using Milliseconds = std::chrono::milliseconds;
00040 using Microseconds = std::chrono::microseconds;
00041
00042 // Miscellaneous
00043 using MersenneTwister = std::mt19937;
00044 using RandomDevice = std::random_device;
00045
00046 // Functions
00047 typedef RationalNumber(*FRationalBinaryRational)(const RationalNumber&, const RationalNumber&);
00048 typedef RealNumber(*FRealBinaryRational)(const RationalNumber&, const RationalNumber&);
00049

```



```

00050 // Primitive types
00051 using UInt64 = unsigned long long int;
00052 using UInt32 = unsigned int;
00053 using UInt16 = unsigned short;
00054 using UInt8 = unsigned char;
00055 using Int64 = long long int;
00056 using Int32 = int;
00057 using Int16 = short;
00058 using Int8 = char;
00059 using Float32 = float;
00060 using Float64 = double;
00061
00062 // Smart pointers
00063 template<typename T>
00064 using Ref = std::shared_ptr<T>;
00065 template<typename T>
00066 using Scope = std::unique_ptr<T>;

```

5.19 Utils.h

```

00001 #pragma once
00002
00003 #include "Core/Utility/Types.h"
00004
00005 #ifdef MTH_DEBUG
00006 #define MTH_ASSERT(expression, message) if(!(expression)) Mathematica::Assert(#expression,
Mathematica::RelativeToBuildPath(__FILE__).c_str(), __FUNCTION__, __LINE__, message)
00007 #define MTH_DEBUG_INFO(function) DisplayFunctionInfo(#function, __FUNCTION__); function
00008 #else
00009 #define MTH_ASSERT(expression, message)
00010 #define MTH_DEBUG_INFO(function) function;
00011 #endif
00012
00013 #ifdef MTH_WIN
00014 constexpr auto MTH_PROJECT_PATH = "MathematicaCLI\\";
00015 #else
00016 constexpr auto MTH_PROJECT_PATH = "MathematicaCLI/";
00017
00018 #ifndef __FUNCSIG__
00019 #define __FUNCSIG__ __FUNCTION__
00020 #endif
00021
00022 #endif
00023
00024 #define MTH_UNUSED(x) Mathematica::Cast<void>(x)
00025 #define MTH_ADDRESS_OF(x) Mathematica::Cast<void*>(&x)
00026 #define MTH_UINT_ADDRESS_OF(x) Mathematica::Cast<UInt32*>(MTH_ADDRESS_OF(x))
00027
00028 #define MTH_HIGH_WORD(x) Mathematica::Cast<UInt8>((x >> 8) & MTH_WORD_MASK)
00029 #define MTH_LOW_WORD(x) Mathematica::Cast<UInt8>(x & MTH_WORD_MASK)
00030 #define MTH_HIGH_DWORD(x) Mathematica::Cast<UInt16>((x >> 16) & MTH_DWORD_MASK)
00031 #define MTH_LOW_DWORD(x) Mathematica::Cast<UInt16>(x & MTH_DWORD_MASK)
00032 #define MTH_HIGH_QWORD(x) Mathematica::Cast<UInt32>((x >> 32) & MTH_QWORD_MASK)
00033 #define MTH_LOW_QWORD(x) Mathematica::Cast<UInt32>(x & MTH_QWORD_MASK)
00034
00035 constexpr auto MTH_VERSION = "Version 0.0.16a";
00036 constexpr auto MTH_NO_MESSAGE = "No message provided.";
00037 constexpr auto MTH_FLOAT32_EPSILON = 1.192092896e-07F;
00038
00039 constexpr auto MTH_QWORD_MASK = 0xffffffff;
00040 constexpr auto MTH_DWORD_MASK = 0xffff;
00041 constexpr auto MTH_WORD_MASK = 0xff;
00042
00043 enum class ELexiconTokenType;
00044 enum class EMathNodeType;
00045
00046 struct LexiconToken;
00047 struct MathNode;
00048
00049 namespace Mathematica
00050 {
00051     // === Smart pointers ===
00052     template<typename T, typename ... Args>
00053     constexpr Scope<T> MakeScope(Args&& ... args)
00054     {
00055         return std::make_unique<T>(std::forward<Args>(args)...);
00056     }
00057
00058     template<typename T, typename ... Args>
00059     constexpr Ref<T> MakeRef(Args&&... args)
00060     {
00061         return std::make_shared<T>(std::forward<Args>(args)...);
00062     }

```

```

00063
00064 // === Pair ===
00065 template <typename F, typename S>
00066 constexpr Pair<F, S> MakePair(F&& first, S&& second)
00067 {
00068     return std::make_pair<F, S>(std::forward<F>(first), std::forward<S>(second));
00069 }
00070
00071 // === Any ===
00072 template <typename T>
00073 constexpr auto AnyCast(const Any& value)
00074 {
00075     return std::any_cast<T>(value);
00076 }
00077
00078 // === Casts ===
00079 template <typename F, typename S>
00080 constexpr auto Cast(const S& what)
00081 {
00082     return static_cast<F>(what);
00083 }
00084
00085 template <typename T>
00086 constexpr auto Recast(const void* what)
00087 {
00088     return reinterpret_cast<T>(what);
00089 }
00090
00091 // === Debug and files ===
00092 void Assert(const char* expression, const char* file, const char* function, Int32 line, const
char* message);
00093 void DisplayFunctionInfo(const char* functionName, const char* callerFunction);
00094 String RelativeToBuildPath(String file);
00095
00096 // === Miscellaneous ===
00097 void ClearScreen();
00098 // TODO : Add implementations for Number as well.
00099 Int32 Max(Int32 a, Int32 b);
00100 Int32 Min(Int32 a, Int32 b);
00101 UInt32 Max(UInt32 a, UInt32 b);
00102 UInt32 Min(UInt32 a, UInt32 b);
00103
00104 template<typename T>
00105 void Swap(T& a, T& b);
00106
00107 template<typename T>
00108 T GetValueFromAnyCast(const Any& data);
00109
00110 // === Functions ===
00111 FRationalBinaryRational GetBinaryFunctionFromRawData(const String& data);
00112 char ToChar(FRationalBinaryRational address);
00113 String Stringify(FRationalBinaryRational address);
00114
00115 // === String manipulation ===
00116 // * The following functions are not locale-safe.
00117 // * This might change in the future.
00118 void TransformToLower(String& string);
00119 void TransformToUpper(String& string);
00120
00121 bool IsLetter(char Char);
00122 bool FindAt(String string, String substr, UInt32 where);
00123
00124 Vector<String> SeparateString(String string, char separator = ' ');
00125 void Replace(String& string, char what, char with);
00126 void RemoveQuotes(String& string);
00127
00128 // === Token ===
00129 void DisplayTokenArray(const Vector<LexiconToken>& tokenArray, bool bInline = true);
00130 void DisplayTokenUUID(const Vector<LexiconToken>& tokenArray, bool bInline = true);
00131 String Stringify(ELexiconTokenType type);
00132
00133 // === Tree ===
00134 void DisplayParsedTree(const Ref<MathNode>& node);
00135 String Stringify(EMathNodeType type);
00136 String Stringify(RationalNumber number);
00137 };

```

Index

Build/Include/Core/ExplanationSystem.h, [19](#)
Build/Include/Core/Hashable.h, [19](#)
Build/Include/Core/Identifiable.h, [20](#)
Build/Include/Core/Lexer.h, [20](#)
Build/Include/Core/LexiconToken.h, [21](#)
Build/Include/Core/Math/Integer.h, [21](#)
Build/Include/Core/Math/Number.h, [22](#)
Build/Include/Core/Math/Operations.h, [24](#)
Build/Include/Core/Math/Rational.h, [24](#)
Build/Include/Core/Math/Real.h, [24](#)
Build/Include/Core/MathNode.h, [25](#)
Build/Include/Core/Parser.h, [25](#)
Build/Include/Core/Solver.h, [26](#)
Build/Include/Core/Utility/Conversions.h, [26](#)
Build/Include/Core/Utility/Profiler.h, [27](#)
Build/Include/Core/Utility/Random.h, [27](#)
Build/Include/Core/Utility/Timer.h, [28](#)
Build/Include/Core/Utility/Types.h, [28](#)
Build/Include/Core/Utility/Utils.h, [29](#)

CompareHashable< HashableObject >, [7](#)

ExplanationSystem, [7](#)

Hashable, [8](#)
HashBinding< HashableObject >, [8](#)

Identifiable, [9](#)
IrrationalNumber, [9](#)
 Rehash, [10](#)
IrrationalPart, [10](#)
 Rehash, [12](#)

Lexer, [12](#)
LexiconToken, [12](#)

MathExpression, [13](#)
 Rehash, [14](#)
MathNode, [14](#)

Parser, [15](#)
ProfileInformation, [16](#)
Profiler, [16](#)
ProfilerSession, [16](#)

RandomEngine, [17](#)
RationalNumber, [17](#)
RealNumber, [18](#)
Rehash
 IrrationalNumber, [10](#)
 IrrationalPart, [12](#)
 MathExpression, [14](#)
 Solver, [18](#)
 Timer, [18](#)