



UNIVERSIDAD DE MÁLAGA

UNIVERSITY OF MALAGA
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

SyntaxTutor

Developer Manual

Version: 1.0.4

Date: 2025-11-08

Author: jose-rzm at GitHub



Chapter 1

SyntaxTutor: An interactive Tool for Learning Syntax Analysis

[SyntaxTutor](#) is an educational application designed to help compiler students understand LL(1) and SLR(1) parsing algorithms. Through a visual and interactive interface, it guides users step-by-step through the computation of FIRST, FOLLOW, CLOSURE, GOTO, predictive parsing tables, and LR automata, offering real-time pedagogical feedback.

Rather than acting as a mere calculator, [SyntaxTutor](#) functions as a learning companion. It explains the reasoning behind each step, highlights common mistakes, and encourages students to engage with the theory behind the algorithms.

1.1 Academic Context

[SyntaxTutor](#) is part of a Final Degree Project (TFG) developed at the University of Málaga (UMA), in the Computer Engineering program. Its main goal is to offer an educational companion for students learning syntax analysis, going beyond traditional calculators by incorporating guided feedback, visualization, and gamified learning.

1.2 Key Features

- **Educational Focus:** built to teach, not just compute.
 - **Visualization:** derivation trees, intermediate steps, sets, and tables.
 - **Exportable Results:** useful for reports or coursework.
-

1.3 Interface Screenshots

1.3.1 Main Menu

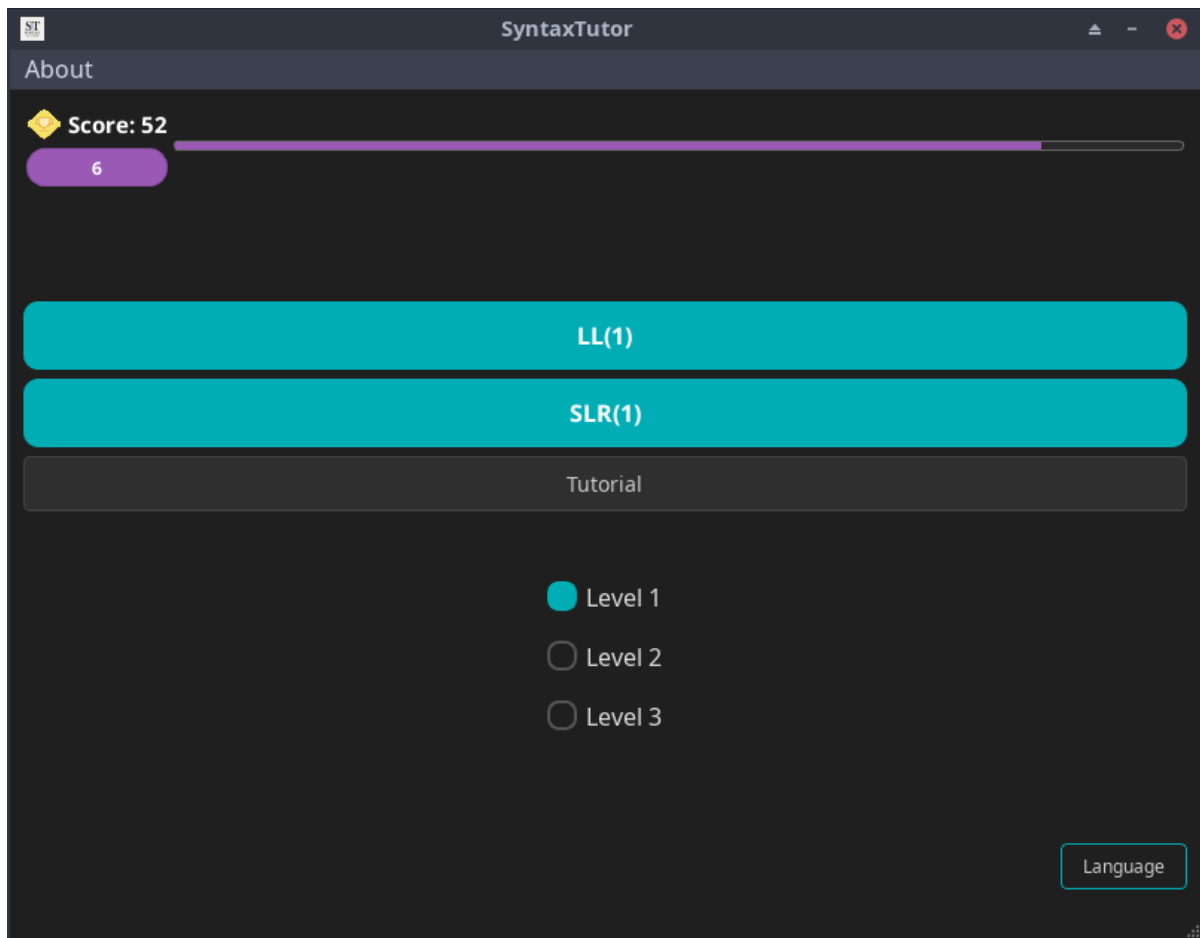


Figure 1.1 Main window

Home screen with gamification, levels, and language options.

1.3.2 LL(1) Learning Mode

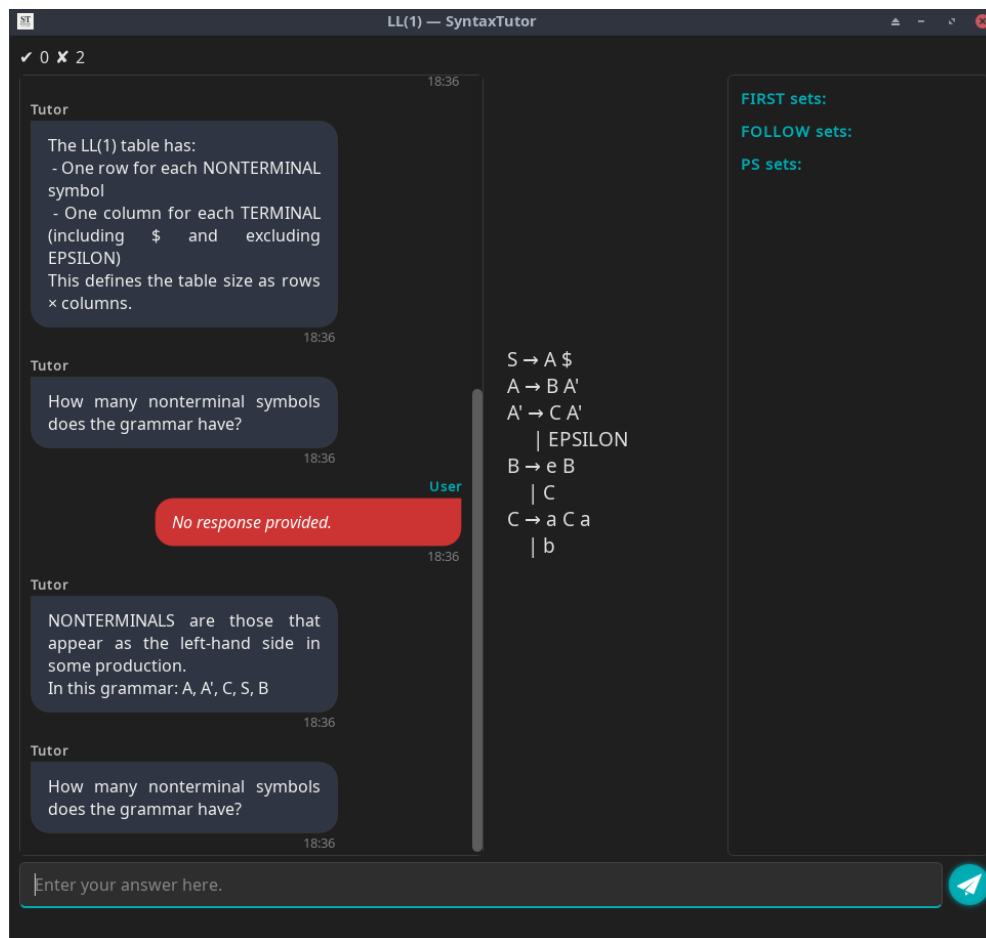


Figure 1.2 LL(1) dialog view

Interactive LL(1) tutor asks questions and provides feedback.

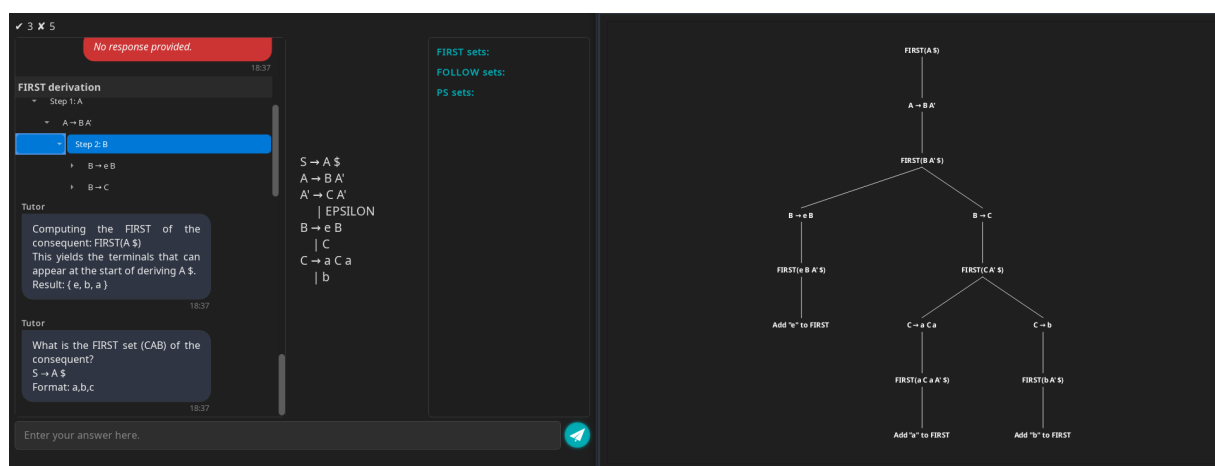


Figure 1.3 LL(1) derivation tree

Derivation tree view showing how FIRST sets are built step-by-step.

LL(1) — SyntaxTutor

✓ 16 ✕ 6

$C' \rightarrow \text{EPSILON}$
Format: a,b,c

Tutor

So, what are the symbols (SD) of $C' \rightarrow \text{EPSILON}$?
Format: a,b,c

a,g

Tutor

Fill in the LL(1) table you calculated during the exercise.

18:52

Enter your answer here.

Complete LL(1) table — SyntaxTutor

	\$	a	b	g
S		A\$		A\$
A		BgA'		BgA'
A'		A		A
B		B'		B'
B'				
C				
C'				

Finish

FOLLOW sets:

- $\text{FIRST}(B') = \{a, \text{EPSILON}\}$
- $\text{FIRST}(C B') = \{a\}$
- $\text{FIRST}(\text{EPSILON}) = \{\text{EPSILON}\}$
- $\text{FIRST}(a C') = \{a\}$
- $\text{FIRST}(b C) = \{b\}$

PS sets:

- $\text{FOLLOW}(A) = \{\$ \}$
- $\text{FOLLOW}(A') = \{\$ \}$
- $\text{FOLLOW}(B) = \{g\}$
- $\text{FOLLOW}(B') = \{g\}$
- $\text{FOLLOW}(C) = \{a, g\}$
- $\text{FOLLOW}(C') = \{a, g\}$

PS sets:

- $\text{PS}(A \rightarrow B g A') = \{a, g\}$
- $\text{PS}(A' \rightarrow A) = \{a, g\}$
- $\text{PS}(A' \rightarrow \text{EPSILON}) = \{\$ \}$
- $\text{PS}(B \rightarrow B') = \{a, g\}$
- $\text{PS}(B' \rightarrow C B') = \{a\}$
- $\text{PS}(B' \rightarrow \text{EPSILON}) = \{g\}$
- $\text{PS}(C \rightarrow a C') = \{a\}$
- $\text{PS}(C' \rightarrow \text{EPSILON}) = \{a, g\}$
- $\text{PS}(C' \rightarrow b C) = \{b\}$
- $\text{PS}(S \rightarrow A \$) = \{a, g\}$

Figure 1.4 LL(1) table task

Completion of the LL(1) predictive table with visual guidance.

1.3.3 SLR(1) Learning Mode

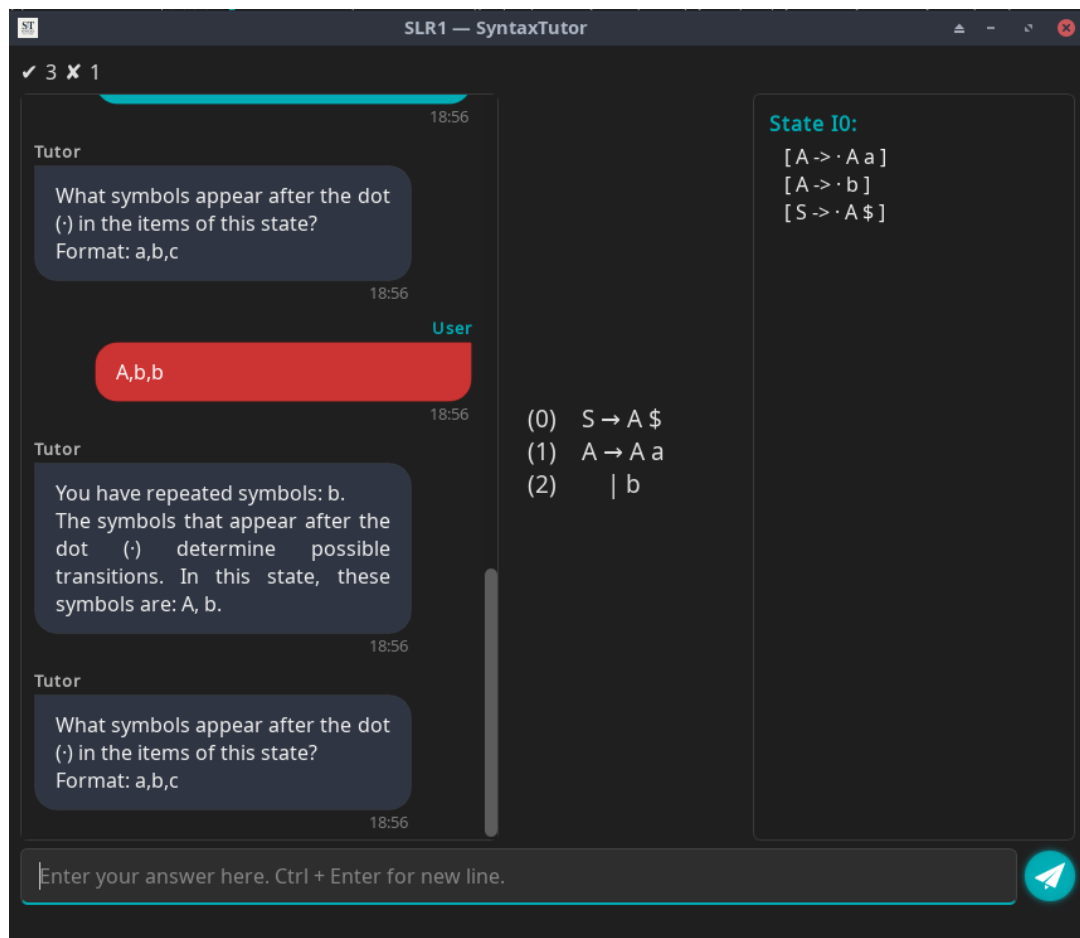


Figure 1.5 SLR(1) item view

User is asked to identify symbols after the dot in an LR(0) item.

Tutor

Be I:

- [$A \rightarrow \cdot A a$]
- [$A \rightarrow \cdot b$]
- [$S \rightarrow \cdot A \$$]

To find $\delta(I, A)$:

1. Look for items with A after the \cdot .
That is, items of the form $\alpha A \beta$
2. Be J:

- [$S \rightarrow \cdot A \$$]
- [$A \rightarrow \cdot A a$]

3. Advance the \cdot one position:

- [$A \rightarrow A \cdot a$]
- [$S \rightarrow A \cdot \$$]

4. $\delta(I, A) = \text{CLOSURE}(J)$

5. Closure of J:

- [$A \rightarrow A \cdot a$]
- [$S \rightarrow A \cdot \$$]

18:56

(0) $S \rightarrow A \$$
 (1) $A \rightarrow A a$
 (2) $\quad | b$

State I0:

- [$A \rightarrow \cdot A a$]
- [$A \rightarrow \cdot b$]
- [$S \rightarrow \cdot A \$$]

Transitions:

- $\delta(I0, A) = I2$

State I2:

- [$S \rightarrow A \cdot \$$]
- [$A \rightarrow A \cdot a$]

Enter your answer here. Ctrl + Enter for new line.

Figure 1.6 SLR(1) automaton construction

Step-by-step explanation of the GOTO/closure construction.

SLR1 — SyntaxTutor

✓ 0 ✕ 0

1 Complete SLR table — SyntaxTutor

	a	b	c	e	\$	A	B	S
State 0								
State 1								
State 2								
State 3								
State 4								
State 5								
State 6								
State 7								

Finish

Enter your answer here. Ctrl + Enter for new line.

Figure 1.7 SLR(1) table fill-in

Interactive SLR(1) table to complete, with states and terminals/non-terminals.

1.3.4 Assisted Mode: Guided Table Completion

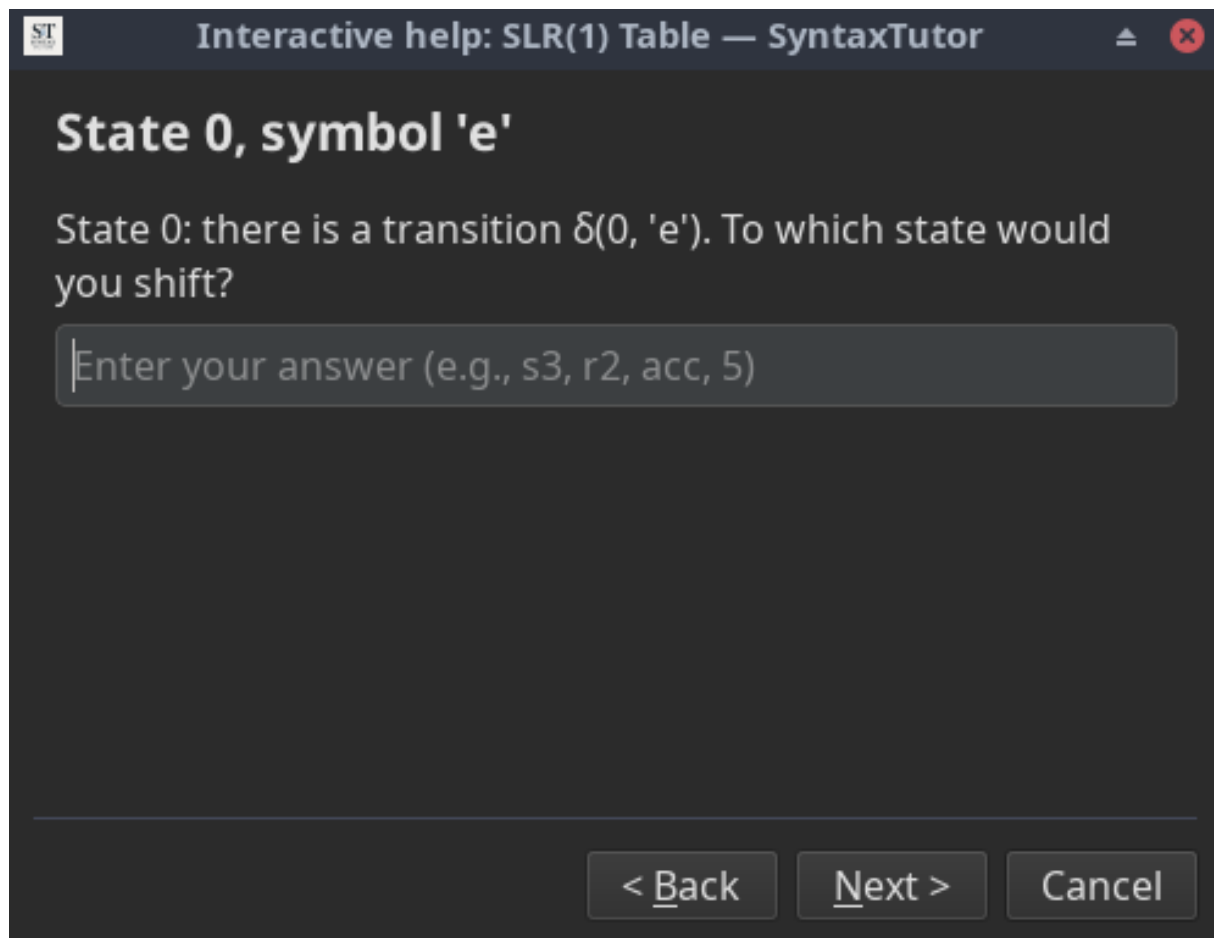


Figure 1.8 SLR(1) guided mode

[SyntaxTutor](#) walks the student through each cell in the parsing table with hints and context.

1.4 Technologies Used

- **C++**: efficient implementation of parsing algorithms
- **Qt6**: modern, cross-platform graphical user interface.
- **Modular architecture**: clean separation between logic and UI, designed for easy extensibility.

1.5 Downloads

Precompiled builds of [SyntaxTutor](#) are available in the Releases tab:

- Linux (X11): executable AppImage
- Windows: ZIP archive with the .exe
- macOS: .app bundles for both Apple Silicon (ARM) and Intel

Warning

The Windows and macOS versions are not digitally signed. Your operating system may display a warning when running the application. You can bypass it manually if you trust the source.

1.6 Building from Source

To build [SyntaxTutor](#) from source, you just need:

- Qt6 (including `qmake6`)
- A C++20-compliant compiler

```
qmake6  
make
```

1.6.1 Documentation

Full documentation for the source code is available via Doxygen:

- **Online HTML Documentation:** <https://jose-rzm.github.io/SyntaxTutor/>
- **PDF Reference Manual:** [refman.pdf](#) (in the `docs/latex/` folder)

The documentation includes:

- Detailed class and function reference
- Graphs of dependencies and inheritance
- Descriptions of parsing algorithms and internal logic

To regenerate it locally, install [Doxygen](#) and run:

```
doxygen
```

This will update the contents of the `docs/` folder with both HTML and LaTeX output.

Chapter 2

Directory Hierarchy

2.1 Directories

app	21
main.cpp	??
backend	21
grammar.cpp	??
grammar.hpp	??
grammar_factory.cpp	??
grammar_factory.hpp	??
ll1_parser.cpp	??
ll1_parser.hpp	??
lr0_item.cpp	??
lr0_item.hpp	??
slr1_parser.cpp	??
slr1_parser.hpp	??
state.hpp	??
symbol_table.cpp	??
symbol_table.hpp	??
gui	22
lftabledialog.cpp	??
lftabledialog.h	??
lftutorwindow.cpp	??
lftutorwindow.h	??
mainwindow.cpp	??
mainwindow.h	??
slrtabledialog.cpp	??
slrtabledialog.h	??
slrtutorwindow.cpp	??
slrtutorwindow.h	??
slrwizard.h	??
slrwizardpage.h	??
UniqueQueue.h	??
widgets	22
customtextedit.cpp	??
customtextedit.h	??
tutorialmanager.cpp	??
tutorialmanager.h	??

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

SyntaxTutor	23
SyntaxTutor::Version	23

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

GrammarFactory::FactoryItem	27
Grammar	28
GrammarFactory	32
LL1Parser	47
Lr0Item	??
QDialog	
LLTableDialog	??
SLRTableDialog	??
QMainWindow	
LLTutorWindow	??
MainWindow	??
SLRTutorWindow	??
QObject	
TutorialManager	??
QStyledItemDelegate	
CenterAlignDelegate	25
CenterAlignDelegate	25
QTextEdit	
CustomTextEdit	26
QWizard	
SLRWizard	??
QWizardPage	
SLRWizardPage	??
SLR1Parser::s_action	??
SLR1Parser	??
state	??
SymbolTable	??
LLTutorWindow::TreeNode	??
TutorialStep	??
UniqueQueue< T >	??

Chapter 5

Class Index

5.1 Class List

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

CenterAlignDelegate	25
CustomTextEdit	26
GrammarFactory::FactoryItem	
Represents an individual grammar item with its associated symbol table	27
Grammar	
Represents a context-free grammar, including its rules, symbol table, and starting symbol	28
GrammarFactory	
Responsible for creating and managing grammar items and performing checks on grammars	32
LL1Parser	47
LLTableDialog	
Dialog for filling and submitting an LL(1) parsing table	??
LLTutorWindow	
Main window for the LL(1) interactive tutoring mode in SyntaxTutor	??
Lr0Item	
Represents an LR(0) item used in LR automata construction	??
MainWindow	
Main application window of SyntaxTutor , managing levels, exercises, and UI state	??
SLR1Parser::s_action	??
SLR1Parser	
Implements an SLR(1) parser for context-free grammars	??
SLRTableDialog	
Dialog window for completing and submitting an SLR(1) parsing table	??
SLRTutorWindow	
Main window for the SLR(1) interactive tutoring mode in SyntaxTutor	??
SLRWizard	
Interactive assistant that guides the student step-by-step through the SLR(1) parsing table	??
SLRWizardPage	
A single step in the SLR(1) guided assistant for table construction	??
state	
Represents a state in the LR(0) automaton	??
SymbolTable	
Stores and manages grammar symbols, including their classification and special markers	??
LLTutorWindow::TreeNode	
TreeNode structure used to build derivation trees	??
TutorialManager	
Manages interactive tutorials by highlighting UI elements and guiding the user	??
TutorialStep	
Represents a single step in the tutorial sequence	??
UniqueQueue< T >	
A queue that ensures each element is inserted only once	??

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

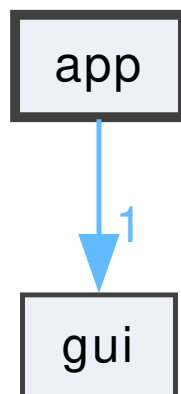
appversion.h	??
app/main.cpp	??
backend/grammar.cpp	??
backend/grammar.hpp	??
backend/grammar_factory.cpp	??
backend/grammar_factory.hpp	??
backend/ll1_parser.cpp	??
backend/ll1_parser.hpp	??
backend/lr0_item.cpp	??
backend/lr0_item.hpp	??
backend/slr1_parser.cpp	??
backend/slr1_parser.hpp	??
backend/state.hpp	??
backend/symbol_table.cpp	??
backend/symbol_table.hpp	??
gui/lltabledialog.cpp	??
gui/lltabledialog.h	??
gui/lltutorwindow.cpp	??
gui/lltutorwindow.h	??
gui/mainwindow.cpp	??
gui/mainwindow.h	??
gui/slrtabledialog.cpp	??
gui/slrtabledialog.h	??
gui/slrtutorwindow.cpp	??
gui/slrtutorwindow.h	??
gui/slrwizard.h	??
gui/slrwizardpage.h	??
gui/UniqueQueue.h	??
widgets/customtextedit.cpp	??
widgets/customtextedit.h	??
widgets/tutorialmanager.cpp	??
widgets/tutorialmanager.h	??

Chapter 7

Directory Documentation

7.1 app Directory Reference

Directory dependency graph for app:



Files

- file [main.cpp](#)

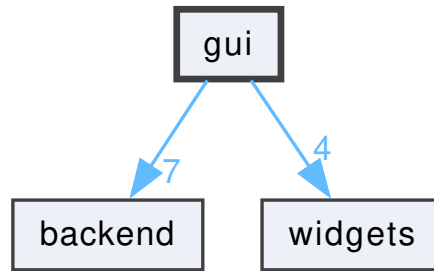
7.2 backend Directory Reference

Files

- file [grammar.cpp](#)
- file [grammar.hpp](#)
- file [grammar_factory.cpp](#)
- file [grammar_factory.hpp](#)
- file [ll1_parser.cpp](#)
- file [ll1_parser.hpp](#)
- file [lr0_item.cpp](#)
- file [lr0_item.hpp](#)
- file [slr1_parser.cpp](#)
- file [slr1_parser.hpp](#)
- file [state.hpp](#)
- file [symbol_table.cpp](#)
- file [symbol_table.hpp](#)

7.3 gui Directory Reference

Directory dependency graph for gui:



Files

- file [lltabledialog.cpp](#)
- file [lltabledialog.h](#)
- file [lltutorwindow.cpp](#)
- file [lltutorwindow.h](#)
- file [mainwindow.cpp](#)
- file [mainwindow.h](#)
- file [slrtabledialog.cpp](#)
- file [slrtabledialog.h](#)
- file [slrtutorwindow.cpp](#)
- file [slrtutorwindow.h](#)
- file [slrwizard.h](#)
- file [slrwizardpage.h](#)
- file [UniqueQueue.h](#)

7.4 widgets Directory Reference

Files

- file [customtextedit.cpp](#)
- file [customtextedit.h](#)
- file [tutorialmanager.cpp](#)
- file [tutorialmanager.h](#)

Chapter 8

Namespace Documentation

8.1 SyntaxTutor Namespace Reference

Namespaces

- namespace [Version](#)

8.2 SyntaxTutor::Version Namespace Reference

Functions

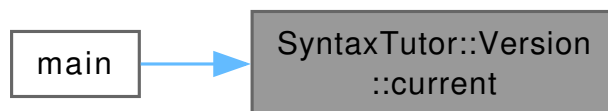
- QString [current](#) ()
- const char * [raw](#) ()

8.2.1 Function Documentation

8.2.1.1 current()

```
QString SyntaxTutor::Version::current () [inline]
```

Here is the caller graph for this function:



8.2.1.2 raw()

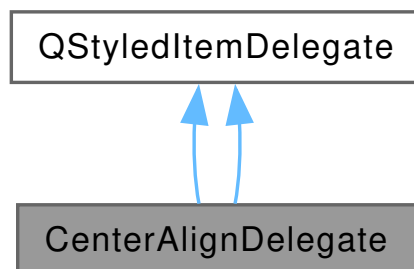
```
const char * SyntaxTutor::Version::raw () [inline]
```


Chapter 9

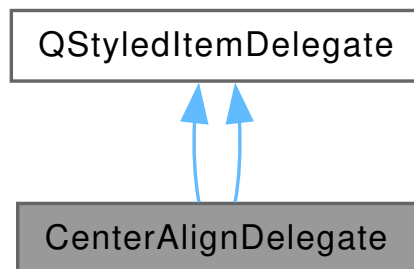
Class Documentation

9.1 CenterAlignDelegate Class Reference

Inheritance diagram for CenterAlignDelegate:



Collaboration diagram for CenterAlignDelegate:



Public Member Functions

- void [initStyleOption](#) (QStyleOptionViewItem *opt, const QModelIndex &idx) const override
- void [initStyleOption](#) (QStyleOptionViewItem *opt, const QModelIndex &idx) const override

9.1.1 Member Function Documentation

9.1.1.1 `initStyleOption()` [1/2]

```
void CenterAlignDelegate::initStyleOption (  
    QStyleOptionViewItem * opt,  
    const QModelIndex & idx) const [inline], [override]
```

9.1.1.2 `initStyleOption()` [2/2]

```
void CenterAlignDelegate::initStyleOption (
    QStyleOptionViewItem * opt,
    const QModelIndex & idx) const [inline], [override]
```

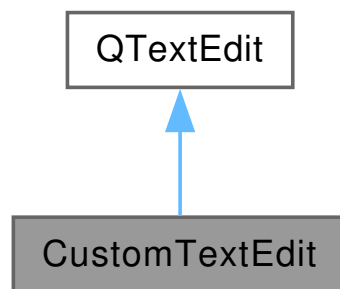
The documentation for this class was generated from the following files:

- [gui/lltabledialog.cpp](#)
- [gui/sltabledialog.cpp](#)

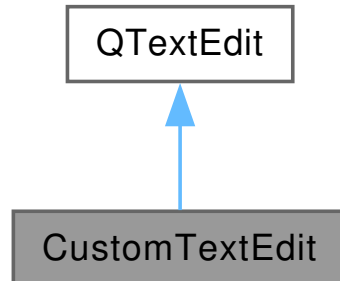
9.2 CustomTextEdit Class Reference

```
#include <customtextedit.h>
```

Inheritance diagram for CustomTextEdit:



Collaboration diagram for CustomTextEdit:



Signals

- void [sendRequested](#) ()

Public Member Functions

- [CustomTextEdit](#) (QWidget *parent=nullptr)

Protected Member Functions

- void [keyPressEvent](#) (QKeyEvent *event) override

9.2.1 Constructor & Destructor Documentation

9.2.1.1 CustomTextEdit()

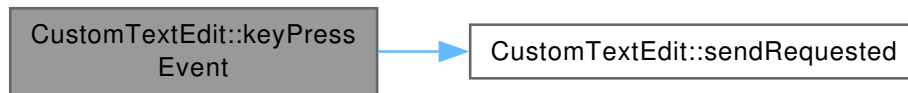
```
CustomTextEdit::CustomTextEdit (
    QWidget * parent = nullptr) [explicit]
```

9.2.2 Member Function Documentation

9.2.2.1 keyPressEvent()

```
void CustomTextEdit::keyPressEvent (
    QKeyEvent * event) [override], [protected]
```

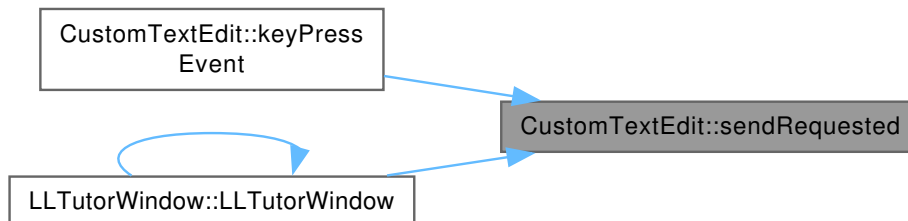
Here is the call graph for this function:



9.2.2.2 sendRequested

```
void CustomTextEdit::sendRequested () [signal]
```

Here is the caller graph for this function:



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

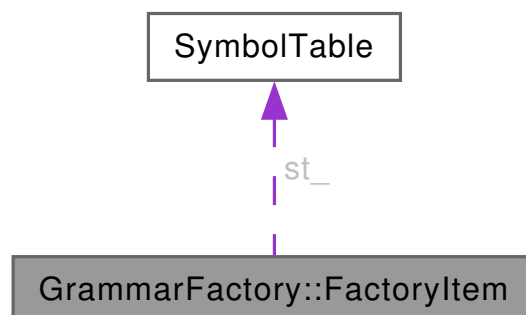
- [widgets/customtextedit.h](#)
- [widgets/customtextedit.cpp](#)

9.3 GrammarFactory::FactoryItem Struct Reference

Represents an individual grammar item with its associated symbol table.

```
#include <grammar_factory.hpp>
```

Collaboration diagram for GrammarFactory::FactoryItem:



Public Member Functions

- [FactoryItem](#) (const std::unordered_map< std::string, std::vector< [production](#) > > &grammar)
Constructor that initializes a [FactoryItem](#) with the provided grammar.

Public Attributes

- `std::unordered_map< std::string, std::vector< production > > g_`
Stores the grammar rules where each key is a non-terminal symbol and each value is a vector of production rules.
- `SymbolTable st_`
Symbol table associated with this grammar item.

9.3.1 Detailed Description

Represents an individual grammar item with its associated symbol table.

9.3.2 Constructor & Destructor Documentation

9.3.2.1 FactoryItem()

```
GrammarFactory::FactoryItem::FactoryItem (
    const std::unordered_map< std::string, std::vector< production > > & grammar)
[explicit]
```

Constructor that initializes a [FactoryItem](#) with the provided grammar.

Parameters

<i>grammar</i>	The grammar to initialize the FactoryItem with.
----------------	---

9.3.3 Member Data Documentation

9.3.3.1 g_

```
std::unordered_map<std::string, std::vector<production> > GrammarFactory::FactoryItem::g_
```

Stores the grammar rules where each key is a non-terminal symbol and each value is a vector of production rules.

9.3.3.2 st_

```
SymbolTable GrammarFactory::FactoryItem::st_
```

Symbol table associated with this grammar item.

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

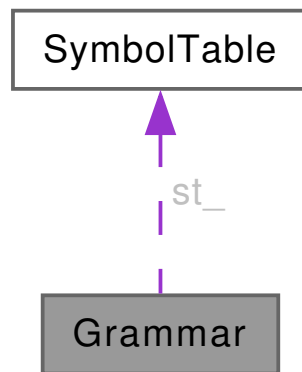
- `backend/grammar_factory.hpp`
- `backend/grammar_factory.cpp`

9.4 Grammar Struct Reference

Represents a context-free grammar, including its rules, symbol table, and starting symbol.

```
#include <grammar.hpp>
```

Collaboration diagram for Grammar:



Public Member Functions

- [Grammar](#) ()
- [Grammar](#) (const std::unordered_map< std::string, std::vector< [production](#) > > &grammar)
- void [SetAxiom](#) (const std::string &axiom)
Sets the axiom (entry point) of the grammar.
- bool [HasEmptyProduction](#) (const std::string &antecedent) const
Checks if a given antecedent has an empty production.
- std::vector< std::pair< const std::string, [production](#) > > [FilterRulesByConsequent](#) (const std::string &arg) const
Filters grammar rules that contain a specific token in their consequent.
- void [Debug](#) () const
Prints the current grammar structure to standard output.
- void [AddProduction](#) (const std::string &antecedent, const std::vector< std::string > &consequent)
Adds a production rule to the grammar and updates the symbol table.
- std::vector< std::string > [Split](#) (const std::string &s)
Splits a string into grammar symbols using the current symbol table.

Public Attributes

- std::unordered_map< std::string, std::vector< [production](#) > > [g_](#)
Stores the grammar rules with each antecedent mapped to a list of productions.
- std::string [axiom_](#)
The axiom or entry point of the grammar.
- [SymbolTable](#) [st_](#)
Symbol table of the grammar.

9.4.1 Detailed Description

Represents a context-free grammar, including its rules, symbol table, and starting symbol.

This structure encapsulates all components required to define and manipulate a grammar, including production rules, the associated symbol table, and metadata such as the start symbol. It supports construction, transformation, and analysis of grammars.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 Grammar() [1/2]

```
Grammar::Grammar () [default]
```

9.4.2.2 Grammar() [2/2]

```
Grammar::Grammar (
    const std::unordered_map< std::string, std::vector< production > > & grammar)
[explicit]
```

9.4.3 Member Function Documentation

9.4.3.1 AddProduction()

```
void Grammar::AddProduction (
    const std::string & antecedent,
    const std::vector< std::string > & consequent)
```

Adds a production rule to the grammar and updates the symbol table.

This function inserts a new production of the form $A \rightarrow$ into the grammar, where `antecedent` is the non-terminal A and `consequent` is the sequence . It also updates the internal symbol table to reflect any new symbols introduced.

Parameters

<i>antecedent</i>	The left-hand side non-terminal of the production.
<i>consequent</i>	The right-hand side sequence of grammar symbols.

9.4.3.2 Debug()

```
void Grammar::Debug () const
```

Prints the current grammar structure to standard output.

This function provides a debug view of the grammar by printing out all rules, the axiom, and other relevant details.

9.4.3.3 FilterRulesByConsequent()

```
std::vector< std::pair< const std::string, production > > Grammar::FilterRulesByConsequent (
    const std::string & arg) const
```

Filters grammar rules that contain a specific token in their consequent.

Parameters

<i>arg</i>	The token to search for within the consequents of the rules.
------------	--

Returns

std::vector of pairs where each pair contains an antecedent and its respective production that includes the specified token.

Searches for rules in which the specified token is part of the consequent and returns those rules.

9.4.3.4 HasEmptyProduction()

```
bool Grammar::HasEmptyProduction (
    const std::string & antecedent) const
```

Checks if a given antecedent has an empty production.

Parameters

<i>antecedent</i>	The left-hand side (LHS) symbol to check.
-------------------	---

Returns

true if there exists an empty production for the antecedent, otherwise false.

An empty production is represented as `<antecedent> -> ;`, indicating that the antecedent can produce an empty string.

9.4.3.5 SetAxiom()

```
void Grammar::SetAxiom (
    const std::string & axiom)
```

Sets the axiom (entry point) of the grammar.

Parameters

<i>axiom</i>	The entry point or start symbol of the grammar.
--------------	---

Defines the starting point for the grammar, which is used in parsing algorithms and must be a non-terminal symbol present in the grammar.

9.4.3.6 Split()

```
std::vector< std::string > Grammar::Split (
    const std::string & s)
```

Splits a string into grammar symbols using the current symbol table.

This function tokenizes the input string `s` into a sequence of grammar symbols based on the known entries in the symbol table. It uses a greedy approach, matching the longest valid symbol at each step.

Parameters

<i>s</i>	The input string to split.
----------	----------------------------

Returns

A vector of grammar symbols extracted from the string.

9.4.4 Member Data Documentation**9.4.4.1 axiom_**

```
std::string Grammar::axiom_
```

The axiom or entry point of the grammar.

9.4.4.2 g_

```
std::unordered_map<std::string, std::vector<production> > Grammar::g_
```

Stores the grammar rules with each antecedent mapped to a list of productions.

9.4.4.3 st_

```
SymbolTable Grammar::st_
```

Symbol table of the grammar.

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

- [backend/grammar.hpp](#)
- [backend/grammar.cpp](#)

9.5 GrammarFactory Struct Reference

Responsible for creating and managing grammar items and performing checks on grammars.

```
#include <grammar_factory.hpp>
```

Classes

- struct [FactoryItem](#)
Represents an individual grammar item with its associated symbol table.

Public Member Functions

- void [Init](#) ()
Initializes the [GrammarFactory](#) and populates the items vector with initial grammar items.
- [Grammar PickOne](#) (int level)
Picks a random grammar based on the specified difficulty level (1, 2, or 3).
- [Grammar GenLL1Grammar](#) (int level)
Generates a LL(1) random grammar based on the specified difficulty level.
- [Grammar GenSLR1Grammar](#) (int level)
Generates a SLR(1) random grammar based on the specified difficulty level.
- [Grammar Lv1](#) ()
Generates a Level 1 grammar.
- [Grammar Lv2](#) ()
Generates a Level 2 grammar by combining Level 1 items.
- [Grammar Lv3](#) ()
Generates a Level 3 grammar by combining a Level 2 item and a Level 1 item.
- [Grammar Lv4](#) ()
Generates a Level 4 grammar by combining Level 3 and Level 1 items.
- [Grammar Lv5](#) ()
Generates a Level 5 grammar by combining Level 4 and Level 1 items.
- [Grammar Lv6](#) ()
Generates a Level 6 grammar by combining Level 5 and Level 1 items.
- [Grammar Lv7](#) ()
Generates a Level 7 grammar by combining Level 6 and Level 1 items.
- [FactoryItem CreateLv2Item](#) ()
Creates a Level 2 grammar item for use in grammar generation.
- bool [HasUnreachableSymbols](#) ([Grammar](#) &grammar) const
Checks if a grammar contains unreachable symbols (non-terminals that cannot be derived from the start symbol).
- bool [IsInfinite](#) ([Grammar](#) &grammar) const
Checks if a grammar is infinite, meaning there are non-terminal symbols that can never derive a terminal string. This happens when a production leads to an infinite recursion or an endless derivation without reaching terminal symbols. For example, a production like:
- bool [HasDirectLeftRecursion](#) (const [Grammar](#) &grammar) const
Checks if a grammar contains direct left recursion (a non-terminal can produce itself on the left side of a production in one step).
- bool [HasIndirectLeftRecursion](#) (const [Grammar](#) &grammar) const
Checks if a grammar contains indirect left recursion.
- bool [HasCycle](#) (const std::unordered_map< std::string, std::unordered_set< std::string > > &graph) const
Checks if directed graph has a cycle using topological sort.
- std::unordered_set< std::string > [NullableSymbols](#) (const [Grammar](#) &grammar) const
Find nullable symbols in a grammar.
- void [RemoveLeftRecursion](#) ([Grammar](#) &grammar)
Removes direct left recursion in a grammar. A grammar has direct left recursion when one of its productions is.

- void [LeftFactorize](#) ([Grammar](#) &grammar)
Performs left factorization. A grammar can be left factorized if it has productions with the same prefix for one non-terminal. For example:
- std::vector< std::string > [LongestCommonPrefix](#) (const std::vector< [production](#) > &productions)
Finds the longest common prefix among a set of productions.
- bool [StartsWith](#) (const [production](#) &prod, const std::vector< std::string > &prefix)
Checks if a production starts with a given prefix.
- std::string [GenerateNewNonTerminal](#) ([Grammar](#) &grammar, const std::string &base)
Generates a new non-terminal symbol that is unique in the grammar.
- void [NormalizeNonTerminals](#) ([FactoryItem](#) &item, const std::string &nt) const
Replaces all non-terminal symbols in a grammar item with a single target non-terminal.
- void [AdjustTerminals](#) ([FactoryItem](#) &base, const [FactoryItem](#) &cmb, const std::string &target_nt) const
Adjusts the terminal symbols between two grammar items.
- std::unordered_map< std::string, std::vector< [production](#) > > [Merge](#) (const [FactoryItem](#) &base, const [FactoryItem](#) &cmb) const
Merges the grammar rules of two grammar items into a single grammar.

Public Attributes

- std::vector< [FactoryItem](#) > [items](#)
A vector of [FactoryItem](#) objects representing different level 1 grammar items created by the Init method.
- std::vector< std::string > [terminal_alphabet_](#)
A vector of terminal symbols (alphabet) used in the grammar.
- std::vector< std::string > [non_terminal_alphabet_](#)
A vector of non-terminal symbols (alphabet) used in the grammar.

9.5.1 Detailed Description

Responsible for creating and managing grammar items and performing checks on grammars.

9.5.2 Member Function Documentation

9.5.2.1 AdjustTerminals()

```
void GrammarFactory::AdjustTerminals (
    FactoryItem & base,
    const FactoryItem & cmb,
    const std::string & target_nt) const
```

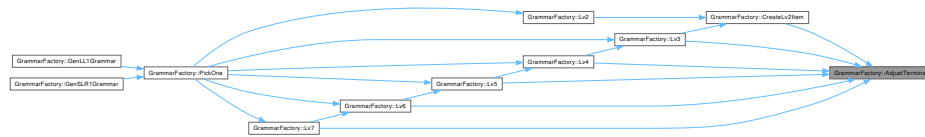
Adjusts the terminal symbols between two grammar items.

This function modifies the terminal symbols of a base grammar item so that they do not conflict with those of the item being combined. It also renames terminals to ensure consistency and inserts the target non-terminal where appropriate.

Parameters

<i>base</i>	The base grammar item to adjust.
<i>cmb</i>	The grammar item being combined with the base.
<i>target_↔ _nt</i>	The target non-terminal symbol used for replacement.

Here is the caller graph for this function:



9.5.2.2 CreateLv2Item()

```
GrammarFactory::FactoryItem GrammarFactory::CreateLv2Item ()
```

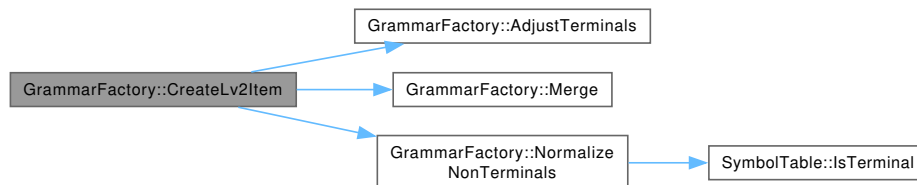
Creates a Level 2 grammar item for use in grammar generation.

This function generates a Level 2 grammar item, which can be used as a building block for creating more complex grammars.

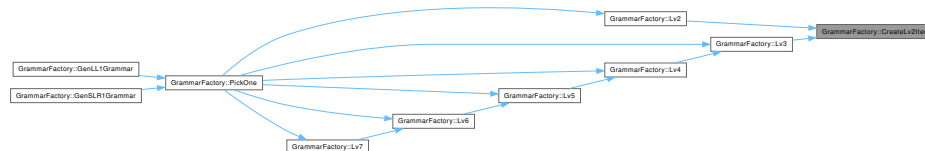
Returns

A [FactoryItem](#) representing a Level 2 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



9.5.2.3 GenerateNewNonTerminal()

```
std::string GrammarFactory::GenerateNewNonTerminal (
    Grammar & grammar,
    const std::string & base)
```

Generates a new non-terminal symbol that is unique in the grammar.

This function creates a new non-terminal symbol by appending a prime symbol (') to the base name until the resulting symbol is not already present in the grammar's symbol table. It is used during left factorization to introduce new non-terminals for factored productions.

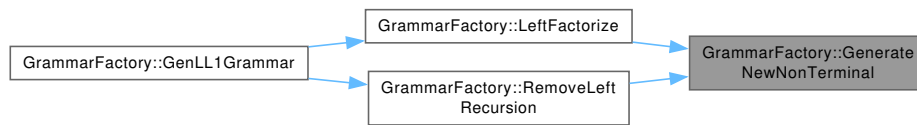
Parameters

<i>grammar</i>	The grammar in which the new non-terminal will be added.
<i>base</i>	The base name for the new non-terminal.

Returns

A unique non-terminal symbol derived from the base name.

Here is the caller graph for this function:



9.5.2.4 GenLL1Grammar()

```
Grammar GrammarFactory::GenLL1Grammar (
    int level)
```

Generates a LL(1) random grammar based on the specified difficulty level.

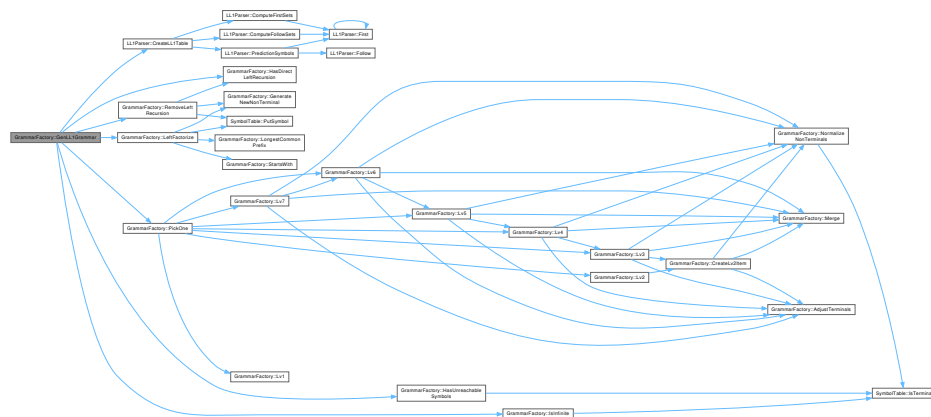
Parameters

<i>level</i>	The difficulty level (1, 2, or 3)
--------------	-----------------------------------

Returns

A random LL(1) grammar.

Here is the call graph for this function:



9.5.2.5 GenSLR1Grammar()

```
Grammar GrammarFactory::GenSLR1Grammar (
    int level)
```

Generates a SLR(1) random grammar based on the specified difficulty level.

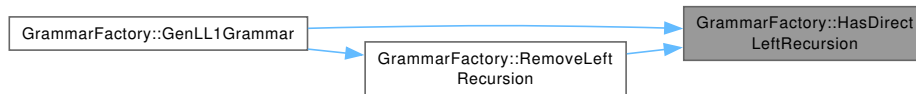
Parameters

<i>level</i>	The difficulty level (1, 2, or 3)
--------------	-----------------------------------

Returns

true if there is direct left recursion, false otherwise.

Here is the caller graph for this function:

**9.5.2.8 HasIndirectLeftRecursion()**

```
bool GrammarFactory::HasIndirectLeftRecursion (
    const Grammar & grammar) const
```

Checks if a grammar contains indirect left recursion.

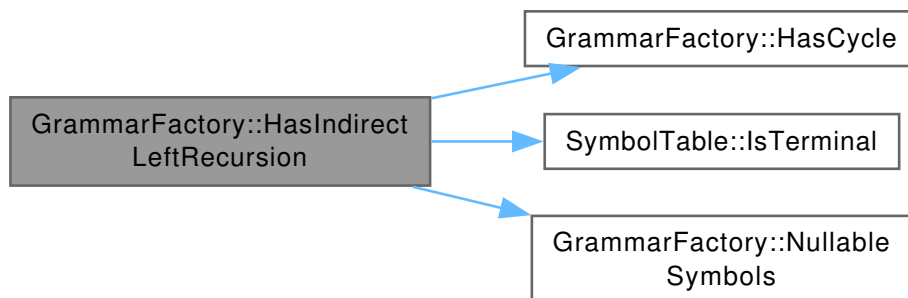
Parameters

<i>grammar</i>	The grammar to check.
----------------	-----------------------

Returns

true if there is direct left recursion, false otherwise.

Here is the call graph for this function:

**9.5.2.9 HasUnreachableSymbols()**

```
bool GrammarFactory::HasUnreachableSymbols (
    Grammar & grammar) const
```

Checks if a grammar contains unreachable symbols (non-terminals that cannot be derived from the start symbol).

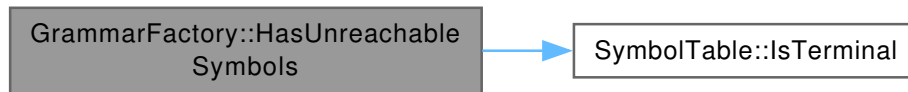
Parameters

<i>grammar</i>	The grammar to check.
----------------	-----------------------

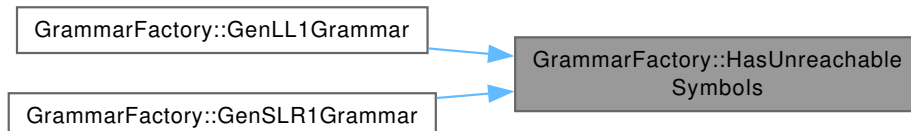
Returns

true if there are unreachable symbols, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.5.2.10 Init()**

```
void GrammarFactory::Init ()
```

Initializes the [GrammarFactory](#) and populates the items vector with initial grammar items.

9.5.2.11 IsInfinite()

```
bool GrammarFactory::IsInfinite (
    Grammar & grammar) const
```

Checks if a grammar is infinite, meaning there are non-terminal symbols that can never derive a terminal string. This happens when a production leads to an infinite recursion or an endless derivation without reaching terminal symbols. For example, a production like:

```
S -> A
A -> a A | B
B -> c B
```

could lead to an infinite derivation of non-terminals.

Parameters

<i>grammar</i>	The grammar to check.
----------------	-----------------------

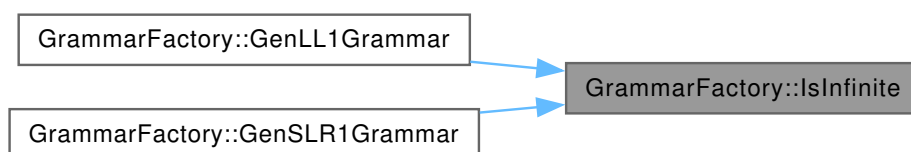
Returns

true if the grammar has infinite derivations, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



9.5.2.12 LeftFactorize()

```
void GrammarFactory::LeftFactorize (
    Grammar & grammar)
```

Performs left factorization. A grammar can be left factorized if it has productions with the same prefix for one non-terminal. For example:

```
A -> a x | a y
```

could be left factorized because it has "a" as the common prefix. The left factorization is done by adding a new non-terminal symbol that contains the uncommon part, and by unifying the common prefix in one production. So:

```
A -> a x | a y
```

would become:

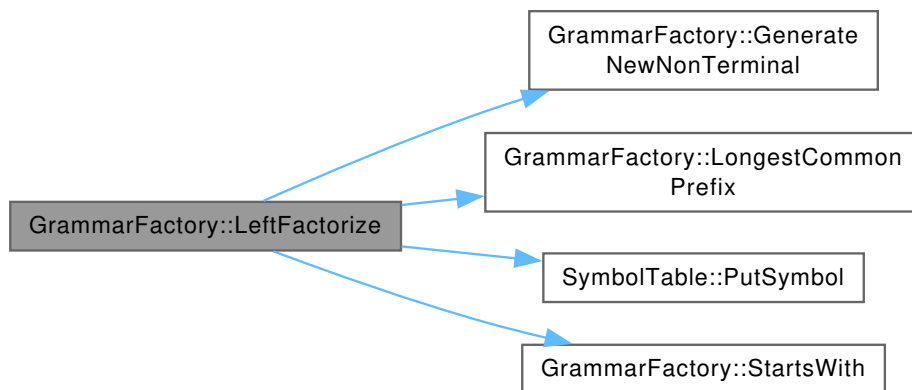
```
A -> a A'
```

```
A' -> x | y
```

Parameters

<i>grammar</i>	The grammar to be left factorized.
----------------	------------------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



9.5.2.13 LongestCommonPrefix()

```
std::vector< std::string > GrammarFactory::LongestCommonPrefix (
    const std::vector< production > & productions)
```

Finds the longest common prefix among a set of productions.

This function computes the longest sequence of symbols that is common to the beginning of all productions in the given vector. It is used during left factorization to identify common prefixes that can be factored out.

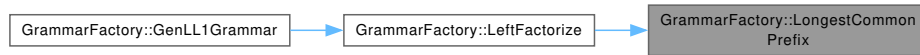
Parameters

<i>productions</i>	A vector of productions to analyze.
--------------------	-------------------------------------

Returns

A vector of strings representing the longest common prefix. If no common prefix exists, an empty vector is returned.

Here is the caller graph for this function:

**9.5.2.14 Lv1()**

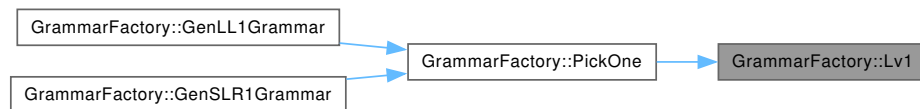
Grammar GrammarFactory::Lv1 ()

Generates a Level 1 grammar.

Returns

A Level 1 grammar.

Here is the caller graph for this function:

**9.5.2.15 Lv2()**

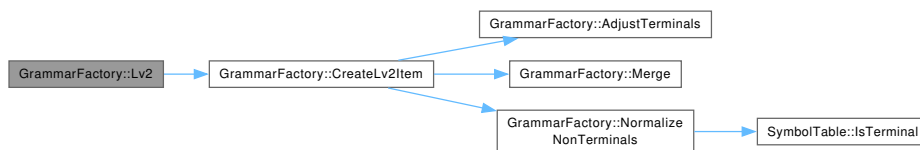
Grammar GrammarFactory::Lv2 ()

Generates a Level 2 grammar by combining Level 1 items.

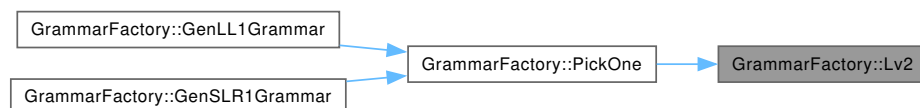
Returns

A Level 2 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.5.2.16 Lv3()**

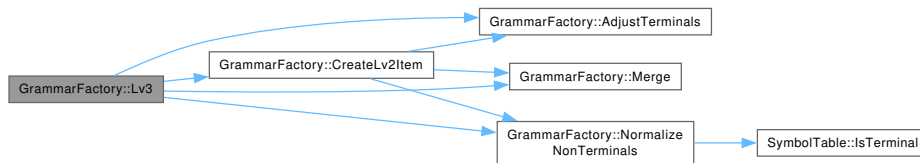
Grammar GrammarFactory::Lv3 ()

Generates a Level 3 grammar by combining a Level 2 item and a Level 1 item.

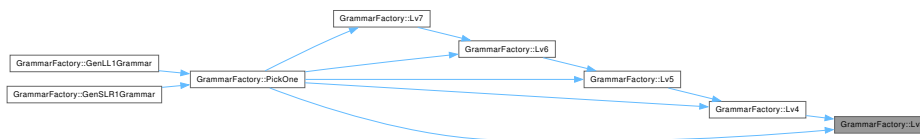
Returns

A Level 3 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.5.2.17 Lv4()**

Grammar GrammarFactory::Lv4 ()

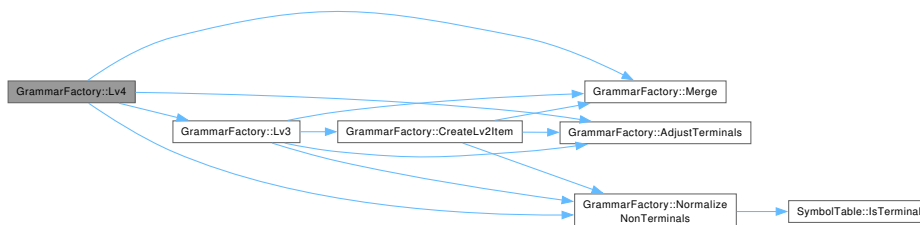
Generates a Level 4 grammar by combining Level 3 and Level 1 items.

This function creates a more complex grammar by combining elements from Level 3 and Level 1 grammars. It is used to generate grammars with increased complexity for testing or parsing purposes.

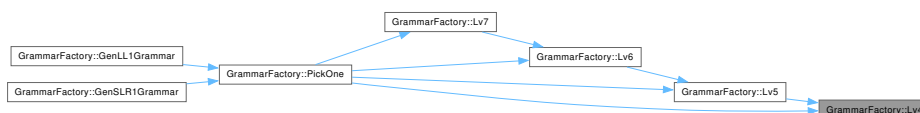
Returns

A Level 4 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.5.2.18 Lv5()**

Grammar GrammarFactory::Lv5 ()

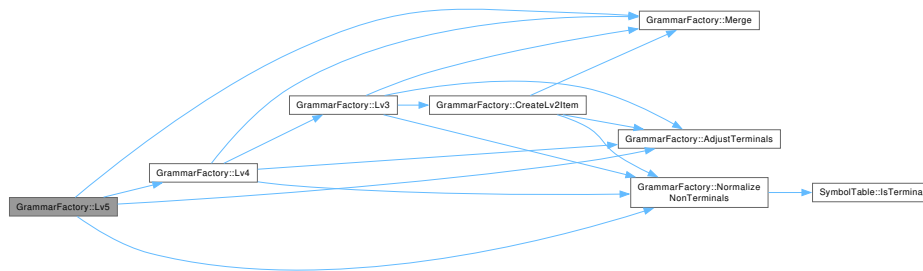
Generates a Level 5 grammar by combining Level 4 and Level 1 items.

This function creates a more advanced grammar by combining elements from Level 4 and Level 1 grammars. It is used to generate grammars with higher complexity for testing or parsing purposes.

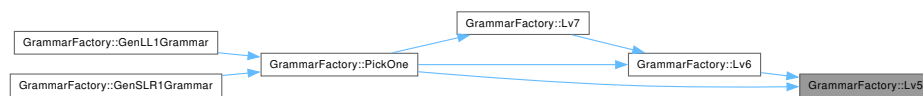
Returns

A Level 5 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.5.2.19 Lv6()**

Grammar GrammarFactory::Lv6 ()

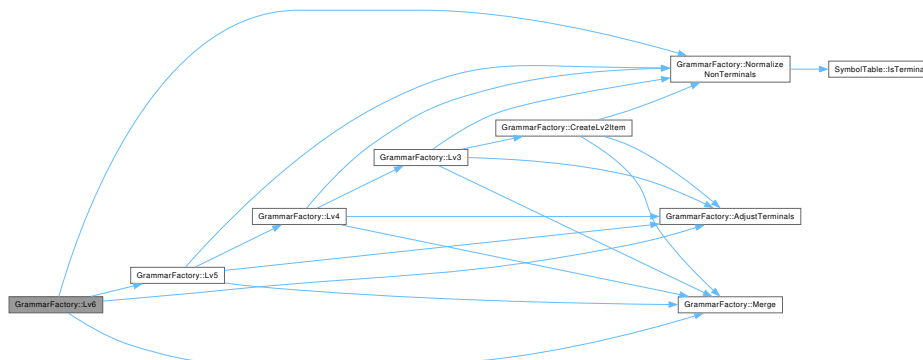
Generates a Level 6 grammar by combining Level 5 and Level 1 items.

This function creates a highly complex grammar by combining elements from Level 5 and Level 1 grammars. It is used to generate grammars with advanced structures for testing or parsing purposes.

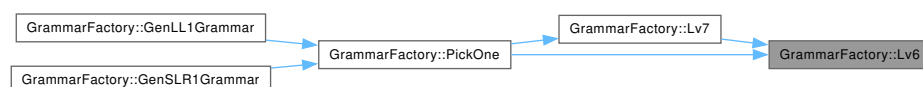
Returns

A Level 6 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



9.5.2.20 Lv7()

`Grammar GrammarFactory::Lv7 ()`

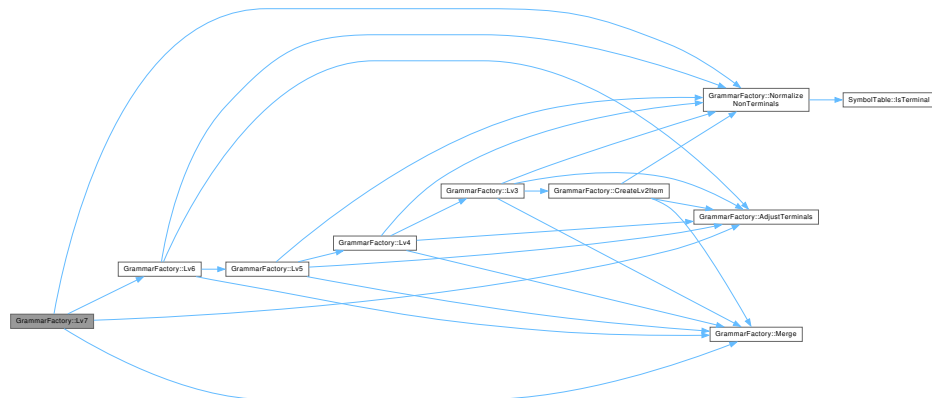
Generates a Level 7 grammar by combining Level 6 and Level 1 items.

This function creates a very complex grammar by combining elements from Level 6 and Level 1 grammars. It is used to generate grammars with highly advanced structures for testing or parsing purposes.

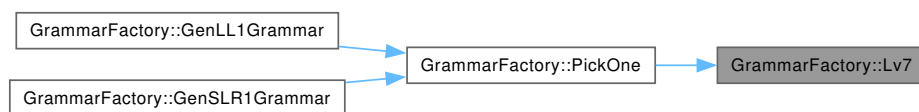
Returns

A Level 7 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



9.5.2.21 Merge()

```

std::unordered_map< std::string, std::vector< production > > GrammarFactory::Merge (
    const FactoryItem & base,
    const FactoryItem & cmb) const
  
```

Merges the grammar rules of two grammar items into a single grammar.

This function performs a raw combination of the production rules from both grammar items, resulting in a single grammar map that contains all productions.

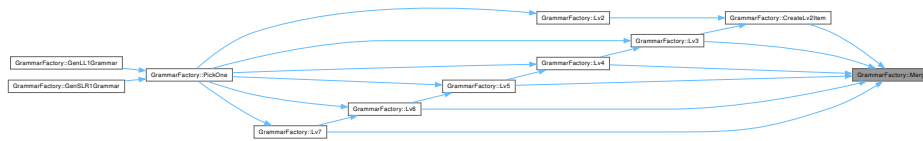
Parameters

<i>base</i>	The first grammar item.
<i>cmb</i>	The second grammar item.

Returns

A merged grammar map containing all production rules from both inputs.

Here is the caller graph for this function:



9.5.2.22 NormalizeNonTerminals()

```
void GrammarFactory::NormalizeNonTerminals (
    FactoryItem & item,
    const std::string & nt) const
```

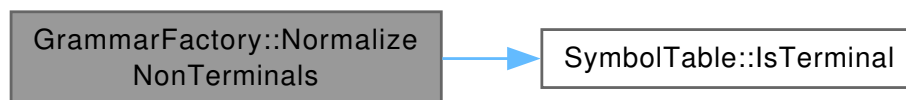
Replaces all non-terminal symbols in a grammar item with a single target non-terminal.

This function is used during grammar combination to normalize the non-terminal symbols in a given [FactoryItem](#), so that they are consistent and compatible with another item.

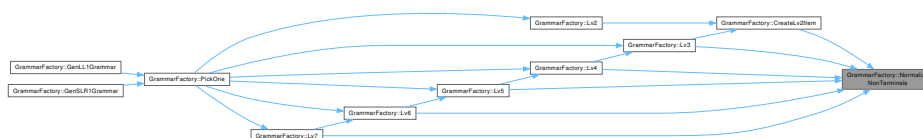
Parameters

<i>item</i>	The grammar item whose non-terminals will be renamed.
<i>nt</i>	The new non-terminal symbol that will replace all existing ones.

Here is the call graph for this function:



Here is the caller graph for this function:



9.5.2.23 NullableSymbols()

```
std::unordered_set< std::string > GrammarFactory::NullableSymbols (
    const Grammar & grammar) const
```

Find nullable symbols in a grammar.

Parameters

<i>grammar</i>	The grammar to check.
----------------	-----------------------

Returns

set of nullable symbols.

Here is the caller graph for this function:



9.5.2.24 PickOne()

```
Grammar GrammarFactory::PickOne (
    int level)
```

Picks a random grammar based on the specified difficulty level (1, 2, or 3).

Parameters

<i>level</i>	The difficulty level (1, 2, or 3).
--------------	------------------------------------

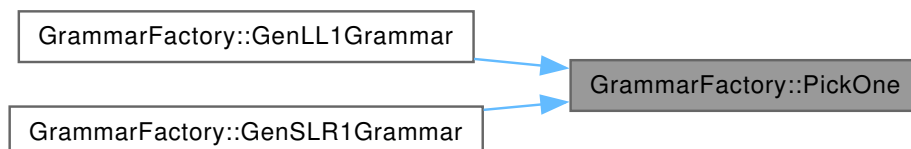
Returns

A randomly picked grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



9.5.2.25 RemoveLeftRecursion()

```
void GrammarFactory::RemoveLeftRecursion (
    Grammar & grammar)
```

Removes direct left recursion in a grammar. A grammar has direct left recursion when one of its productions is.

$$A \rightarrow A \ a$$

where A is a non-terminal symbol and "a" the rest of the production. The procedure removes direct left recursion by adding a new non-terminal. So, if the productions with left recursion are:

```
A -> A a | b
```

the result would be:

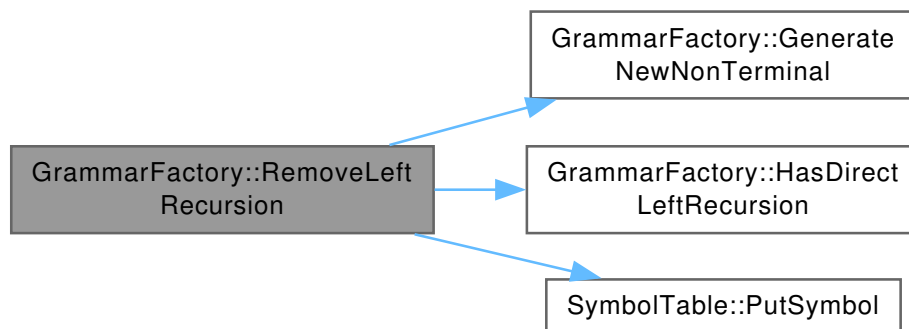
```
A -> b A'
```

```
A' -> a A' | epsilon
```

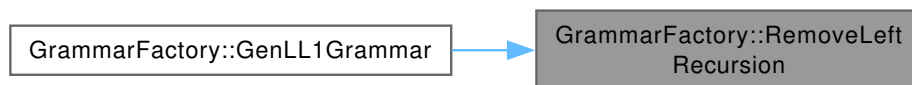
Parameters

<i>grammar</i>	The grammar to remove left recursion.
----------------	---------------------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



9.5.2.26 StartsWith()

```
bool GrammarFactory::StartsWith (
    const production & prod,
    const std::vector< std::string > & prefix)
```

Checks if a production starts with a given prefix.

This function determines whether the symbols in a production match the provided prefix sequence at the beginning. It is used during left factorization to identify productions that share a common prefix.

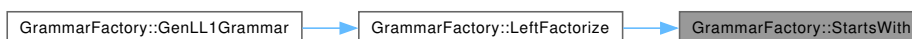
Parameters

<i>prod</i>	The production to check.
<i>prefix</i>	The sequence of symbols to compare against the beginning of the production.

Returns

`true` if the production starts with the prefix, `false` otherwise.

Here is the caller graph for this function:



9.5.3 Member Data Documentation

9.5.3.1 items

```
std::vector<FactoryItem> GrammarFactory::items
```

A vector of [FactoryItem](#) objects representing different level 1 grammar items created by the Init method.

9.5.3.2 non_terminal_alphabet_

```
std::vector<std::string> GrammarFactory::non_terminal_alphabet_
```

Initial value:

```
{"A", "B", "C", "D",  
                                "E", "F", "G"}
```

A vector of non-terminal symbols (alphabet) used in the grammar.

9.5.3.3 terminal_alphabet_

```
std::vector<std::string> GrammarFactory::terminal_alphabet_
```

Initial value:

```
{"a", "b", "c", "d", "e", "f",  
                                "g", "h", "i", "j", "k", "l"}
```

A vector of terminal symbols (alphabet) used in the grammar.

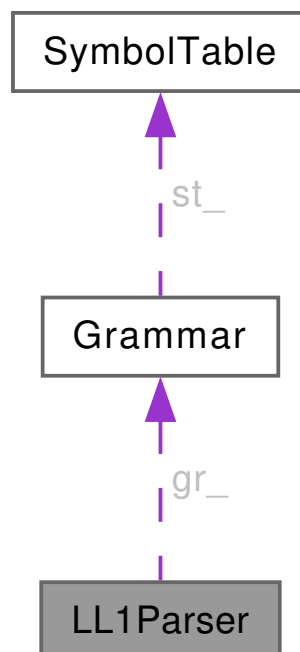
The documentation for this struct was generated from the following files:

- [backend/grammar_factory.hpp](#)
- [backend/grammar_factory.cpp](#)

9.6 LL1Parser Class Reference

```
#include <ll1_parser.hpp>
```

Collaboration diagram for LL1Parser:



Public Member Functions

- [LL1Parser](#) ()=default
- [LL1Parser](#) ([Grammar](#) gr)

- Constructs an [LL1Parser](#) with a grammar object and an input file.
- bool [CreateLL1Table](#) ()
Creates the LL(1) parsing table for the grammar.
- void [First](#) (std::span< const std::string > rule, std::unordered_set< std::string > &result)
Calculates the FIRST set for a given production rule in a grammar.
- void [ComputeFirstSets](#) ()
Computes the FIRST sets for all non-terminal symbols in the grammar.
- void [ComputeFollowSets](#) ()
Computes the FOLLOW sets for all non-terminal symbols in the grammar. The FOLLOW set of a non-terminal symbol A contains all terminal symbols that can appear immediately after A in any sentential form derived from the grammar's start symbol. Additionally, if A can be the last symbol in a derivation, the end-of-input marker ($\$$) is included in its FOLLOW set. This function computes the FOLLOW sets using the following rules:
- std::unordered_set< std::string > [Follow](#) (const std::string &arg)
Computes the FOLLOW set for a given non-terminal symbol in the grammar.
- std::unordered_set< std::string > [PredictionSymbols](#) (const std::string &antecedent, const std::vector< std::string > &consequent)
Computes the prediction symbols for a given production rule.

Public Attributes

- ll1_table [ll1_t](#)
The LL(1) parsing table, mapping non-terminals and terminals to productions.
- Grammar [gr_](#)
[Grammar](#) object associated with this parser.
- std::unordered_map< std::string, std::unordered_set< std::string > > [first_sets_](#)
FIRST sets for each non-terminal in the grammar.
- std::unordered_map< std::string, std::unordered_set< std::string > > [follow_sets_](#)
FOLLOW sets for each non-terminal in the grammar.

9.6.1 Constructor & Destructor Documentation

9.6.1.1 LL1Parser() [1/2]

```
LL1Parser::LL1Parser () [default]
```

9.6.1.2 LL1Parser() [2/2]

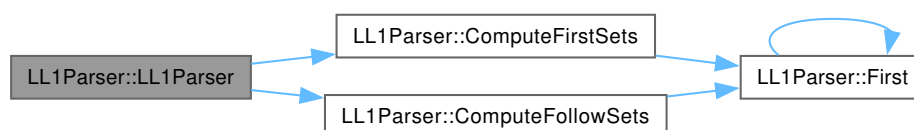
```
LL1Parser::LL1Parser (
    Grammar gr) [explicit]
```

Constructs an [LL1Parser](#) with a grammar object and an input file.

Parameters

<i>gr</i>	Grammar object to parse with
-----------	--

Here is the call graph for this function:



9.6.2 Member Function Documentation

9.6.2.1 ComputeFirstSets()

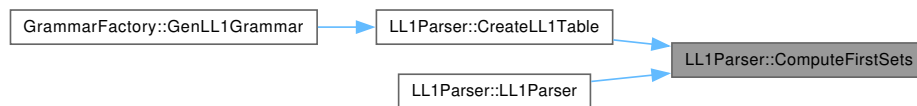
```
void LL1Parser::ComputeFirstSets ()
```

Computes the FIRST sets for all non-terminal symbols in the grammar.

This function calculates the FIRST set for each non-terminal symbol in the grammar by iteratively applying a least fixed-point algorithm. This approach ensures that the FIRST sets are fully populated by repeatedly expanding and updating the sets until no further changes occur (i.e., a fixed-point is reached). Here is the call graph for this function:



Here is the caller graph for this function:



9.6.2.2 ComputeFollowSets()

```
void LL1Parser::ComputeFollowSets ()
```

Computes the FOLLOW sets for all non-terminal symbols in the grammar. The FOLLOW set of a non-terminal symbol A contains all terminal symbols that can appear immediately after A in any sentential form derived from the grammar's start symbol. Additionally, if A can be the last symbol in a derivation, the end-of-input marker ($\$$) is included in its FOLLOW set. This function computes the FOLLOW sets using the following rules:

1. Initialize $FOLLOW(S) = \{ \$ \}$, where S is the start symbol.
2. For each production rule of the form $A \rightarrow \alpha B \beta$:
 - Add $FIRST(\beta) \setminus \{ \epsilon \}$ to $FOLLOW(B)$.
 - If $\epsilon \in FIRST(\beta)$, add $FOLLOW(A)$ to $FOLLOW(B)$.
3. Repeat step 2 until no changes occur in any FOLLOW set. The computed FOLLOW sets are cached in the `follow_sets_` member variable for later use by the parser.

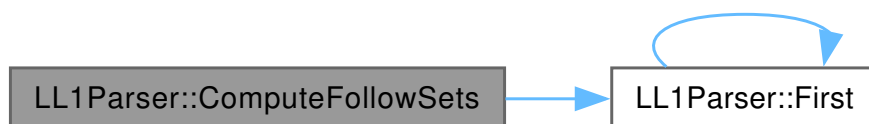
Note

This function assumes that the FIRST sets for all symbols have already been computed and are available in the `first_sets_` member variable.

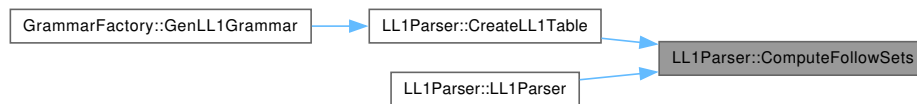
See also

[First](#)
[follow_sets_](#)

Here is the call graph for this function:



Here is the caller graph for this function:



9.6.2.3 CreateLL1Table()

```
bool LL1Parser::CreateLL1Table ()
```

Creates the LL(1) parsing table for the grammar.

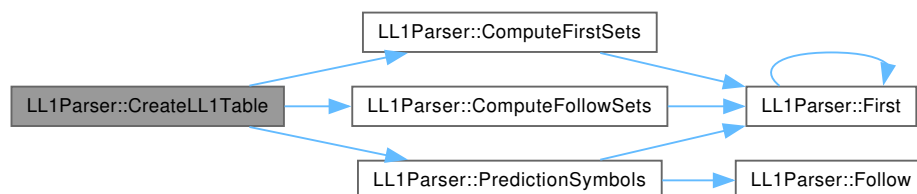
This function constructs the LL(1) parsing table by iterating over each production in the grammar and determining the appropriate cells for each non-terminal and director symbol (prediction symbol) combination. If the grammar is LL(1) compatible, each cell will contain at most one production, indicating no conflicts. If conflicts are found, the function will return `false`, signaling that the grammar is not LL(1).

- For each production rule $A \rightarrow \alpha$, the function calculates the prediction symbols using the `PredictionSymbols` function.
- It then fills the parsing table at the cell corresponding to the non-terminal `A` and each prediction symbol in the set.
- If a cell already contains a production, this indicates a conflict, meaning the grammar is not LL(1).

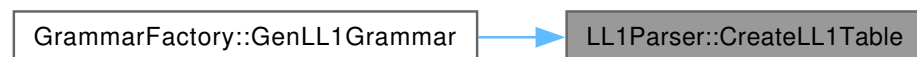
Returns

`true` if the table is created successfully, indicating the grammar is LL(1) compatible; `false` if any conflicts are detected, showing that the grammar does not meet LL(1) requirements.

Here is the call graph for this function:



Here is the caller graph for this function:



9.6.2.4 First()

```
void LL1Parser::First (
    std::span< const std::string > rule,
    std::unordered_set< std::string > & result)
```

Calculates the FIRST set for a given production rule in a grammar.

The FIRST set of a production rule contains all terminal symbols that can appear at the beginning of any string derived from that rule. If the rule can derive the empty string (epsilon), epsilon is included in the FIRST set.

This function computes the FIRST set by examining each symbol in the production rule:

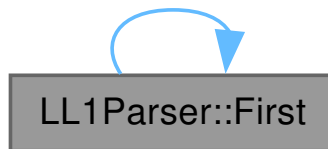
- If a terminal symbol is encountered, it is added directly to the FIRST set, as it is the starting symbol of some derivation.

- If a non-terminal symbol is encountered, its FIRST set is recursively computed and added to the result, excluding epsilon unless it is followed by another symbol that could also lead to epsilon.
- If the entire rule could derive epsilon (i.e., each symbol in the rule can derive epsilon), then epsilon is added to the FIRST set.

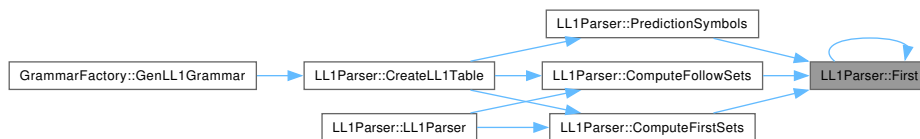
Parameters

<i>rule</i>	A span of strings representing the production rule for which to compute the FIRST set. Each string in the span is a symbol (either terminal or non-terminal).
<i>result</i>	A reference to an unordered set of strings where the computed FIRST set will be stored. The set will contain all terminal symbols that can start derivations of the rule, and possibly epsilon if the rule can derive an empty string.

Here is the call graph for this function:



Here is the caller graph for this function:



9.6.2.5 Follow()

```
std::unordered_set< std::string > LL1Parser::Follow (
    const std::string & arg)
```

Computes the FOLLOW set for a given non-terminal symbol in the grammar.

The FOLLOW set for a non-terminal symbol includes all symbols that can appear immediately to the right of that symbol in any derivation, as well as any end-of-input markers if the symbol can appear at the end of derivations. FOLLOW sets are used in LL(1) parsing table construction to determine possible continuations after a non-terminal.

Parameters

<i>arg</i>	Non-terminal symbol for which to compute the FOLLOW set.
------------	--

Returns

An unordered set of strings containing symbols that form the FOLLOW set for *arg*.

Here is the caller graph for this function:



9.6.2.6 PredictionSymbols()

```
std::unordered_set< std::string > LL1Parser::PredictionSymbols (
    const std::string & antecedent,
    const std::vector< std::string > & consequent)
```

Computes the prediction symbols for a given production rule.

- The prediction symbols for a rule determine the set of input symbols that can trigger this rule in the parsing table. This function calculates the prediction symbols based on the FIRST set of the consequent and, if epsilon (the empty symbol) is in the FIRST set, also includes the FOLLOW set of the antecedent.
- - If the FIRST set of the consequent does not contain epsilon, the prediction symbols are simply the FIRST symbols of the consequent.

If the FIRST set of the consequent contains epsilon, the prediction symbols are computed as $FIRST(consequent) \setminus \{\epsilon\} \cup FOLLOW(antecedent)$.

•

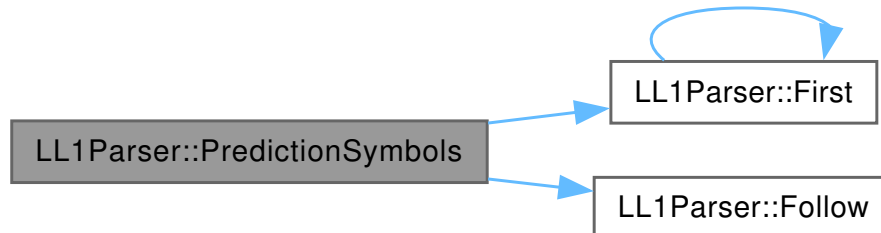
Parameters

<i>antecedent</i>	The left-hand side non-terminal symbol of the rule.
<i>consequent</i>	A vector of symbols on the right-hand side of the rule (production body).

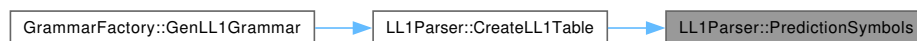
Returns

An unordered set of strings containing the prediction symbols for the specified rule.

Here is the call graph for this function:



Here is the caller graph for this function:



9.6.3 Member Data Documentation

9.6.3.1 first_sets_

```
std::unordered_map<std::string, std::unordered_set<std::string> > LL1Parser::first_sets_
```

FIRST sets for each non-terminal in the grammar.

9.6.3.2 follow_sets_

```
std::unordered_map<std::string, std::unordered_set<std::string> > LL1Parser::follow_sets_
```

FOLLOW sets for each non-terminal in the grammar.

9.6.3.3 gr_

`Grammar` `LL1Parser::gr_`
`Grammar` object associated with this parser.

9.6.3.4 ll1_t_

`ll1_table` `LL1Parser::ll1_t_`
 The LL(1) parsing table, mapping non-terminals and terminals to productions.
 The documentation for this class was generated from the following files:

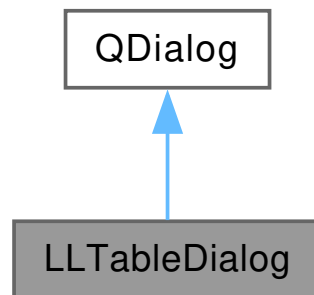
- `backend/ll1_parser.hpp`
- `backend/ll1_parser.cpp`

9.7 LLTableDialog Class Reference

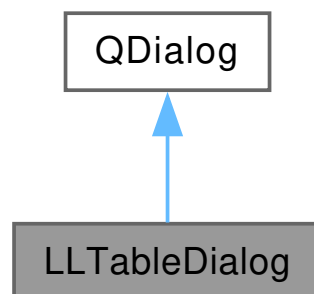
Dialog for filling and submitting an LL(1) parsing table.

```
#include <lltabledialog.h>
```

Inheritance diagram for LLTableDialog:



Collaboration diagram for LLTableDialog:



Signals

- void `submitted` (const `QVector< QVector< QString > >` &data)
Signal emitted when the user submits the table.

Public Member Functions

- `LLTableDialog` (const `QStringList` &rowHeaders, const `QStringList` &colHeaders, `QWidget` *parent, `QVector< QVector< QString > >` *initialData=nullptr)
Constructs the LL(1) table dialog with given headers and optional initial data.
- `QVector< QVector< QString > >` `getTableData` () const
Returns the contents of the table filled by the user.

- void `setInitialData` (const QVector< QVector< QString > > &data)
Pre-fills the table with existing user data.
- void `highlightIncorrectCells` (const QList< QPair< int, int > > &coords)
Highlights cells that are incorrect based on provided coordinates.

9.7.1 Detailed Description

Dialog for filling and submitting an LL(1) parsing table.

This class represents a dialog window that displays a table for users to complete the LL(1) parsing matrix. It provides functionality to initialize the table with data, retrieve the user's input, and highlight incorrect answers.

9.7.2 Constructor & Destructor Documentation

9.7.2.1 LLTableDialog()

```
LLTableDialog::LLTableDialog (
    const QStringList & rowHeaders,
    const QStringList & colHeaders,
    QWidget * parent,
    QVector< QVector< QString > > * initialData = nullptr)
```

Constructs the LL(1) table dialog with given headers and optional initial data.

Parameters

<i>rowHeaders</i>	Row labels (non-terminal symbols).
<i>colHeaders</i>	Column labels (terminal symbols).
<i>parent</i>	Parent widget.
<i>initialData</i>	Optional initial table data to pre-fill cells.

9.7.3 Member Function Documentation

9.7.3.1 getTableData()

```
QVector< QVector< QString > > LLTableDialog::getTableData () const
```

Returns the contents of the table filled by the user.

Returns

A 2D vector representing the LL(1) table.

9.7.3.2 highlightIncorrectCells()

```
void LLTableDialog::highlightIncorrectCells (
    const QList< QPair< int, int > > & coords)
```

Highlights cells that are incorrect based on provided coordinates.

Parameters

<i>coords</i>	A list of (row, column) pairs to highlight as incorrect.
---------------	--

9.7.3.3 setInitialData()

```
void LLTableDialog::setInitialData (
    const QVector< QVector< QString > > & data)
```


Pre-fills the table with existing user data.

This is used to populate the table with a previous (possibly incorrect) answer when retrying a task or providing feedback.

Parameters

<i>data</i>	A 2D vector of strings representing the initial cell values.
-------------	--

9.7.3.4 submitted

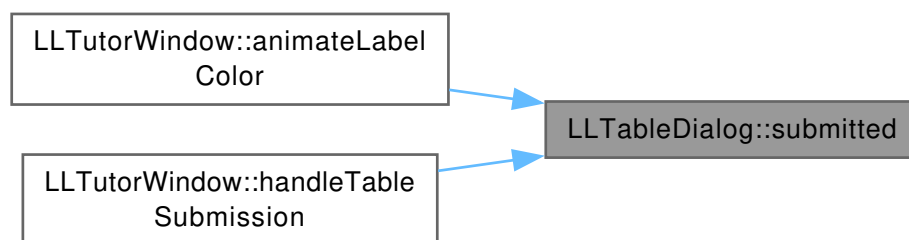
```
void LLTableDialog::submitted (
    const QVector< QVector< QString > > & data) [signal]
```

Signal emitted when the user submits the table.

Parameters

<i>data</i>	The filled table data submitted by the user.
-------------	--

Here is the caller graph for this function:



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

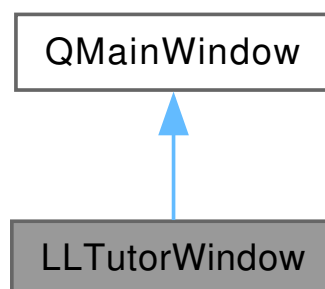
- [gui/lltabledialog.h](#)
- [gui/lltabledialog.cpp](#)

9.8 LLTutorWindow Class Reference

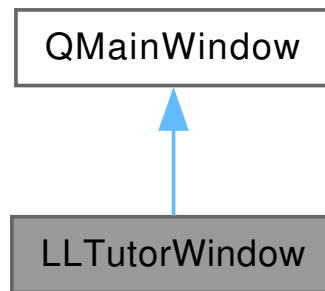
Main window for the LL(1) interactive tutoring mode in [SyntaxTutor](#).

```
#include <lltutorwindow.h>
```

Inheritance diagram for LLTutorWindow:



Collaboration diagram for LLTutorWindow:



Classes

- struct [TreeNode](#)
[TreeNode](#) structure used to build derivation trees.

Signals

- void [sessionFinished](#) (int cntRight, int cntWrong)

Public Member Functions

- [LLTutorWindow](#) (const [Grammar](#) &grammar, [TutorialManager](#) *tm=nullptr, QWidget *parent=nullptr)
Constructs the LL(1) tutor window with a given grammar.
- [~LLTutorWindow](#) ()
- QString [generateQuestion](#) ()
Generates a question for the current state of the tutor.
- void [updateState](#) (bool isCorrect)
Updates the tutor state after verifying user response.
- QString [FormatGrammar](#) (const [Grammar](#) &grammar)
Formats a grammar for display in the chat interface.
- void [addMessage](#) (const QString &text, bool isUser)
- void [addWidgetMessage](#) (QWidget *widget)
< Add text message to chat
- void [exportConversationToPdf](#) (const QString &filePath)
< Add widget (e.g., table, tree)
- void [showTable](#) ()
< Export chat to PDF
- void [showTableForCPrime](#) ()
Display the full LL(1) table in C' ex.
- void [updateProgressPanel](#) ()
- void [animateLabelPop](#) (QLabel *label)
- void [animateLabelColor](#) (QLabel *label, const QColor &flashColor)
- void [wrongAnimation](#) ()
Visual shake/flash for incorrect answer.
- void [wrongUserResponseAnimation](#) ()
Animation specific to user chat input.
- void [markLastUserIncorrect](#) ()
Marks last message as incorrect.
- void [TeachFirstTree](#) (const std::vector< std::string > &symbols, std::unordered_set< std::string > &first_set, int depth, std::unordered_set< std::string > &processing, QTreeWidgetItem *parent)

- `std::unique_ptr< TreeNode > buildTreeNode` (const `std::vector< std::string > &symbols`, `std::unordered_set< std::string > &first_set`, `int depth`, `std::vector< std::pair< std::string, std::vector< std::string > > > &active_derivations`)
- `int computeSubtreeWidth` (const `std::unique_ptr< TreeNode > &node`, `int hSpacing`)
- `void drawTree` (const `std::unique_ptr< TreeNode > &root`, `QGraphicsScene *scene`, `QPointF pos`, `int hSpacing`, `int vSpacing`)
- `void showTreeGraphics` (`std::unique_ptr< TreeNode > root`)
- `bool verifyResponse` (const `QString &userResponse`)
- `bool verifyResponseForA` (const `QString &userResponse`)
- `bool verifyResponseForA1` (const `QString &userResponse`)
- `bool verifyResponseForA2` (const `QString &userResponse`)
- `bool verifyResponseForB` (const `QString &userResponse`)
- `bool verifyResponseForB1` (const `QString &userResponse`)
- `bool verifyResponseForB2` (const `QString &userResponse`)
- `bool verifyResponseForC` ()
- `QString solution` (const `std::string &state`)
- `QStringList solutionForA` ()
- `QString solutionForA1` ()
- `QString solutionForA2` ()
- `QSet< QString > solutionForB` ()
- `QSet< QString > solutionForB1` ()
- `QSet< QString > solutionForB2` ()
- `QString feedback` ()
- `QString feedbackForA` ()
- `QString feedbackForA1` ()
- `QString feedbackForA2` ()
- `QString feedbackForAPrime` ()
- `QString feedbackForB` ()
- `QString feedbackForB1` ()
- `QString feedbackForB2` ()
- `QString feedbackForBPrime` ()
- `QString feedbackForC` ()
- `QString feedbackForCPrime` ()
- `void feedbackForB1TreeWidget` ()
- `void feedbackForB1TreeGraphics` ()
- `QString TeachFollow` (const `QString &nt`)
- `QString TeachPredictionSymbols` (const `QString &ant`, const `production &conseq`)
- `QString TeachLL1Table` ()
- `void handleTableSubmission` (const `QVector< QVector< QString > > &raw`, const `QStringList &colHeaders`)

Protected Member Functions

- `void closeEvent` (`QCloseEvent *event`) override
- `bool eventFilter` (`QObject *obj`, `QEvent *event`) override

9.8.1 Detailed Description

Main window for the LL(1) interactive tutoring mode in [SyntaxTutor](#).

This class guides students through the construction and analysis of LL(1) parsing tables. It uses a finite-state sequence to present progressively more complex tasks, verifies user responses, provides corrective feedback, and supports visualizations like derivation trees.

The tutor is designed to teach the student *how* the LL(1) table is built, not just test it — including interactive tasks, animated feedback, and hints.

Key features include:

- Interactive question flow based on grammar analysis.

- Derivation tree generation (TeachFirst).
- Step-by-step verification of FIRST, FOLLOW, prediction symbols, and table entries.
- Exportable conversation log for grading or review.

9.8.2 Constructor & Destructor Documentation

9.8.2.1 LLTutorWindow()

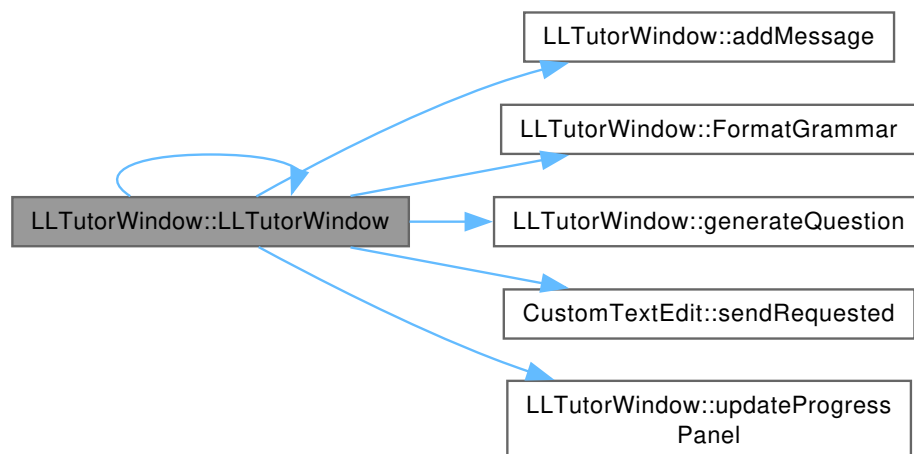
```
LLTutorWindow::LLTutorWindow (
    const Grammar & grammar,
    TutorialManager * tm = nullptr,
    QWidget * parent = nullptr) [explicit]
```

Constructs the LL(1) tutor window with a given grammar.

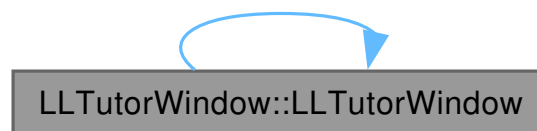
Parameters

<i>grammar</i>	The grammar to use during the session.
<i>tm</i>	Optional pointer to the tutorial manager (for help overlays).
<i>parent</i>	Parent widget.

Here is the call graph for this function:



Here is the caller graph for this function:



9.8.2.2 ~LLTutorWindow()

```
LLTutorWindow::~~LLTutorWindow ()
```

9.8.3 Member Function Documentation

9.8.3.1 addMessage()

```
void LLTutorWindow::addMessage (
    const QString & text,
    bool isUser)
```

Here is the caller graph for this function:



9.8.3.2 addWidgetMessage()

```
void LLTutorWindow::addWidgetMessage (
    QWidget * widget)
```

< Add text message to chat

9.8.3.3 animateLabelColor()

```
void LLTutorWindow::animateLabelColor (
    QLabel * label,
    const QColor & flashColor)
```

Here is the call graph for this function:



9.8.3.4 animateLabelPop()

```
void LLTutorWindow::animateLabelPop (
    QLabel * label)
```

9.8.3.5 buildTreeNode()

```
std::unique_ptr< LLTutorWindow::TreeNode > LLTutorWindow::buildTreeNode (
    const std::vector< std::string > & symbols,
    std::unordered_set< std::string > & first_set,
    int depth,
    std::vector< std::pair< std::string, std::vector< std::string > > > & active_↵
derivations)
```

9.8.3.6 closeEvent()

```
void LLTutorWindow::closeEvent (
    QCloseEvent * event) [inline], [override], [protected]
```

Here is the call graph for this function:



9.8.3.7 computeSubtreeWidth()

```
int LLTutorWindow::computeSubtreeWidth (
    const std::unique_ptr< TreeNode > & node,
    int hSpacing)
```

9.8.3.8 drawTree()

```
void LLTutorWindow::drawTree (
    const std::unique_ptr< TreeNode > & root,
    QGraphicsScene * scene,
    QPointF pos,
    int hSpacing,
    int vSpacing)
```

9.8.3.9 eventFilter()

```
bool LLTutorWindow::eventFilter (
    QObject * obj,
    QEvent * event) [override], [protected]
```

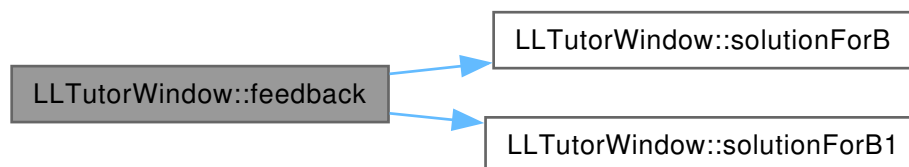
9.8.3.10 exportConversationToPdf()

```
void LLTutorWindow::exportConversationToPdf (
    const QString & filePath)
< Add widget (e.g., table, tree)
```

9.8.3.11 feedback()

```
QString LLTutorWindow::feedback ()
```

Here is the call graph for this function:

**9.8.3.12 feedbackForA()**

```
QString LLTutorWindow::feedbackForA ()
```

9.8.3.13 feedbackForA1()

```
QString LLTutorWindow::feedbackForA1 ()
```

9.8.3.14 feedbackForA2()

```
QString LLTutorWindow::feedbackForA2 ()
```

9.8.3.15 feedbackForAPrime()

```
QString LLTutorWindow::feedbackForAPrime ()
```

9.8.3.16 feedbackForB()

```
QString LLTutorWindow::feedbackForB ()
```

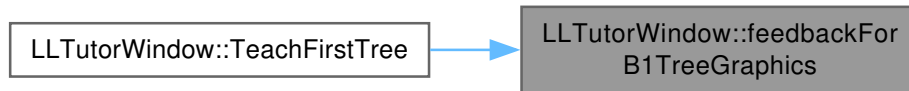
9.8.3.17 feedbackForB1()

```
QString LLTutorWindow::feedbackForB1 ()
```

9.8.3.18 feedbackForB1TreeGraphics()

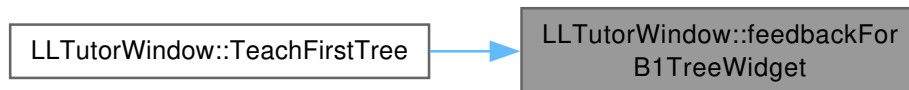
```
void LLTutorWindow::feedbackForB1TreeGraphics ()
```

Here is the caller graph for this function:

**9.8.3.19 feedbackForB1TreeWidget()**

```
void LLTutorWindow::feedbackForB1TreeWidget ()
```

Here is the caller graph for this function:

**9.8.3.20 feedbackForB2()**

```
QString LLTutorWindow::feedbackForB2 ()
```

9.8.3.21 feedbackForBPrime()

```
QString LLTutorWindow::feedbackForBPrime ()
```

9.8.3.22 feedbackForC()

```
QString LLTutorWindow::feedbackForC ()
```

9.8.3.23 feedbackForCPrime()

```
QString LLTutorWindow::feedbackForCPrime ()
```

9.8.3.24 FormatGrammar()

```
QString LLTutorWindow::FormatGrammar (
    const Grammar & grammar)
```

Formats a grammar for display in the chat interface.

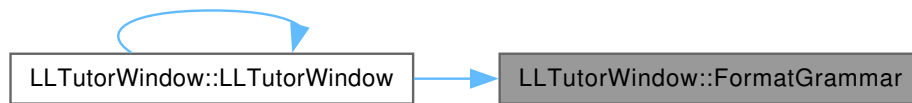
Parameters

<i>grammar</i>	The grammar to format.
----------------	------------------------

Returns

A QString representation.

Here is the caller graph for this function:

**9.8.3.25 generateQuestion()**

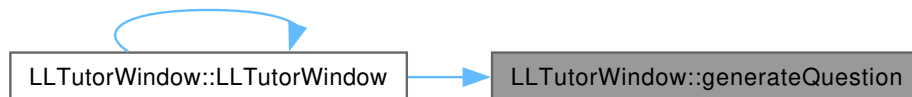
```
QString LLTutorWindow::generateQuestion ()
```

Generates a question for the current state of the tutor.

Returns

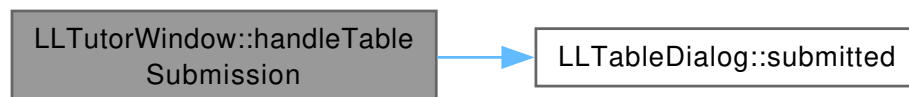
A formatted question string.

Here is the caller graph for this function:

**9.8.3.26 handleTableSubmission()**

```
void LLTutorWindow::handleTableSubmission (
    const QVector< QVector< QString > > & raw,
    const QStringList & colHeaders)
```

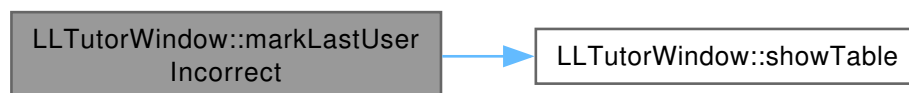
Here is the call graph for this function:

**9.8.3.27 markLastUserIncorrect()**

```
void LLTutorWindow::markLastUserIncorrect ()
```

Marks last message as incorrect.

Here is the call graph for this function:

**9.8.3.28 sessionFinished**

```
void LLTutorWindow::sessionFinished (
    int cntRight,
    int cntWrong) [signal]
```


Here is the caller graph for this function:

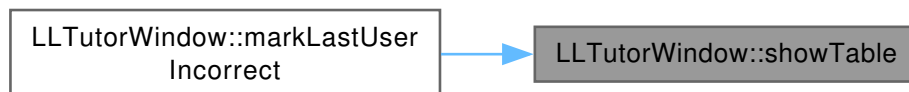


9.8.3.29 showTable()

```
void LLTutorWindow::showTable ()
```

< Export chat to PDF

Display the full LL(1) table in C ex. Here is the caller graph for this function:



9.8.3.30 showTableForCPrime()

```
void LLTutorWindow::showTableForCPrime ()
```

Display the full LL(1) table in C' ex.

9.8.3.31 showTreeGraphics()

```
void LLTutorWindow::showTreeGraphics (
    std::unique_ptr< TreeNode > root)
```

9.8.3.32 solution()

```
QString LLTutorWindow::solution (
    const std::string & state)
```

9.8.3.33 solutionForA()

```
QStringList LLTutorWindow::solutionForA ()
```

9.8.3.34 solutionForA1()

```
QString LLTutorWindow::solutionForA1 ()
```

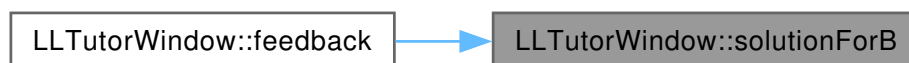
9.8.3.35 solutionForA2()

```
QString LLTutorWindow::solutionForA2 ()
```

9.8.3.36 solutionForB()

```
QSet< QString > LLTutorWindow::solutionForB ()
```

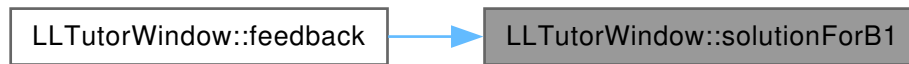
Here is the caller graph for this function:



9.8.3.37 solutionForB1()

```
QSet< QString > LLTutorWindow::solutionForB1 ()
```

Here is the caller graph for this function:



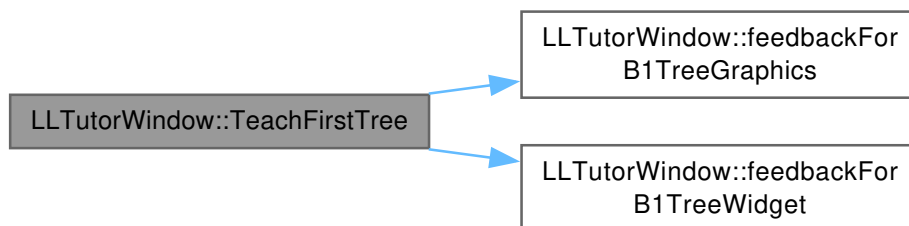
9.8.3.38 solutionForB2()

```
QSet< QString > LLTutorWindow::solutionForB2 ()
```

9.8.3.39 TeachFirstTree()

```
void LLTutorWindow::TeachFirstTree (
    const std::vector< std::string > & symbols,
    std::unordered_set< std::string > & first_set,
    int depth,
    std::unordered_set< std::string > & processing,
    QTreeWidgetItem * parent)
```

Here is the call graph for this function:



9.8.3.40 TeachFollow()

```
QString LLTutorWindow::TeachFollow (
    const QString & nt)
```

9.8.3.41 TeachLL1Table()

```
QString LLTutorWindow::TeachLL1Table ()
```

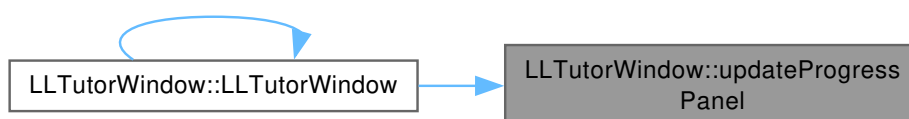
9.8.3.42 TeachPredictionSymbols()

```
QString LLTutorWindow::TeachPredictionSymbols (
    const QString & ant,
    const production & conseq)
```

9.8.3.43 updateProgressPanel()

```
void LLTutorWindow::updateProgressPanel ()
```

Here is the caller graph for this function:



9.8.3.44 updateState()

```
void LLTutorWindow::updateState (  
    bool isCorrect)
```

Updates the tutor state after verifying user response.

Parameters

<i>isCorrect</i>	Whether the user answered correctly.
------------------	--------------------------------------

9.8.3.45 verifyResponse()

```
bool LLTutorWindow::verifyResponse (  
    const QString & userResponse)
```

9.8.3.46 verifyResponseForA()

```
bool LLTutorWindow::verifyResponseForA (  
    const QString & userResponse)
```

9.8.3.47 verifyResponseForA1()

```
bool LLTutorWindow::verifyResponseForA1 (  
    const QString & userResponse)
```

9.8.3.48 verifyResponseForA2()

```
bool LLTutorWindow::verifyResponseForA2 (  
    const QString & userResponse)
```

9.8.3.49 verifyResponseForB()

```
bool LLTutorWindow::verifyResponseForB (  
    const QString & userResponse)
```

9.8.3.50 verifyResponseForB1()

```
bool LLTutorWindow::verifyResponseForB1 (  
    const QString & userResponse)
```

9.8.3.51 verifyResponseForB2()

```
bool LLTutorWindow::verifyResponseForB2 (  
    const QString & userResponse)
```

9.8.3.52 verifyResponseForC()

```
bool LLTutorWindow::verifyResponseForC ()
```

9.8.3.53 wrongAnimation()

```
void LLTutorWindow::wrongAnimation ()
```

Visual shake/flash for incorrect answer.

9.8.3.54 wrongUserResponseAnimation()

void LLTutorWindow::wrongUserResponseAnimation ()

Animation specific to user chat input.

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

- [gui/lltutorwindow.h](#)
- [gui/lltutorwindow.cpp](#)

9.9 Lr0Item Struct Reference

Represents an LR(0) item used in LR automata construction.

```
#include <lr0_item.hpp>
```

Public Member Functions

- [Lr0Item](#) (std::string antecedent, std::vector< std::string > consequent, std::string epsilon, std::string eol)
Constructs an LR(0) item with the dot at position 0.
- [Lr0Item](#) (std::string antecedent, std::vector< std::string > consequent, unsigned int dot, std::string epsilon, std::string eol)
Constructs an LR(0) item with a custom dot position.
- std::string [NextToDot](#) () const
Returns the symbol immediately after the dot, or empty if the dot is at the end.
- void [PrintItem](#) () const
Prints the LR(0) item to the standard output in a human-readable format.
- std::string [ToString](#) () const
Converts the item to a string representation, including the dot position.
- void [AdvanceDot](#) ()
Advances the dot one position to the right.
- bool [IsComplete](#) () const
Checks whether the dot has reached the end of the production.
- bool [operator==](#) (const [Lr0Item](#) &other) const
Equality operator for comparing two LR(0) items.

Public Attributes

- std::string [antecedent_](#)
The non-terminal on the left-hand side of the production.
- std::vector< std::string > [consequent_](#)
The sequence of symbols on the right-hand side of the production.
- std::string [epsilon_](#)
The symbol representing the empty string (ϵ).
- std::string [eol_](#)
The symbol representing end-of-line or end-of-input (\$).
- unsigned int [dot_](#) = 0
The position of the dot (·) in the production.

9.9.1 Detailed Description

Represents an LR(0) item used in LR automata construction.

An LR(0) item has a production of the form $A \rightarrow \alpha \bullet \beta$, where the dot indicates the current parsing position.

This structure tracks the antecedent (left-hand side), consequent (right-hand side), the dot position, and special symbols like EPSILON and end-of-line (\$).

9.9.2 Constructor & Destructor Documentation

9.9.2.1 Lr0Item() [1/2]

```
Lr0Item::Lr0Item (
    std::string antecedent,
    std::vector< std::string > consequent,
    std::string epsilon,
    std::string eol)
```

Constructs an LR(0) item with the dot at position 0.

Parameters

<i>antecedent</i>	The left-hand side non-terminal.
<i>consequent</i>	The right-hand side of the production.
<i>epsilon</i>	The EPSILON symbol.
<i>eol</i>	The end-of-line symbol.

Here is the caller graph for this function:



9.9.2.2 Lr0Item() [2/2]

```
Lr0Item::Lr0Item (
    std::string antecedent,
    std::vector< std::string > consequent,
    unsigned int dot,
    std::string epsilon,
    std::string eol)
```

Constructs an LR(0) item with a custom dot position.

Parameters

<i>antecedent</i>	The left-hand side non-terminal.
<i>consequent</i>	The right-hand side of the production.
<i>dot</i>	The position of the dot.
<i>epsilon</i>	The EPSILON symbol.
<i>eol</i>	The end-of-line symbol.

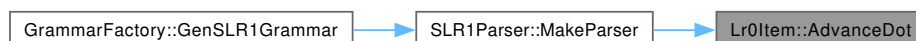
9.9.3 Member Function Documentation

9.9.3.1 AdvanceDot()

```
void Lr0Item::AdvanceDot ()
```

Advances the dot one position to the right.

Here is the caller graph for this function:



9.9.3.2 IsComplete()

```
bool Lr0Item::IsComplete () const
```

Checks whether the dot has reached the end of the production.

Returns

true if the item is complete; false otherwise.

Here is the caller graph for this function:



9.9.3.3 NextToDot()

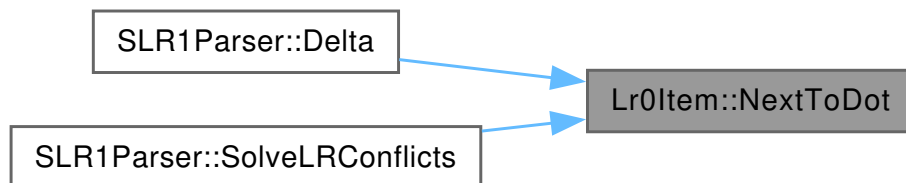
```
std::string Lr0Item::NextToDot () const
```

Returns the symbol immediately after the dot, or empty if the dot is at the end.

Returns

The symbol after the dot, or an empty string.

Here is the caller graph for this function:



9.9.3.4 operator==()

```
bool Lr0Item::operator== (
    const Lr0Item & other) const
```

Equality operator for comparing two LR(0) items.

Parameters

<i>other</i>	The item to compare with.
--------------	---------------------------

Returns

true if both items are equal; false otherwise.

Here is the call graph for this function:



9.9.3.5 PrintItem()

```
void Lr0Item::PrintItem () const
```

Prints the LR(0) item to the standard output in a human-readable format.

9.9.3.6 ToString()

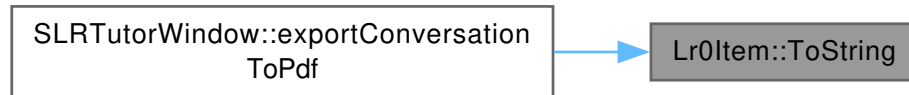
```
std::string Lr0Item::ToString () const
```

Converts the item to a string representation, including the dot position.

Returns

A string representation of the item.

Here is the caller graph for this function:



9.9.4 Member Data Documentation

9.9.4.1 antecedent_

```
std::string Lr0Item::antecedent_
```

The non-terminal on the left-hand side of the production.

9.9.4.2 consequent_

```
std::vector<std::string> Lr0Item::consequent_
```

The sequence of symbols on the right-hand side of the production.

9.9.4.3 dot_

```
unsigned int Lr0Item::dot_ = 0
```

The position of the dot (·) in the production.

9.9.4.4 eol_

```
std::string Lr0Item::eol_
```

The symbol representing end-of-line or end-of-input (\$).

9.9.4.5 epsilon_

```
std::string Lr0Item::epsilon_
```

The symbol representing the empty string (ϵ).

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

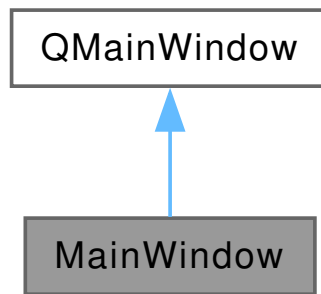
- [backend/lr0_item.hpp](#)
- [backend/lr0_item.cpp](#)

9.10 MainWindow Class Reference

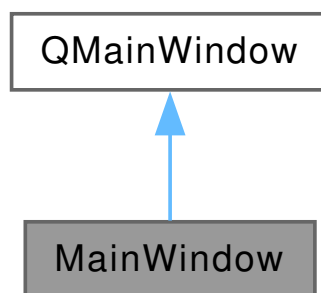
Main application window of [SyntaxTutor](#), managing levels, exercises, and UI state.

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Signals

- void `userLevelChanged` (unsigned lvl)
Emitted when the user's level changes.
- void `userLevelUp` (unsigned newLevel)
Emitted when the user levels up.

Public Member Functions

- `MainWindow` (QWidget *parent=nullptr)
Constructs the main window.
- `~MainWindow` ()
Destructor.
- unsigned `thresholdFor` (unsigned level)
Returns the required score threshold to unlock a level.
- unsigned `userLevel` () const
Returns the current user level.
- void `setUserLevel` (unsigned lvl)
Sets the user level, clamping it to the allowed maximum.

Properties

- unsigned `userLevel`

9.10.1 Detailed Description

Main application window of [SyntaxTutor](#), managing levels, exercises, and UI state.

This class serves as the central hub of the application. It handles level selection, navigation to LL(1) and SLR(1) exercises, tutorial management, settings persistence, and emits signals for user progress. It also includes UI logic for dynamic behavior like unlocking levels and changing language.

9.10.2 Constructor & Destructor Documentation

9.10.2.1 MainWindow()

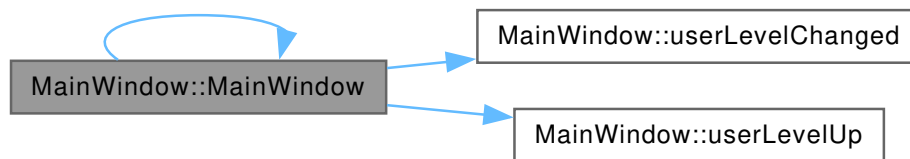
```
MainWindow::MainWindow (
    QWidget * parent = nullptr)
```

Constructs the main window.

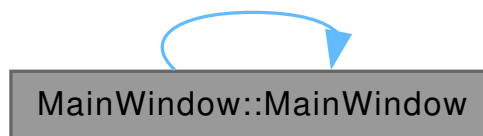
Parameters

<i>parent</i>	Parent widget.
---------------	----------------

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.2 ~MainWindow()

```
MainWindow::~MainWindow ()
```

Destructor.

9.10.3 Member Function Documentation

9.10.3.1 setUserLevel()

```
void MainWindow::setUserLevel (
    unsigned lvl) [inline]
```

Sets the user level, clamping it to the allowed maximum.

Parameters

<i>lvl</i>	New level to assign.
------------	----------------------

Here is the call graph for this function:



9.10.3.2 thresholdFor()

```
unsigned MainWindow::thresholdFor (
    unsigned level) [inline]
```

Returns the required score threshold to unlock a level.

Parameters

<i>level</i>	The level number.
--------------	-------------------

Returns

The score needed to unlock the given level.

9.10.3.3 `userLevel()`

```
unsigned MainWindow::userLevel () const [inline]
```

Returns the current user level.

9.10.3.4 `userLevelChanged`

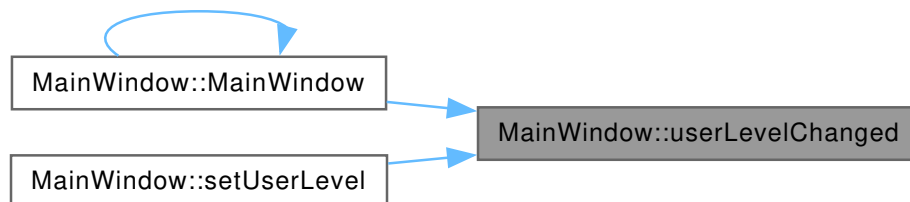
```
void MainWindow::userLevelChanged (
    unsigned lvl) [signal]
```

Emitted when the user's level changes.

Parameters

<i>lvl</i>	New user level.
------------	-----------------

Here is the caller graph for this function:



9.10.3.5 `userLevelUp`

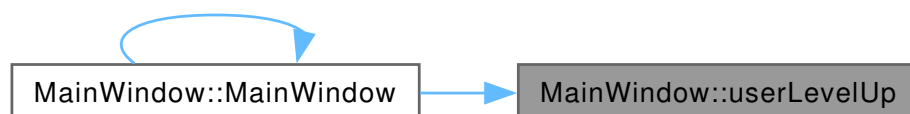
```
void MainWindow::userLevelUp (
    unsigned newLevel) [signal]
```

Emitted when the user levels up.

Parameters

<i>newLevel</i>	The new level achieved.
-----------------	-------------------------

Here is the caller graph for this function:



9.10.4 Property Documentation

9.10.4.1 userLevel

unsigned MainWindow::userLevel [read], [write]

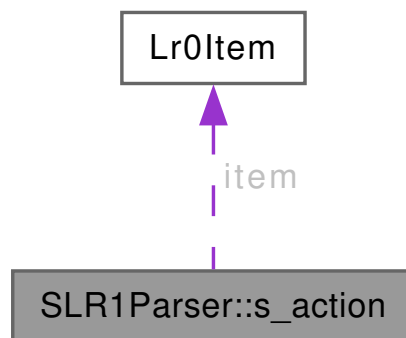
The documentation for this class was generated from the following files:

- [gui/mainwindow.h](#)
- [gui/mainwindow.cpp](#)

9.11 SLR1Parser::s_action Struct Reference

```
#include <slr1_parser.hpp>
```

Collaboration diagram for SLR1Parser::s_action:



Public Attributes

- const [Lr0Item](#) * [item](#)
- [Action](#) [action](#)

9.11.1 Member Data Documentation

9.11.1.1 action

[Action](#) SLR1Parser::s_action::action

9.11.1.2 item

const [Lr0Item](#)* SLR1Parser::s_action::item

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

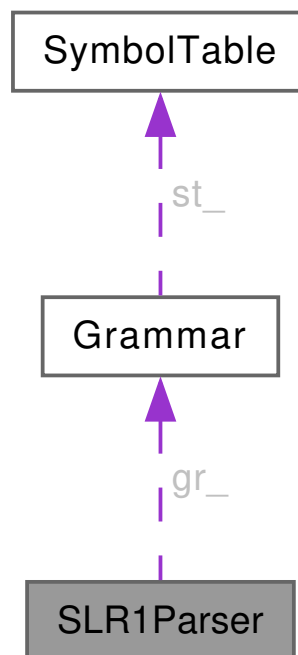
- [backend/slr1_parser.hpp](#)

9.12 SLR1Parser Class Reference

Implements an SLR(1) parser for context-free grammars.

```
#include <slr1_parser.hpp>
```

Collaboration diagram for SLR1Parser:



Classes

- struct [s_action](#)

Public Types

- enum class [Action](#) { [Shift](#) , [Reduce](#) , [Accept](#) , [Empty](#) }
Represents the possible actions in the SLR(1) parsing table.
- using [action_table](#)
Represents the action table for the SLR(1) parser.
- using [transition_table](#)
Represents the transition table for the SLR(1) parser.

Public Member Functions

- [SLR1Parser](#) ()=default
- [SLR1Parser](#) ([Grammar](#) gr)
- `std::unordered_set< Lr0Item > AllItems ()` const
Retrieves all LR(0) items in the grammar. This function returns a set of all LR(0) items derived from the grammar's productions. Each LR(0) item represents a production with a marker indicating the current position in the production (e.g., $A \rightarrow \alpha \bullet \beta$).
- `void Closure (std::unordered_set< Lr0Item > &items)`
Computes the closure of a set of LR(0) items.
- `void ClosureUtil (std::unordered_set< Lr0Item > &items, unsigned int size, std::unordered_set< std::string > &visited)`
Helper function for computing the closure of LR(0) items.
- `std::unordered_set< Lr0Item > Delta (const std::unordered_set< Lr0Item > &items, const std::string &str)`
Computes the GOTO transition (δ) for a given set of LR(0) items and a symbol. This function is equivalent to the $\delta(I, X)$ function in LR parsing, where it computes the set of items reached from a state I via symbol X .
- `bool SolveLRConflicts (const state &st)`
Resolves LR conflicts in a given state.

- void [First](#) (std::span< const std::string > rule, std::unordered_set< std::string > &result)
Calculates the FIRST set for a given production rule in a grammar.
- void [ComputeFirstSets](#) ()
Computes the FIRST sets for all non-terminal symbols in the grammar.
- void [ComputeFollowSets](#) ()
Computes the FOLLOW sets for all non-terminal symbols in the grammar. The FOLLOW set of a non-terminal symbol A contains all terminal symbols that can appear immediately after A in any sentential form derived from the grammar's start symbol. Additionally, if A can be the last symbol in a derivation, the end-of-input marker ($\$$) is included in its FOLLOW set. This function computes the FOLLOW sets using the following rules:
- std::unordered_set< std::string > [Follow](#) (const std::string &arg)
Computes the FOLLOW set for a given non-terminal symbol in the grammar.
- void [MakeInitialState](#) ()
Creates the initial state of the parser's state machine.
- bool [MakeParser](#) ()
Constructs the SLR(1) parsing tables (action and transition tables).
- std::string [PrintItems](#) (const std::unordered_set< [Lr0Item](#) > &items) const
Returns a string representation of a set of LR(0) items.

Public Attributes

- [Grammar](#) [gr_](#)
The grammar being processed by the parser.
- std::unordered_map< std::string, std::unordered_set< std::string > > [first_sets_](#)
Cached FIRST sets for all symbols in the grammar.
- std::unordered_map< std::string, std::unordered_set< std::string > > [follow_sets_](#)
Cached FOLLOW sets for all non-terminal symbols in the grammar.
- [action_table](#) [actions_](#)
The action table used by the parser to determine shift/reduce actions.
- [transition_table](#) [transitions_](#)
The transition table used by the parser to determine state transitions.
- std::unordered_set< [state](#) > [states_](#)
The set of states in the parser's state machine.

9.12.1 Detailed Description

Implements an SLR(1) parser for context-free grammars.

This class builds an SLR(1) parsing table and LR(0) automaton from a given grammar. It provides methods for computing closure sets, GOTO transitions, constructing states, and performing syntax analysis using the generated table.

9.12.2 Member Typedef Documentation

9.12.2.1 [action_table](#)

using [SLR1Parser::action_table](#)

Initial value:

```
std::map<unsigned int, std::map<std::string, SLR1Parser::s_action>>
```

Represents the action table for the SLR(1) parser.

The action table is a map that associates each state and input symbol with a specific action (Shift, Reduce, Accept, or Empty). It is used to determine the parser's behavior during the parsing process.

The table is structured as:

- Outer map: Keys are state IDs (unsigned int).
- Inner map: Keys are input symbols (std::string), and values are [s_action](#) structs representing the action to take.

9.12.2.2 transition_table

using `SLR1Parser::transition_table`

Initial value:

```
std::map<unsigned int, std::map<std::string, unsigned int>
```

Represents the transition table for the SLR(1) parser.

The transition table is a map that associates each state and symbol with the next state to transition to. It is used to guide the parser's state transitions during the parsing process.

The table is structured as:

- Outer map: Keys are state IDs (unsigned int).
- Inner map: Keys are symbols (std::string), and values are the next state IDs (unsigned int).

9.12.3 Member Enumeration Documentation

9.12.3.1 Action

```
enum class SLR1Parser::Action [strong]
```

Represents the possible actions in the SLR(1) parsing table.

This enumeration defines the types of actions that can be taken by the parser during the parsing process:

- `Shift`: Shift the input symbol onto the stack and transition to a new state.
- `Reduce`: Reduce a production rule and pop symbols from the stack.
- `Accept`: Accept the input as a valid string in the grammar.
- `Empty`: No action is defined for the current state and input symbol.

Enumerator

Shift	
Reduce	
Accept	
Empty	

9.12.4 Constructor & Destructor Documentation

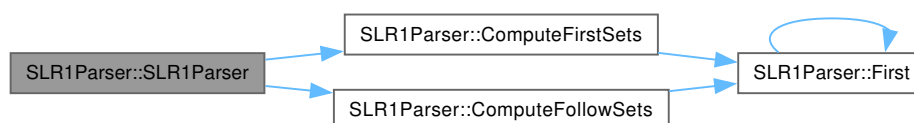
9.12.4.1 SLR1Parser() [1/2]

```
SLR1Parser::SLR1Parser () [default]
```

9.12.4.2 SLR1Parser() [2/2]

```
SLR1Parser::SLR1Parser (
    Grammar gr) [explicit]
```

Here is the call graph for this function:



9.12.5 Member Function Documentation

9.12.5.1 AllItems()

```
std::unordered_set< Lr0Item > SLR1Parser::AllItems () const
```

Retrieves all LR(0) items in the grammar. This function returns a set of all LR(0) items derived from the grammar's productions. Each LR(0) item represents a production with a marker indicating the current position in the production (e.g., $A \rightarrow \alpha \bullet \beta$).

Returns

A set of all LR(0) items in the grammar.

9.12.5.2 Closure()

```
void SLR1Parser::Closure (
    std::unordered_set< Lr0Item > & items)
```

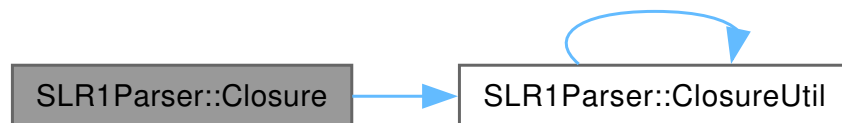
Computes the closure of a set of LR(0) items.

This function computes the closure of a given set of LR(0) items by adding all items that can be derived from the current items using the grammar's productions. The closure operation ensures that all possible derivations are considered when constructing the parser's states.

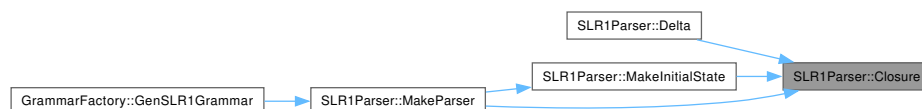
Parameters

<i>items</i>	The set of LR(0) items for which to compute the closure.
--------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.5.3 ClosureUtil()

```
void SLR1Parser::ClosureUtil (
    std::unordered_set< Lr0Item > & items,
    unsigned int size,
    std::unordered_set< std::string > & visited)
```

Helper function for computing the closure of LR(0) items.

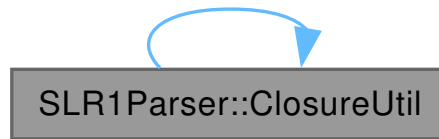
This function recursively computes the closure of a set of LR(0) items by adding items derived from non-terminal symbols. It avoids redundant work by tracking visited non-terminals and stopping when no new items are added.

Parameters

<i>items</i>	The set of LR(0) items being processed.
<i>size</i>	The size of the items set at the start of the current iteration.

<i>visited</i>	A set of non-terminals that have already been processed.
----------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.5.4 ComputeFirstSets()

```
void SLR1Parser::ComputeFirstSets ()
```

Computes the FIRST sets for all non-terminal symbols in the grammar.

This function calculates the FIRST set for each non-terminal symbol in the grammar by iteratively applying a least fixed-point algorithm. This approach ensures that the FIRST sets are fully populated by repeatedly expanding and updating the sets until no further changes occur (i.e., a fixed-point is reached). Here is the call graph for this function:



Here is the caller graph for this function:



9.12.5.5 ComputeFollowSets()

```
void SLR1Parser::ComputeFollowSets ()
```

Computes the FOLLOW sets for all non-terminal symbols in the grammar. The FOLLOW set of a non-terminal symbol A contains all terminal symbols that can appear immediately after A in any sentential form derived from the grammar's start symbol. Additionally, if A can be the last symbol in a derivation, the end-of-input marker ($\$$) is included in its FOLLOW set. This function computes the FOLLOW sets using the following rules:

1. Initialize $FOLLOW(S) = \{ \$ \}$, where S is the start symbol.
2. For each production rule of the form $A \rightarrow \alpha B \beta$:
 - Add $FIRST(\beta) \setminus \{ \epsilon \}$ to $FOLLOW(B)$.
 - If $\epsilon \in FIRST(\beta)$, add $FOLLOW(A)$ to $FOLLOW(B)$.
3. Repeat step 2 until no changes occur in any FOLLOW set. The computed FOLLOW sets are cached in the `follow_sets_` member variable for later use by the parser.

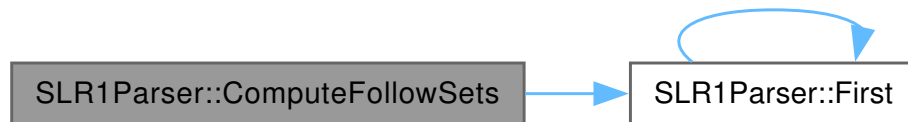
Note

This function assumes that the FIRST sets for all symbols have already been computed and are available in the `first_sets_` member variable.

See also

[First](#)
[follow_sets_](#)

Here is the call graph for this function:



Here is the caller graph for this function:

**9.12.5.6 Delta()**

```
std::unordered_set< Lr0Item > SLR1Parser::Delta (
    const std::unordered_set< Lr0Item > & items,
    const std::string & str)
```

Computes the GOTO transition (δ) for a given set of LR(0) items and a symbol. This function is equivalent to the $\delta(I, X)$ function in LR parsing, where it computes the set of items reached from a state I via symbol X .

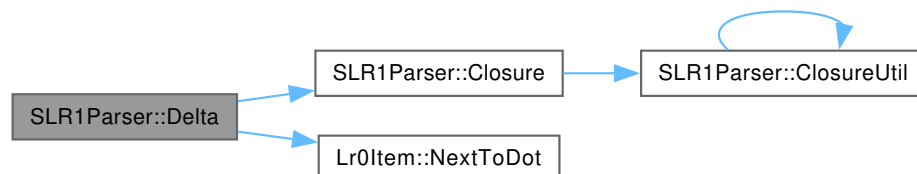
Parameters

<i>items</i>	The current set of LR(0) items (state).
<i>str</i>	The grammar symbol used for the transition.

Returns

The resulting item set after the GOTO transition.

Here is the call graph for this function:

**9.12.5.7 First()**

```
void SLR1Parser::First (
    std::span< const std::string > rule,
    std::unordered_set< std::string > & result)
```

Calculates the FIRST set for a given production rule in a grammar.

The FIRST set of a production rule contains all terminal symbols that can appear at the beginning of any string derived from that rule. If the rule can derive the empty string (epsilon), epsilon is included in the FIRST set.

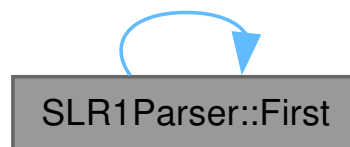
This function computes the FIRST set by examining each symbol in the production rule:

- If a terminal symbol is encountered, it is added directly to the FIRST set, as it is the starting symbol of some derivation.
- If a non-terminal symbol is encountered, its FIRST set is recursively computed and added to the result, excluding epsilon unless it is followed by another symbol that could also lead to epsilon.
- If the entire rule could derive epsilon (i.e., each symbol in the rule can derive epsilon), then epsilon is added to the FIRST set.

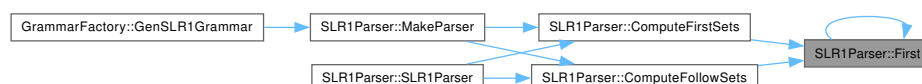
Parameters

<i>rule</i>	A span of strings representing the production rule for which to compute the FIRST set. Each string in the span is a symbol (either terminal or non-terminal).
<i>result</i>	A reference to an unordered set of strings where the computed FIRST set will be stored. The set will contain all terminal symbols that can start derivations of the rule, and possibly epsilon if the rule can derive an empty string.

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.5.8 Follow()

```
std::unordered_set< std::string > SLR1Parser::Follow (
    const std::string & arg)
```

Computes the FOLLOW set for a given non-terminal symbol in the grammar.

The FOLLOW set for a non-terminal symbol includes all symbols that can appear immediately to the right of that symbol in any derivation, as well as any end-of-input markers if the symbol can appear at the end of derivations. FOLLOW sets are used in LL(1) parsing table construction to determine possible continuations after a non-terminal.

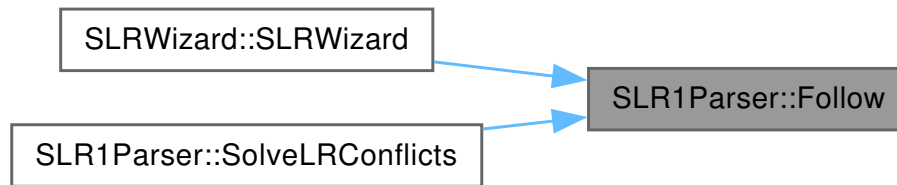
Parameters

<i>arg</i>	Non-terminal symbol for which to compute the FOLLOW set.
------------	--

Returns

An unordered set of strings containing symbols that form the FOLLOW set for `arg`.

Here is the caller graph for this function:

**9.12.5.9 MakeInitialState()**

```
void SLR1Parser::MakeInitialState ()
```

Creates the initial state of the parser's state machine.

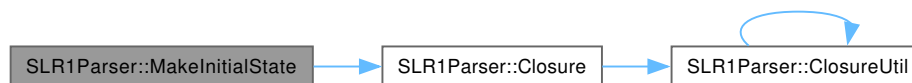
This function initializes the starting state of the parser by computing the closure of the initial set of LR(0) items derived from the grammar's start symbol. The initial state is added to the `states_` set, and its transitions are prepared for further processing in the parser construction.

See also

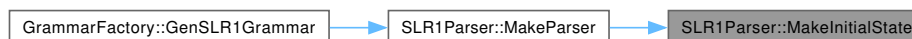
[states_](#)

[transitions_](#)

Here is the call graph for this function:



Here is the caller graph for this function:

**9.12.5.10 MakeParser()**

```
bool SLR1Parser::MakeParser ()
```

Constructs the SLR(1) parsing tables (action and transition tables).

This function builds the SLR(1) parsing tables by computing the canonical collection of LR(0) items, generating the action and transition tables, and resolving conflicts (if any). It returns `true` if the grammar is SLR(1) and the tables are successfully constructed, or `false` if a conflict is detected that cannot be resolved.

Returns

`true` if the parsing tables are successfully constructed, `false` if the grammar is not SLR(1) or a conflict is encountered.

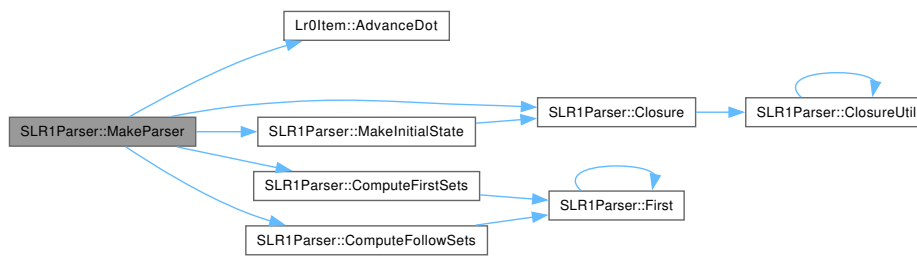
See also

[actions_](#)

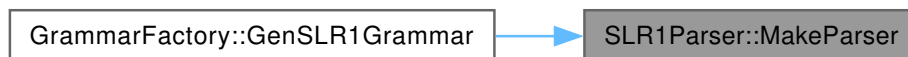
[transitions_](#)

[states_](#)

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.5.11 PrintItems()

```
std::string SLR1Parser::PrintItems (
    const std::unordered_set< Lr0Item > & items) const
```

Returns a string representation of a set of LR(0) items.

This function converts a set of LR(0) items into a human-readable string, including dot positions, to help visualize parser states.

Parameters

<i>items</i>	The set of LR(0) items to print.
--------------	----------------------------------

Returns

A formatted string representation of the items.

9.12.5.12 SolveLRConflicts()

```
bool SLR1Parser::SolveLRConflicts (
    const state & st)
```

Resolves LR conflicts in a given state.

This function attempts to resolve shift/reduce or reduce/reduce conflicts in a given state using SLR(1) parsing rules. It checks the FOLLOW sets of non-terminals to determine the correct action and updates the action table accordingly.

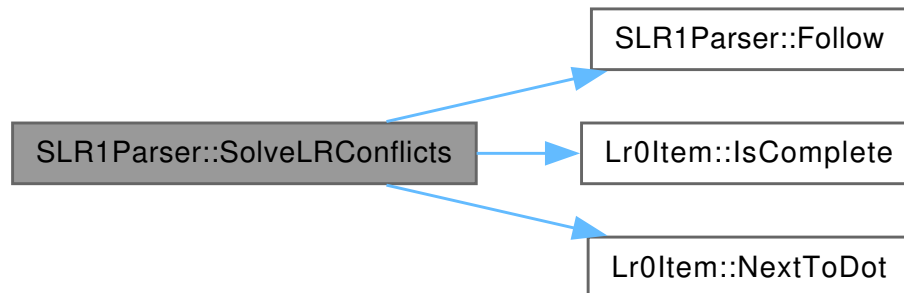
Parameters

<i>st</i>	The state in which to resolve conflicts.
-----------	--

Returns

`true` if all conflicts are resolved, `false` if an unresolvable conflict is detected.

Here is the call graph for this function:

**9.12.6 Member Data Documentation****9.12.6.1 actions_**

`action_table` `SLR1Parser::actions_`

The action table used by the parser to determine shift/reduce actions.

9.12.6.2 first_sets_

`std::unordered_map<std::string, std::unordered_set<std::string> >` `SLR1Parser::first_sets_`
 Cached FIRST sets for all symbols in the grammar.

9.12.6.3 follow_sets_

`std::unordered_map<std::string, std::unordered_set<std::string> >` `SLR1Parser::follow_sets_`
 Cached FOLLOW sets for all non-terminal symbols in the grammar.

9.12.6.4 gr_

`Grammar` `SLR1Parser::gr_`

The grammar being processed by the parser.

9.12.6.5 states_

`std::unordered_set<state>` `SLR1Parser::states_`
 The set of states in the parser's state machine.

9.12.6.6 transitions_

`transition_table` `SLR1Parser::transitions_`

The transition table used by the parser to determine state transitions.

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

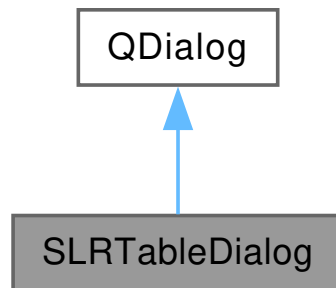
- [backend/slr1_parser.hpp](#)
- [backend/slr1_parser.cpp](#)

9.13 SLRTableDialog Class Reference

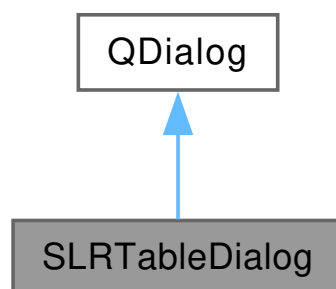
Dialog window for completing and submitting an SLR(1) parsing table.

```
#include <slrtabledialog.h>
```

Inheritance diagram for SLRTableDialog:



Collaboration diagram for SLRTableDialog:



Public Member Functions

- [SLRTableDialog](#) (int rowCount, int colCount, const QStringList &colHeaders, QWidget *parent=nullptr, QVector< QVector< QString > > *initialData=nullptr)
Constructs the SLR(1) table dialog.
- QVector< QVector< QString > > [getTableData](#) () const
Retrieves the content of the table after user interaction.
- void [setInitialData](#) (const QVector< QVector< QString > > &data)
Fills the table with existing data.

9.13.1 Detailed Description

Dialog window for completing and submitting an SLR(1) parsing table.

This class displays a table-based UI for students to fill in the ACTION and GOTO parts of the SLR(1) parsing table. It supports initializing the table with data, retrieving user input, and integrating with correction logic in tutorial or challenge mode.

9.13.2 Constructor & Destructor Documentation

9.13.2.1 SLRTableDialog()

```

SLRTableDialog::SLRTableDialog (
    int rowCount,
    int colCount,
    const QStringList & colHeaders,
    QWidget * parent = nullptr,
    QVector< QVector< QString > > * initialData = nullptr)

```

Constructs the SLR(1) table dialog.

Parameters

<i>rowCount</i>	Number of rows (usually equal to number of LR(0) states).
<i>colCount</i>	Number of columns (symbols = terminals + non-terminals).
<i>colHeaders</i>	Header labels for the columns.
<i>parent</i>	Parent widget.
<i>initialData</i>	Optional initial data to pre-fill the table.

9.13.3 Member Function Documentation**9.13.3.1 `getTableData()`**

`QVector< QVector< QString > > SLRTableDialog::getTableData () const`
Retrieves the content of the table after user interaction.

Returns

A 2D vector representing the current table values.

9.13.3.2 `setInitialData()`

```
void SLRTableDialog::setInitialData (
    const QVector< QVector< QString > > & data)
```

Fills the table with existing data.

This method is used to show a previous user submission (e.g., during retries or feedback).

Parameters

<i>data</i>	2D vector containing the table data to display.
-------------	---

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

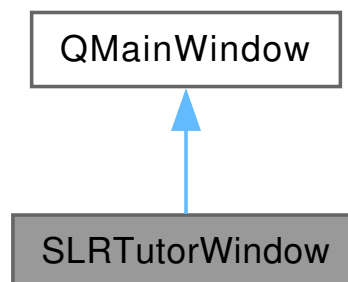
- [gui/slrtabledialog.h](#)
- [gui/slrtabledialog.cpp](#)

9.14 SLRTutorWindow Class Reference

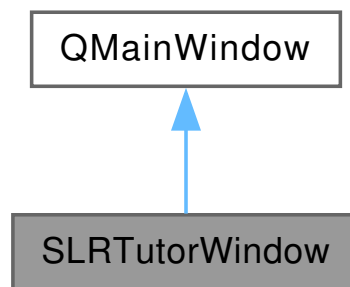
Main window for the SLR(1) interactive tutoring mode in [SyntaxTutor](#).

```
#include <slrtutorwindow.h>
```

Inheritance diagram for SLRTutorWindow:



Collaboration diagram for SLRTutorWindow:



Signals

- void [sessionFinished](#) (int cntRight, int cntWrong)

Public Member Functions

- [SLRTutorWindow](#) (const [Grammar](#) &g, [TutorialManager](#) *tm=nullptr, QWidget *parent=nullptr)
Constructs the SLR(1) tutor window with a given grammar.
- [~SLRTutorWindow](#) ()
- QString [generateQuestion](#) ()
Generates a new question for the current tutor state.
- void [updateState](#) (bool isCorrect)
Updates tutor state based on whether the last answer was correct.
- QString [FormatGrammar](#) (const [Grammar](#) &grammar)
- void [fillSortedGrammar](#) ()
< Utility for displaying grammar
- void [addMessage](#) (const QString &text, bool isUser)
< Prepares grammar in display-friendly format
- void [exportConversationToPdf](#) (const QString &filePath)
< Add message to chat
- void [showTable](#) ()
< Export full interaction
- void [launchSLRWizard](#) ()
< Render SLR(1) table
- void [updateProgressPanel](#) ()
- void [addUserState](#) (unsigned id)
< Refresh visual progress
- void [addUserTransition](#) (unsigned fromId, const std::string &symbol, unsigned told)
< Register a user-created state
- void [animateLabelPop](#) (QLabel *label)
- void [animateLabelColor](#) (QLabel *label, const QColor &flashColor)
- void [wrongAnimation](#) ()
- void [wrongUserResponseAnimation](#) ()
- void [markLastUserIncorrect](#) ()
- bool [verifyResponse](#) (const QString &userResponse)
- bool [verifyResponseForA](#) (const QString &userResponse)
- bool [verifyResponseForA1](#) (const QString &userResponse)
- bool [verifyResponseForA2](#) (const QString &userResponse)
- bool [verifyResponseForA3](#) (const QString &userResponse)
- bool [verifyResponseForA4](#) (const QString &userResponse)
- bool [verifyResponseForB](#) (const QString &userResponse)

- bool [verifyResponseForC](#) (const QString &userResponse)
- bool [verifyResponseForCA](#) (const QString &userResponse)
- bool [verifyResponseForCB](#) (const QString &userResponse)
- bool [verifyResponseForD](#) (const QString &userResponse)
- bool [verifyResponseForD1](#) (const QString &userResponse)
- bool [verifyResponseForD2](#) (const QString &userResponse)
- bool [verifyResponseForE](#) (const QString &userResponse)
- bool [verifyResponseForE1](#) (const QString &userResponse)
- bool [verifyResponseForE2](#) (const QString &userResponse)
- bool [verifyResponseForF](#) (const QString &userResponse)
- bool [verifyResponseForFA](#) (const QString &userResponse)
- bool [verifyResponseForG](#) (const QString &userResponse)
- bool [verifyResponseForH](#) ()
- QString [solution](#) (const std::string &state)
- std::unordered_set< [Lr0Item](#) > [solutionForA](#) ()
- QString [solutionForA1](#) ()
- QString [solutionForA2](#) ()
- std::vector< std::pair< std::string, std::vector< std::string > > > [solutionForA3](#) ()
- std::unordered_set< [Lr0Item](#) > [solutionForA4](#) ()
- unsigned [solutionForB](#) ()
- unsigned [solutionForC](#) ()
- QStringList [solutionForCA](#) ()
- std::unordered_set< [Lr0Item](#) > [solutionForCB](#) ()
- QStringList [solutionForD](#) ()
- QString [solutionForD1](#) ()
- QString [solutionForD2](#) ()
- std::ptrdiff_t [solutionForE](#) ()
- QSet< unsigned > [solutionForE1](#) ()
- QMap< unsigned, unsigned > [solutionForE2](#) ()
- QSet< unsigned > [solutionForF](#) ()
- QSet< QString > [solutionForFA](#) ()
- QSet< QString > [solutionForG](#) ()
- QString [feedback](#) ()
- QString [feedbackForA](#) ()
- QString [feedbackForA1](#) ()
- QString [feedbackForA2](#) ()
- QString [feedbackForA3](#) ()
- QString [feedbackForA4](#) ()
- QString [feedbackForAPrime](#) ()
- QString [feedbackForB](#) ()
- QString [feedbackForB1](#) ()
- QString [feedbackForB2](#) ()
- QString [feedbackForBPrime](#) ()
- QString [feedbackForC](#) ()
- QString [feedbackForCA](#) ()
- QString [feedbackForCB](#) ()
- QString [feedbackForD](#) ()
- QString [feedbackForD1](#) ()
- QString [feedbackForD2](#) ()
- QString [feedbackForDPrime](#) ()
- QString [feedbackForE](#) ()
- QString [feedbackForE1](#) ()
- QString [feedbackForE2](#) ()
- QString [feedbackForF](#) ()
- QString [feedbackForFA](#) ()

- QString [feedbackForG](#) ()
- QString [TeachDeltaFunction](#) (const std::unordered_set< [Lr0Item](#) > &items, const QString &symbol)
- void [TeachClosureStep](#) (std::unordered_set< [Lr0Item](#) > &items, unsigned int size, std::unordered_set< std::string > &visited, int depth, QString &output)
- QString [TeachClosure](#) (const std::unordered_set< [Lr0Item](#) > &initialItems)

Protected Member Functions

- void [closeEvent](#) (QCloseEvent *event) override

9.14.1 Detailed Description

Main window for the SLR(1) interactive tutoring mode in [SyntaxTutor](#).

This class implements an interactive, step-by-step tutorial to teach students how to construct SLR(1) parsing tables, including closure, GOTO, automaton construction, FOLLOW sets, and the final table.

It supports animated feedback, pedagogical guidance, error correction, and export of the tutoring session.

The tutor follows a finite-state flow ([StateSlr](#)) to structure learning, with corrective explanations and automatic evaluation at each step.

9.14.2 Constructor & Destructor Documentation

9.14.2.1 SLRTutorWindow()

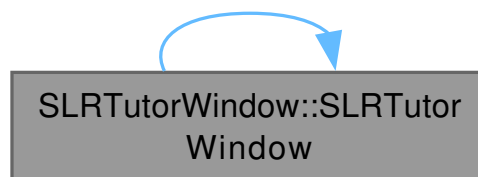
```
SLRTutorWindow::SLRTutorWindow (
    const Grammar & g,
    TutorialManager * tm = nullptr,
    QWidget * parent = nullptr) [explicit]
```

Constructs the SLR(1) tutor window with a given grammar.

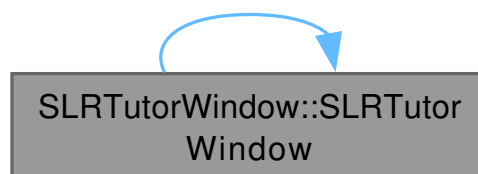
Parameters

<i>g</i>	The grammar used for the session.
<i>tm</i>	Optional pointer to the tutorial manager (for guided tour).
<i>parent</i>	Parent widget.

Here is the call graph for this function:



Here is the caller graph for this function:



9.14.2.2 ~SLRTutorWindow()

```
SLRTutorWindow::~SLRTutorWindow ()
```

9.14.3 Member Function Documentation**9.14.3.1 addMessage()**

```
void SLRTutorWindow::addMessage (
    const QString & text,
    bool isUser)
```

< Prepares grammar in display-friendly format

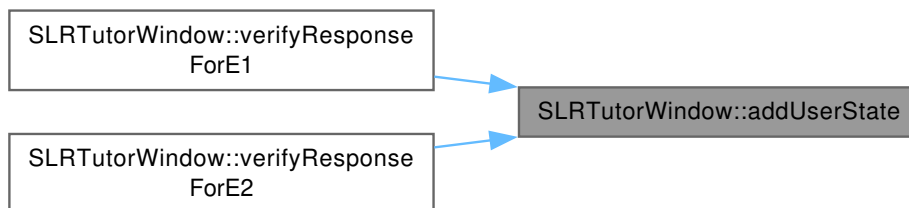
Here is the caller graph for this function:

**9.14.3.2 addUserState()**

```
void SLRTutorWindow::addUserState (
    unsigned id)
```

< Refresh visual progress

Here is the caller graph for this function:

**9.14.3.3 addUserTransition()**

```
void SLRTutorWindow::addUserTransition (
    unsigned fromId,
    const std::string & symbol,
    unsigned toId)
```

< Register a user-created state

9.14.3.4 animateLabelColor()

```
void SLRTutorWindow::animateLabelColor (
    QLabel * label,
    const QColor & flashColor)
```

9.14.3.5 animateLabelPop()

```
void SLRTutorWindow::animateLabelPop (
    QLabel * label)
```

9.14.3.6 closeEvent()

```
void SLRTutorWindow::closeEvent (
    QCloseEvent * event) [inline], [override], [protected]
```

Here is the call graph for this function:

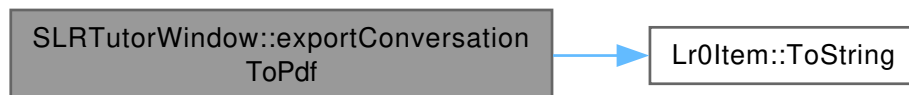


9.14.3.7 exportConversationToPdf()

```
void SLRTutorWindow::exportConversationToPdf (
    const QString & filePath)
```

< Add message to chat

Here is the call graph for this function:



9.14.3.8 feedback()

```
QString SLRTutorWindow::feedback ()
```

9.14.3.9 feedbackForA()

```
QString SLRTutorWindow::feedbackForA ()
```

9.14.3.10 feedbackForA1()

```
QString SLRTutorWindow::feedbackForA1 ()
```

9.14.3.11 feedbackForA2()

```
QString SLRTutorWindow::feedbackForA2 ()
```

9.14.3.12 feedbackForA3()

```
QString SLRTutorWindow::feedbackForA3 ()
```

9.14.3.13 feedbackForA4()

```
QString SLRTutorWindow::feedbackForA4 ()
```

9.14.3.14 feedbackForAPrime()

```
QString SLRTutorWindow::feedbackForAPrime ()
```

9.14.3.15 feedbackForB()

```
QString SLRTutorWindow::feedbackForB ()
```

9.14.3.16 feedbackForB1()

```
QString SLRTutorWindow::feedbackForB1 ()
```

9.14.3.17 feedbackForB2()

```
QString SLRTutorWindow::feedbackForB2 ()
```

9.14.3.18 feedbackForBPrime()

```
QString SLRTutorWindow::feedbackForBPrime ()
```

9.14.3.19 feedbackForC()

```
QString SLRTutorWindow::feedbackForC ()
```

9.14.3.20 feedbackForCA()

```
QString SLRTutorWindow::feedbackForCA ()
```

9.14.3.21 feedbackForCB()

```
QString SLRTutorWindow::feedbackForCB ()
```

9.14.3.22 feedbackForD()

```
QString SLRTutorWindow::feedbackForD ()
```

9.14.3.23 feedbackForD1()

```
QString SLRTutorWindow::feedbackForD1 ()
```

9.14.3.24 feedbackForD2()

```
QString SLRTutorWindow::feedbackForD2 ()
```

9.14.3.25 feedbackForDPrime()

```
QString SLRTutorWindow::feedbackForDPrime ()
```

9.14.3.26 feedbackForE()

```
QString SLRTutorWindow::feedbackForE ()
```

9.14.3.27 feedbackForE1()

```
QString SLRTutorWindow::feedbackForE1 ()
```

9.14.3.28 feedbackForE2()

```
QString SLRTutorWindow::feedbackForE2 ()
```

9.14.3.29 feedbackForF()

```
QString SLRTutorWindow::feedbackForF ()
```

9.14.3.30 feedbackForFA()

```
QString SLRTutorWindow::feedbackForFA ()
```

9.14.3.31 feedbackForG()

```
QString SLRTutorWindow::feedbackForG ()
```

9.14.3.32 fillSortedGrammar()

```
void SLRTutorWindow::fillSortedGrammar ()  
< Utility for displaying grammar
```

9.14.3.33 FormatGrammar()

```
QString SLRTutorWindow::FormatGrammar (
    const Grammar & grammar)
```

9.14.3.34 generateQuestion()

```
QString SLRTutorWindow::generateQuestion ()
```

Generates a new question for the current tutor state.

Returns

The formatted question string.

9.14.3.35 launchSLRWizard()

```
void SLRTutorWindow::launchSLRWizard ()
```

< Render SLR(1) table

9.14.3.36 markLastUserIncorrect()

```
void SLRTutorWindow::markLastUserIncorrect ()
```

9.14.3.37 sessionFinished

```
void SLRTutorWindow::sessionFinished (
    int cntRight,
    int cntWrong) [signal]
```

Here is the caller graph for this function:

**9.14.3.38 showTable()**

```
void SLRTutorWindow::showTable ()
```

< Export full interaction

Here is the caller graph for this function:

**9.14.3.39 solution()**

```
QString SLRTutorWindow::solution (
    const std::string & state)
```

9.14.3.40 solutionForA()

```
std::unordered_set< Lr0Item > SLRTutorWindow::solutionForA ()
```

9.14.3.41 solutionForA1()

```
QString SLRTutorWindow::solutionForA1 ()
```

9.14.3.42 solutionForA2()

```
QString SLRTutorWindow::solutionForA2 ()
```

9.14.3.43 solutionForA3()

```
std::vector< std::pair< std::string, std::vector< std::string > > > SLRTutorWindow::solutionForA3 ()
```

9.14.3.44 solutionForA4()

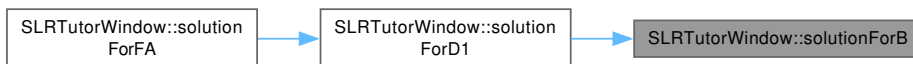
```
std::unordered_set< Lr0Item > SLRTutorWindow::solutionForA4 ()
```

Here is the caller graph for this function:

**9.14.3.45 solutionForB()**

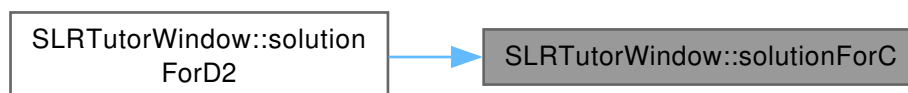
```
unsigned SLRTutorWindow::solutionForB ()
```

Here is the caller graph for this function:

**9.14.3.46 solutionForC()**

```
unsigned SLRTutorWindow::solutionForC ()
```

Here is the caller graph for this function:

**9.14.3.47 solutionForCA()**

```
QStringList SLRTutorWindow::solutionForCA ()
```

9.14.3.48 solutionForCB()

```
std::unordered_set< Lr0Item > SLRTutorWindow::solutionForCB ()
```

9.14.3.49 solutionForD()

```
QStringList SLRTutorWindow::solutionForD ()
```

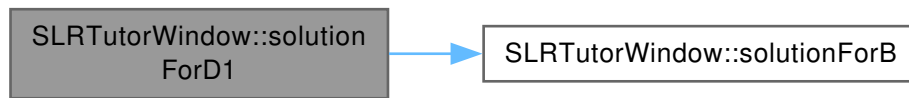
Here is the call graph for this function:



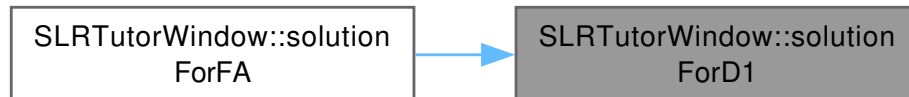
9.14.3.50 solutionForD1()

```
QString SLRTutorWindow::solutionForD1 ()
```

Here is the call graph for this function:

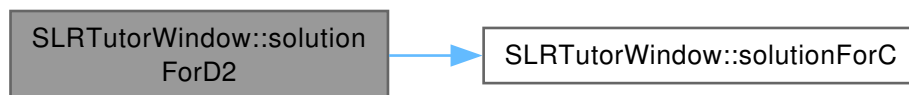


Here is the caller graph for this function:

**9.14.3.51 solutionForD2()**

```
QString SLRTutorWindow::solutionForD2 ()
```

Here is the call graph for this function:

**9.14.3.52 solutionForE()**

```
std::ptrdiff_t SLRTutorWindow::solutionForE ()
```

9.14.3.53 solutionForE1()

```
QSet< unsigned > SLRTutorWindow::solutionForE1 ()
```

9.14.3.54 solutionForE2()

```
QMap< unsigned, unsigned > SLRTutorWindow::solutionForE2 ()
```

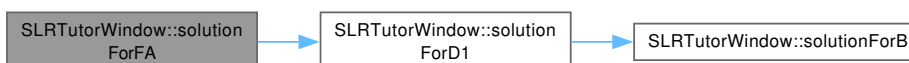
9.14.3.55 solutionForF()

```
QSet< unsigned > SLRTutorWindow::solutionForF ()
```

9.14.3.56 solutionForFA()

```
QSet< QString > SLRTutorWindow::solutionForFA ()
```

Here is the call graph for this function:

**9.14.3.57 solutionForG()**

```
QSet< QString > SLRTutorWindow::solutionForG ()
```


9.14.3.58 TeachClosure()

```
QString SLRTutorWindow::TeachClosure (
    const std::unordered_set< Lr0Item > & initialItems)
```

9.14.3.59 TeachClosureStep()

```
void SLRTutorWindow::TeachClosureStep (
    std::unordered_set< Lr0Item > & items,
    unsigned int size,
    std::unordered_set< std::string > & visited,
    int depth,
    QString & output)
```

9.14.3.60 TeachDeltaFunction()

```
QString SLRTutorWindow::TeachDeltaFunction (
    const std::unordered_set< Lr0Item > & items,
    const QString & symbol)
```

9.14.3.61 updateProgressPanel()

```
void SLRTutorWindow::updateProgressPanel ()
```

9.14.3.62 updateState()

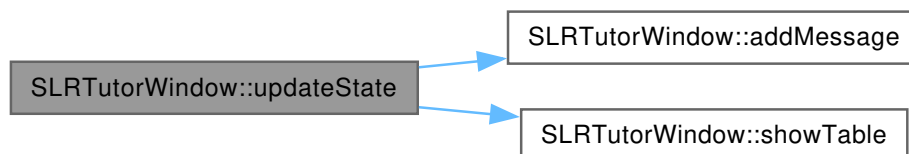
```
void SLRTutorWindow::updateState (
    bool isCorrect)
```

Updates tutor state based on whether the last answer was correct.

Parameters

<i>isCorrect</i>	Whether the user's answer was correct.
------------------	--

Here is the call graph for this function:

**9.14.3.63 verifyResponse()**

```
bool SLRTutorWindow::verifyResponse (
    const QString & userResponse)
```

9.14.3.64 verifyResponseForA()

```
bool SLRTutorWindow::verifyResponseForA (
    const QString & userResponse)
```

9.14.3.65 verifyResponseForA1()

```
bool SLRTutorWindow::verifyResponseForA1 (
    const QString & userResponse)
```

9.14.3.66 verifyResponseForA2()

```
bool SLRTutorWindow::verifyResponseForA2 (  
    const QString & userResponse)
```

9.14.3.67 verifyResponseForA3()

```
bool SLRTutorWindow::verifyResponseForA3 (  
    const QString & userResponse)
```

9.14.3.68 verifyResponseForA4()

```
bool SLRTutorWindow::verifyResponseForA4 (  
    const QString & userResponse)
```

9.14.3.69 verifyResponseForB()

```
bool SLRTutorWindow::verifyResponseForB (  
    const QString & userResponse)
```

9.14.3.70 verifyResponseForC()

```
bool SLRTutorWindow::verifyResponseForC (  
    const QString & userResponse)
```

9.14.3.71 verifyResponseForCA()

```
bool SLRTutorWindow::verifyResponseForCA (  
    const QString & userResponse)
```

9.14.3.72 verifyResponseForCB()

```
bool SLRTutorWindow::verifyResponseForCB (  
    const QString & userResponse)
```

9.14.3.73 verifyResponseForD()

```
bool SLRTutorWindow::verifyResponseForD (  
    const QString & userResponse)
```

9.14.3.74 verifyResponseForD1()

```
bool SLRTutorWindow::verifyResponseForD1 (  
    const QString & userResponse)
```

9.14.3.75 verifyResponseForD2()

```
bool SLRTutorWindow::verifyResponseForD2 (  
    const QString & userResponse)
```

9.14.3.76 verifyResponseForE()

```
bool SLRTutorWindow::verifyResponseForE (  
    const QString & userResponse)
```

9.14.3.77 verifyResponseForE1()

```
bool SLRTutorWindow::verifyResponseForE1 (
    const QString & userResponse)
```

Here is the call graph for this function:

**9.14.3.78 verifyResponseForE2()**

```
bool SLRTutorWindow::verifyResponseForE2 (
    const QString & userResponse)
```

Here is the call graph for this function:

**9.14.3.79 verifyResponseForF()**

```
bool SLRTutorWindow::verifyResponseForF (
    const QString & userResponse)
```

9.14.3.80 verifyResponseForFA()

```
bool SLRTutorWindow::verifyResponseForFA (
    const QString & userResponse)
```

9.14.3.81 verifyResponseForG()

```
bool SLRTutorWindow::verifyResponseForG (
    const QString & userResponse)
```

9.14.3.82 verifyResponseForH()

```
bool SLRTutorWindow::verifyResponseForH ()
```

9.14.3.83 wrongAnimation()

```
void SLRTutorWindow::wrongAnimation ()
```

9.14.3.84 wrongUserResponseAnimation()

```
void SLRTutorWindow::wrongUserResponseAnimation ()
```

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

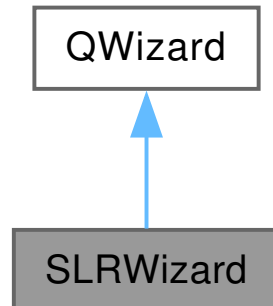
- [gui/slrtutorwindow.h](#)
- [gui/slrtutorwindow.cpp](#)

9.15 SLRWizard Class Reference

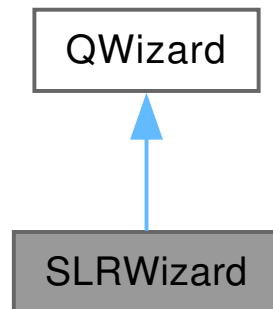
Interactive assistant that guides the student step-by-step through the SLR(1) parsing table.

```
#include <slrwizard.h>
```

Inheritance diagram for SLRWizard:



Collaboration diagram for SLRWizard:



Public Member Functions

- [SLRWizard](#) ([SLR1Parser](#) &parser, const QVector< QVector< QString > > &rawTable, const QString←
List &colHeaders, const QVector< QPair< QString, QVector< QString > > > &sortedGrammar, QWidget
*parent=nullptr)
Constructs the SLR(1) wizard with all necessary parsing context.
- QVector< QString > [stdVectorToQVector](#) (const std::vector< std::string > &vec)
Converts a std::vector<std::string> to QVector<QString> for UI compatibility.

9.15.1 Detailed Description

Interactive assistant that guides the student step-by-step through the SLR(1) parsing table.

This wizard-based dialog presents the user with one cell of the SLR(1) parsing table at a time, asking them to deduce the correct ACTION or GOTO entry based on the LR(0) automaton and FOLLOW sets. It is designed as an educational aid to explain the reasoning behind each parsing decision.

Each page includes:

- The current state and symbol (terminal or non-terminal).
- A guided explanation based on the grammar and LR(0) state.
- The expected entry (e.g., s3, r1, acc, or a state number).

9.15.2 Constructor & Destructor Documentation

9.15.2.1 SLRWizard()

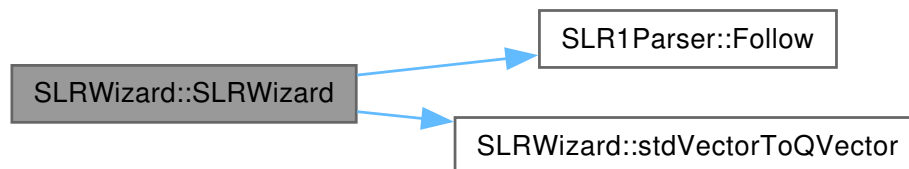
```
SLRWizard::SLRWizard (
    SLR1Parser & parser,
    const QVector< QVector< QString > > & rawTable,
    const QStringList & colHeaders,
    const QVector< QPair< QString, QVector< QString > > > & sortedGrammar,
    QWidget * parent = nullptr) [inline]
```

Constructs the SLR(1) wizard with all necessary parsing context.

Parameters

<i>parser</i>	The SLR(1) parser instance containing the LR(0) states and transitions.
<i>rawTable</i>	The target parsing table (student version or reference).
<i>colHeaders</i>	Header symbols (terminals and non-terminals).
<i>sortedGrammar</i>	Ordered list of grammar rules for reduce explanations.
<i>parent</i>	Parent widget.

Here is the call graph for this function:



9.15.3 Member Function Documentation

9.15.3.1 stdVectorToQVector()

```
QVector< QString > SLRWizard::stdVectorToQVector (
    const std::vector< std::string > & vec) [inline]
```

Converts a `std::vector<std::string>` to `QVector<QString>` for UI compatibility.

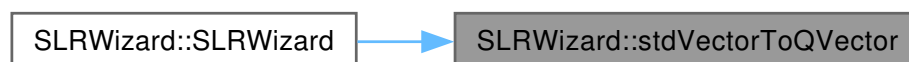
Parameters

<i>vec</i>	The input vector of strings.
------------	------------------------------

Returns

A `QVector` of `QString`s.

Here is the caller graph for this function:



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

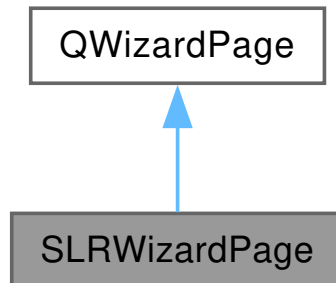
- [gui/slrwizard.h](#)

9.16 SLRWizardPage Class Reference

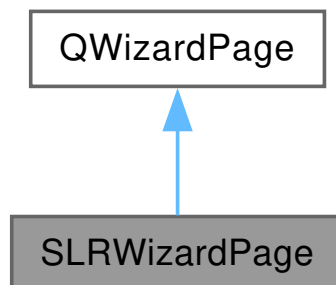
A single step in the SLR(1) guided assistant for table construction.

```
#include <slrwizardpage.h>
```

Inheritance diagram for SLRWizardPage:



Collaboration diagram for SLRWizardPage:



Public Member Functions

- [SLRWizardPage](#) (int [state](#), const QString &symbol, const QString &explanation, const QString &expected, QWidget *parent=nullptr)

Constructs a page for a specific cell in the SLR(1) table.

9.16.1 Detailed Description

A single step in the SLR(1) guided assistant for table construction.

This wizard page presents a specific (state, symbol) cell in the SLR(1) parsing table, and prompts the student to enter the correct ACTION or GOTO value.

The page checks the user's input against the expected answer and provides immediate feedback, disabling the "Next" button until the correct response is entered.

9.16.2 Constructor & Destructor Documentation

9.16.2.1 SLRWizardPage()

```
SLRWizardPage::SLRWizardPage (
    int state,
    const QString & symbol,
    const QString & explanation,
    const QString & expected,
    QWidget * parent = nullptr) [inline]
```

Constructs a page for a specific cell in the SLR(1) table.

Parameters

<i>state</i>	The state ID (row index in the table).
<i>symbol</i>	The grammar symbol (column header).
<i>explanation</i>	A pedagogical explanation shown to the user.
<i>expected</i>	The expected answer (e.g., "s2", "r1", "acc", or a state number).
<i>parent</i>	The parent widget.

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

- [gui/slrwizardpage.h](#)

9.17 state Struct Reference

Represents a state in the LR(0) automaton.

```
#include <state.hpp>
```

Public Member Functions

- `bool operator==(const state &other) const`
Equality operator for comparing states based on their items.

Public Attributes

- `std::unordered_set< Lr0Item > items_`
The set of LR(0) items that make up this state.
- `unsigned int id_`
Unique identifier of the state.

9.17.1 Detailed Description

Represents a state in the LR(0) automaton.

Each state consists of a unique identifier and a set of LR(0) items that define its core. States are used to build the SLR(1) parsing table.

9.17.2 Member Function Documentation

9.17.2.1 operator==()

```
bool state::operator==(
    const state & other) const [inline]
```

Equality operator for comparing states based on their items.

Parameters

<i>other</i>	The state to compare with.
--------------	----------------------------

Returns

true if both states have the same item set; false otherwise.

9.17.3 Member Data Documentation

9.17.3.1 id_

`unsigned int state::id_`

Unique identifier of the state.

9.17.3.2 items_

`std::unordered_set<Lr0Item> state::items_`

The set of LR(0) items that make up this state.

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

- [backend/state.hpp](#)

9.18 SymbolTable Struct Reference

Stores and manages grammar symbols, including their classification and special markers.

```
#include <symbol_table.hpp>
```

Public Member Functions

- void [PutSymbol](#) (const std::string &identifier, bool isTerminal)
Adds a non-terminal symbol to the symbol table.
- bool [In](#) (const std::string &s) const
Checks if a symbol exists in the symbol table.
- bool [IsTerminal](#) (const std::string &s) const
Checks if a symbol is a terminal.
- bool [IsTerminalWthoEol](#) (const std::string &s) const
Checks if a symbol is a terminal excluding EOL.

Public Attributes

- std::string [EOL_](#) {"\$"}
End-of-line symbol used in parsing, initialized as "\$".
- std::string [EPSILON_](#) {"EPSILON"}
Epsilon symbol, representing empty transitions, initialized as "EPSILON".
- std::unordered_map< std::string, [symbol_type](#) > [st_](#)
Main symbol table, mapping identifiers to a pair of symbol type and its regex.
- std::unordered_set< std::string > [terminals_](#) {[EOL_](#)}
Set of all terminal symbols (including EOL).
- std::unordered_set< std::string > [terminals_wtho_eol_](#) {}
Set of terminal symbols excluding the EOL symbol (\$).
- std::unordered_set< std::string > [non_terminals_](#)
Set of all non-terminal symbols.

9.18.1 Detailed Description

Stores and manages grammar symbols, including their classification and special markers.

This structure holds information about all terminals and non-terminals used in a grammar, as well as special symbols such as EPSILON and the end-of-line marker (\$). It supports symbol classification, membership checks, and filtered views such as terminals excluding \$.

9.18.2.4 PutSymbol()

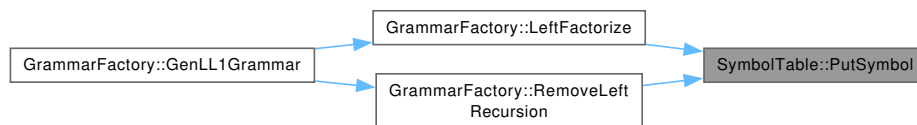
```
void SymbolTable::PutSymbol (
    const std::string & identifier,
    bool isTerminal)
```

Adds a non-terminal symbol to the symbol table.

Parameters

<i>identifier</i>	Name of the symbol.
<i>isTerminal</i>	True if the identifier is a terminal symbol

Here is the caller graph for this function:



9.18.3 Member Data Documentation

9.18.3.1 EOL_

```
std::string SymbolTable::EOL_ {"$"}

```

End-of-line symbol used in parsing, initialized as "\$".

9.18.3.2 EPSILON_

```
std::string SymbolTable::EPSILON_ {"EPSILON"}

```

Epsilon symbol, representing empty transitions, initialized as "EPSILON".

9.18.3.3 non_terminals_

```
std::unordered_set<std::string> SymbolTable::non_terminals_

```

Set of all non-terminal symbols.

9.18.3.4 st_

```
std::unordered_map<std::string, symbol_type> SymbolTable::st_

```

Initial value:

```
{
    {EOL_, symbol_type::TERMINAL}, {EPSILON_, symbol_type::TERMINAL}}
```

Main symbol table, mapping identifiers to a pair of symbol type and its regex.

9.18.3.5 terminals_

```
std::unordered_set<std::string> SymbolTable::terminals_ {EOL_}

```

Set of all terminal symbols (including EOL).

9.18.3.6 terminals_wtho_eol_

```
std::unordered_set<std::string> SymbolTable::terminals_wtho_eol_ {}

```

Set of terminal symbols excluding the EOL symbol (\$).

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

- backend/[symbol_table.hpp](#)
- backend/[symbol_table.cpp](#)

9.19 LLTutorWindow::TreeNode Struct Reference

[TreeNode](#) structure used to build derivation trees.

```
#include <lltutorwindow.h>
```

Public Attributes

- [QString](#) [label](#)
- `std::vector< std::unique_ptr< TreeNode > >` [children](#)

9.19.1 Detailed Description

[TreeNode](#) structure used to build derivation trees.

9.19.2 Member Data Documentation

9.19.2.1 children

```
std::vector<std::unique_ptr<TreeNode> > LLTutorWindow::TreeNode::children
```

9.19.2.2 label

```
QString LLTutorWindow::TreeNode::label
```

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

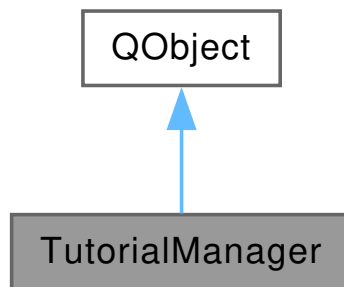
- [gui/lltutorwindow.h](#)

9.20 TutorialManager Class Reference

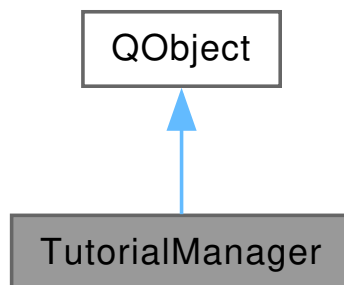
Manages interactive tutorials by highlighting UI elements and guiding the user.

```
#include <tutorialmanager.h>
```

Inheritance diagram for TutorialManager:



Collaboration diagram for TutorialManager:



Public Slots

- void [nextStep](#) ()
Advances to the next tutorial step.

Signals

- void [stepStarted](#) (int index)
Emitted when a new tutorial step starts.
- void [tutorialFinished](#) ()
Emitted when the full tutorial is finished.
- void [ll1Finished](#) ()
Emitted when the LL(1) tutorial ends.
- void [slr1Finished](#) ()
Emitted when the SLR(1) tutorial ends.

Public Member Functions

- [TutorialManager](#) (QWidget *rootWindow)
Constructs a [TutorialManager](#) for a given window.
- void [addStep](#) (QWidget *target, const QString &htmlText)
Adds a new step to the tutorial sequence.
- void [start](#) ()
Starts the tutorial from the beginning.
- void [setRootWindow](#) (QWidget *newRoot)
Sets the root window (used for repositioning the overlay).
- void [clearSteps](#) ()
Clears all steps in the tutorial.
- void [hideOverlay](#) ()
Hides the tutorial overlay immediately.
- void [finishLL1](#) ()
Ends the LL(1) tutorial sequence and emits its corresponding signal.
- void [finishSLR1](#) ()
Ends the SLR(1) tutorial sequence and emits its corresponding signal.

Protected Member Functions

- bool [eventFilter](#) (QObject *obj, QEvent *ev) override
Intercepts UI events to handle overlay behavior.

9.20.1 Detailed Description

Manages interactive tutorials by highlighting UI elements and guiding the user.

This class implements a step-by-step overlay system that visually highlights widgets and shows textual instructions to guide the user through the interface. It supports multiple tutorials (e.g., for LL(1) and SLR(1) modes), with custom steps and signals for tutorial completion.

9.20.2 Constructor & Destructor Documentation

9.20.2.1 TutorialManager()

```
TutorialManager::TutorialManager (
    QWidget * rootWindow)
```

Constructs a [TutorialManager](#) for a given window.

Parameters

<i>rootWindow</i>	The main application window used for relative positioning.
-------------------	--

9.20.3 Member Function Documentation

9.20.3.1 addStep()

```
void TutorialManager::addStep (
    QWidget * target,
    const QString & htmlText)
```

Adds a new step to the tutorial sequence.

Parameters

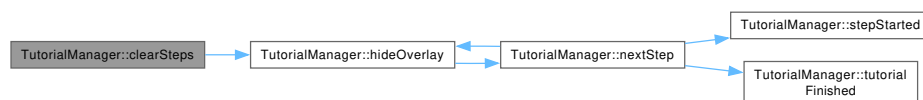
<i>target</i>	The widget to highlight during the step.
<i>htmlText</i>	The instructional HTML message for the step.

9.20.3.2 clearSteps()

```
void TutorialManager::clearSteps ()
```

Clears all steps in the tutorial.

Here is the call graph for this function:



9.20.3.3 eventFilter()

```
bool TutorialManager::eventFilter (
    QObject * obj,
    QEvent * ev) [override], [protected]
```

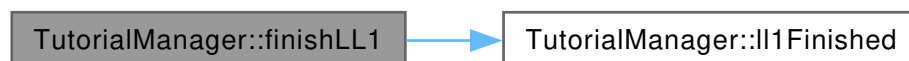
Intercepts UI events to handle overlay behavior.

9.20.3.4 finishLL1()

```
void TutorialManager::finishLL1 ()
```

Ends the LL(1) tutorial sequence and emits its corresponding signal.

Here is the call graph for this function:



9.20.3.5 finishSLR1()

```
void TutorialManager::finishSLR1 ()
```

Ends the SLR(1) tutorial sequence and emits its corresponding signal.

Here is the call graph for this function:



9.20.3.6 hideOverlay()

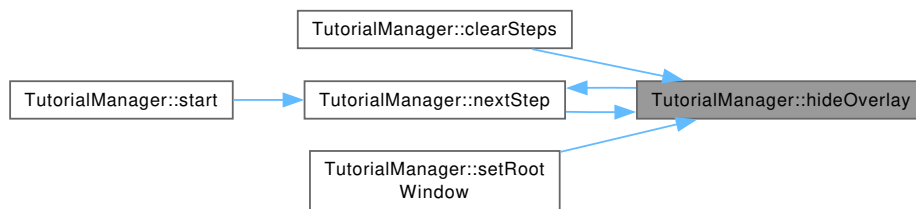
```
void TutorialManager::hideOverlay ()
```

Hides the tutorial overlay immediately.

Here is the call graph for this function:



Here is the caller graph for this function:

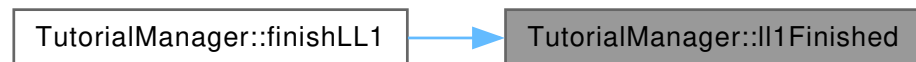


9.20.3.7 ll1Finished

```
void TutorialManager::ll1Finished () [signal]
```

Emitted when the LL(1) tutorial ends.

Here is the caller graph for this function:

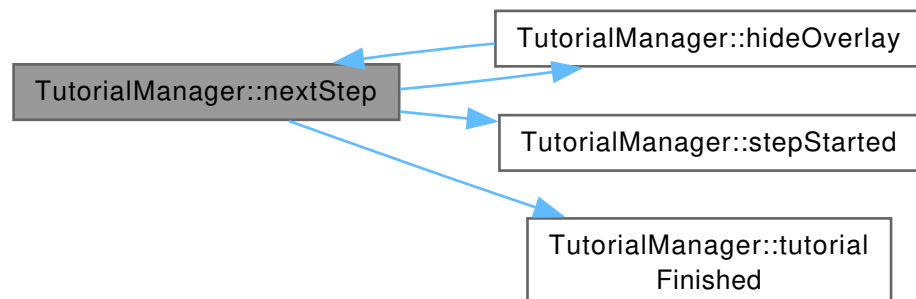


9.20.3.8 nextStep

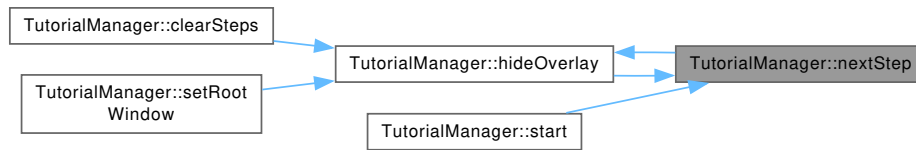
```
void TutorialManager::nextStep () [slot]
```

Advances to the next tutorial step.

Here is the call graph for this function:



Here is the caller graph for this function:



9.20.3.9 setRootWindow()

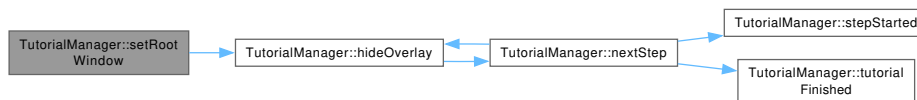
```
void TutorialManager::setRootWindow (
    QWidget * newRoot)
```

Sets the root window (used for repositioning the overlay).

Parameters

<i>newRoot</i>	The new main window to reference.
----------------	-----------------------------------

Here is the call graph for this function:



9.20.3.10 slr1Finished

```
void TutorialManager::slr1Finished () [signal]
```

Emitted when the SLR(1) tutorial ends.

Here is the caller graph for this function:

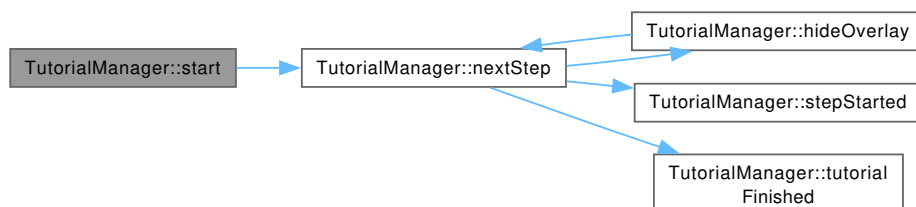


9.20.3.11 start()

```
void TutorialManager::start ()
```

Starts the tutorial from the beginning.

Here is the call graph for this function:



9.20.3.12 stepStarted

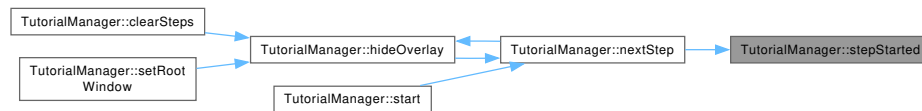
```
void TutorialManager::stepStarted (
    int index) [signal]
```

Emitted when a new tutorial step starts.

Parameters

<i>index</i>	Index of the current step.
--------------	----------------------------

Here is the caller graph for this function:

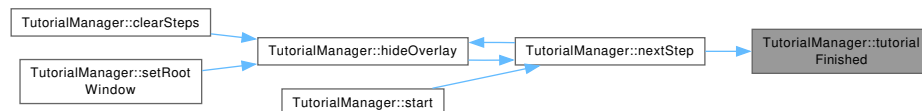


9.20.3.13 tutorialFinished

```
void TutorialManager::tutorialFinished () [signal]
```

Emitted when the full tutorial is finished.

Here is the caller graph for this function:



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

- [widgets/tutorialmanager.h](#)
- [widgets/tutorialmanager.cpp](#)

9.21 TutorialStep Struct Reference

Represents a single step in the tutorial sequence.

```
#include <tutorialmanager.h>
```

Public Attributes

- `QWidget *` [target](#)
Widget to highlight during the tutorial step.
- `QString` [htmlText](#)
HTML text to show as instruction or explanation.

9.21.1 Detailed Description

Represents a single step in the tutorial sequence.

Each step highlights a target widget and displays an associated HTML-formatted message.

9.21.2 Member Data Documentation

9.21.2.1 htmlText

```
QString TutorialStep::htmlText
```

HTML text to show as instruction or explanation.

9.21.2.2 target

`QWidget* TutorialStep::target`

Widget to highlight during the tutorial step.

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

- [widgets/tutorialmanager.h](#)

9.22 UniqueQueue< T > Class Template Reference

A queue that ensures each element is inserted only once.

```
#include <UniqueQueue.h>
```

Public Member Functions

- void [push](#) (const T &value)
Pushes an element to the queue if it hasn't been inserted before.
- void [pop](#) ()
Removes the front element from the queue.
- const T & [front](#) () const
Accesses the front element of the queue.
- bool [empty](#) () const
Checks whether the queue is empty.
- void [clear](#) ()
Clears the queue and the set of seen elements.

9.22.1 Detailed Description

```
template<typename T>
```

```
class UniqueQueue< T >
```

A queue that ensures each element is inserted only once.

This data structure behaves like a standard FIFO queue but prevents duplicate insertions.

Internally, it uses a `std::queue` for ordering and a `std::unordered_set` to track seen elements.

Template Parameters

<i>T</i>	The type of elements stored in the queue. Must be hashable and comparable.
----------	--

9.22.2 Member Function Documentation

9.22.2.1 clear()

```
template<typename T>
```

```
void UniqueQueue< T >::clear () [inline]
```

Clears the queue and the set of seen elements.

9.22.2.2 empty()

```
template<typename T>
```

```
bool UniqueQueue< T >::empty () const [inline]
```

Checks whether the queue is empty.

Returns

true if the queue is empty; false otherwise.

9.22.2.3 front()

```
template<typename T>
const T & UniqueQueue< T >::front () const [inline]
```

Accesses the front element of the queue.

Returns

A reference to the front element.

9.22.2.4 pop()

```
template<typename T>
void UniqueQueue< T >::pop () [inline]
```

Removes the front element from the queue.

9.22.2.5 push()

```
template<typename T>
void UniqueQueue< T >::push (
    const T & value) [inline]
```

Pushes an element to the queue if it hasn't been inserted before.

Parameters

<i>value</i>	The element to insert.
--------------	------------------------

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

- [gui/UniqueQueue.h](#)

Chapter 10

File Documentation

10.1 /home/jose/Documentos/UMA/TFG/SyntaxTutor/README.md File Reference

10.2 app/main.cpp File Reference

```
#include "appversion.h"
#include "mainwindow.h"
#include <QApplication>
#include <QFont>
#include <QFontDatabase>
#include <QImageReader>
#include <QSettings>
#include <QTranslator>
```

Include dependency graph for main.cpp:



Functions

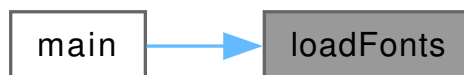
- void [loadFonts](#) ()
- int [main](#) (int argc, char *argv[])

10.2.1 Function Documentation

10.2.1.1 loadFonts()

```
void loadFonts ()
```

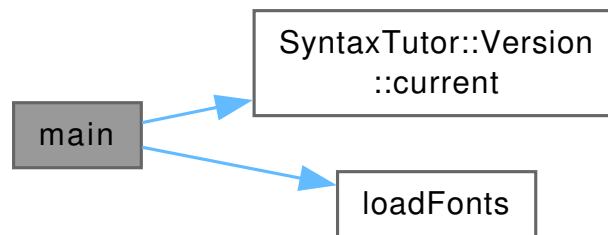
Here is the caller graph for this function:



10.2.1.2 main()

```
int main (  
    int argc,  
    char * argv[])
```

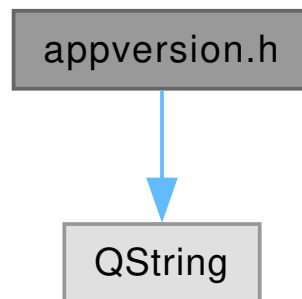
Here is the call graph for this function:



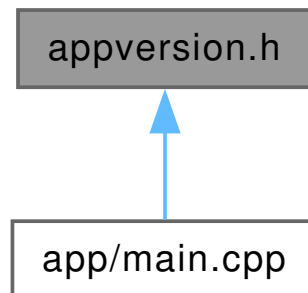
10.3 appversion.h File Reference

```
#include <QString>
```

Include dependency graph for appversion.h:



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



Namespaces

- namespace [SyntaxTutor](#)
- namespace [SyntaxTutor::Version](#)

Functions

- [QString SyntaxTutor::Version::current \(\)](#)
- [const char * SyntaxTutor::Version::raw \(\)](#)

10.4 appversion.h

[Go to the documentation of this file.](#)

```
00001 #ifndef APPVERSION_H
```

```

00002 #define APPVERSION_H
00003
00004 #include <QString>
00005
00006 namespace SyntaxTutor::Version {
00007     inline QString current() {
00008         return QStringLiteral("1.0.4");
00009     }
00010
00011     inline const char* raw() {
00012         return "1.0.4";
00013     }
00014 } // namespace SyntaxTutor::Version
00015 #endif // APPVERSION_H

```

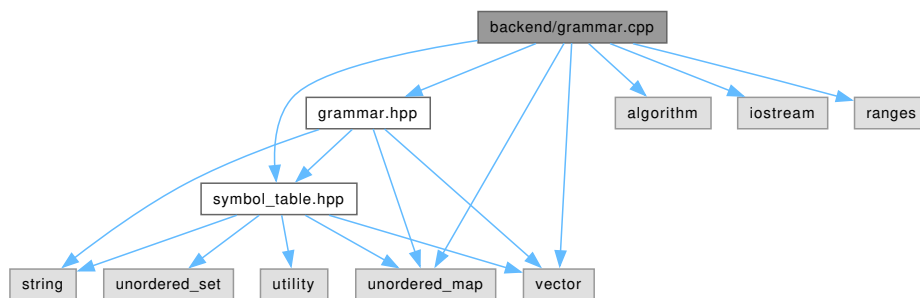
10.5 backend/grammar.cpp File Reference

```

#include "grammar.hpp"
#include "symbol_table.hpp"
#include <algorithm>
#include <iostream>
#include <ranges>
#include <unordered_map>
#include <vector>

```

Include dependency graph for grammar.cpp:



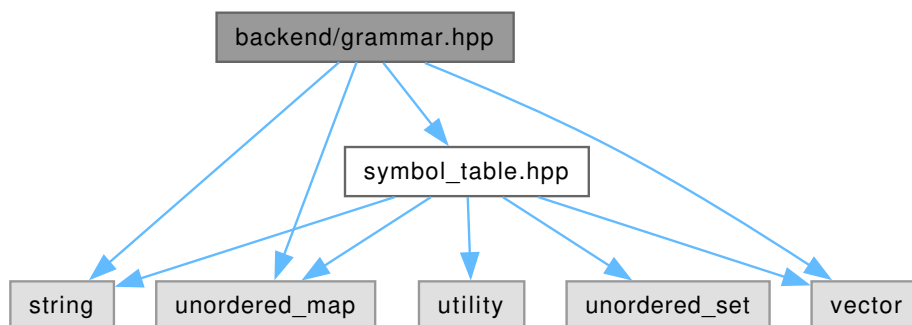
10.6 backend/grammar.hpp File Reference

```

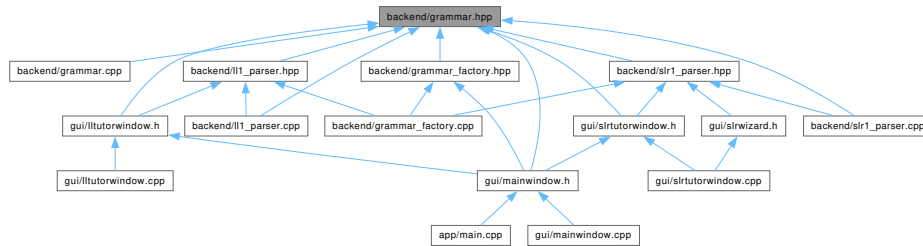
#include "symbol_table.hpp"
#include <string>
#include <unordered_map>
#include <vector>

```

Include dependency graph for grammar.hpp:



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



Classes

- struct [Grammar](#)

Represents a context-free grammar, including its rules, symbol table, and starting symbol.

Typedefs

- using [production](#) = `std::vector<std::string>`

Represents the right-hand side of a grammar rule.

10.6.1 Typedef Documentation

10.6.1.1 production

using [production](#) = `std::vector<std::string>`

Represents the right-hand side of a grammar rule.

A production is a sequence of grammar symbols (terminals or non-terminals) that can be derived from a non-terminal symbol in the grammar.

For example, in the rule $A \rightarrow a B c$, the production would be: {"a", "B", "c"}

10.7 grammar.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018 #pragma once
00019 #include "symbol_table.hpp"
00020 #include <string>
00021 #include <unordered_map>
00022 #include <vector>
00023
00034 using production = std::vector<std::string>;
00035
00046 struct Grammar {
00047
00048     Grammar();
00049     explicit Grammar(
00050         const std::unordered_map<std::string, std::vector<production>&
00051             grammar);
00052
00061     void SetAxiom(const std::string& axiom);
```

```

00062
00073     bool HasEmptyProduction(const std::string& antecedent) const;
00074
00086     std::vector<std::pair<const std::string, production>>
00087     FilterRulesByConsequent(const std::string& arg) const;
00088
00095     void Debug() const; // NOSONAR
00096
00109     void AddProduction(const std::string& antecedent,
00110                       const std::vector<std::string>& consequent);
00111
00123     std::vector<std::string> Split(const std::string& s);
00124
00129     std::unordered_map<std::string, std::vector<production>> g_;
00130
00134     std::string axiom_;
00135
00139     SymbolTable st_;
00140 };

```

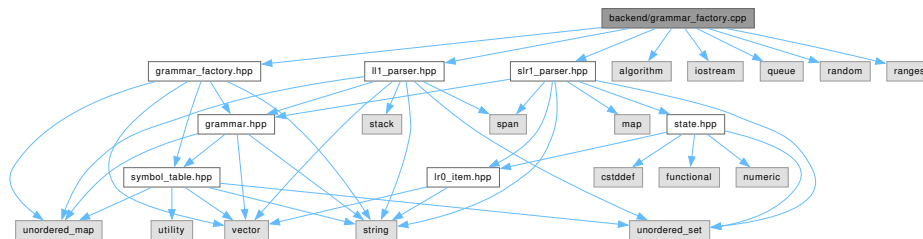
10.8 backend/grammar_factory.cpp File Reference

```

#include "grammar_factory.hpp"
#include "l1l_parser.hpp"
#include "slr1_parser.hpp"
#include <algorithm>
#include <iostream>
#include <queue>
#include <random>
#include <ranges>

```

Include dependency graph for grammar_factory.cpp:



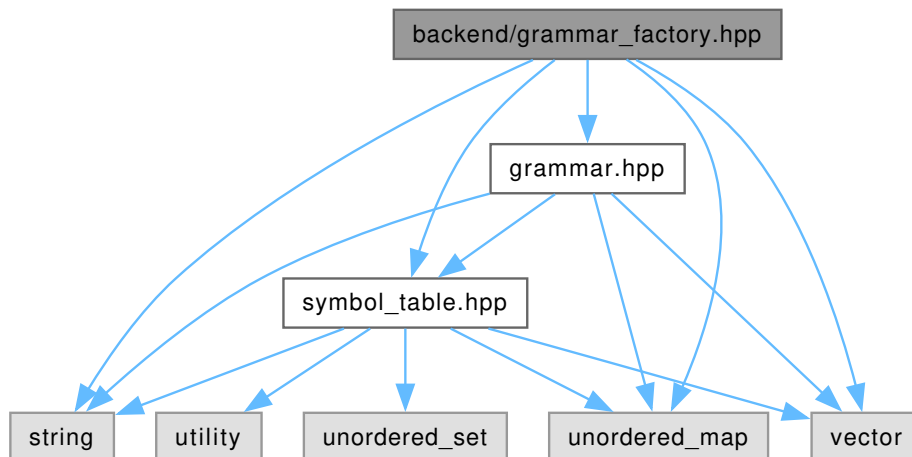
10.9 backend/grammar_factory.hpp File Reference

```

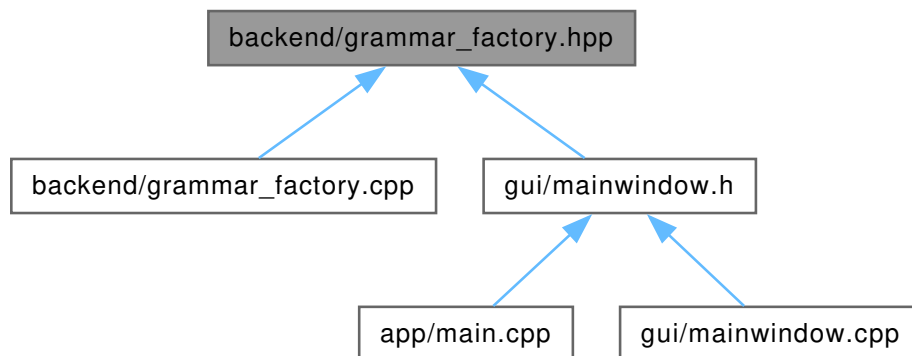
#include "grammar.hpp"
#include "symbol_table.hpp"
#include <string>
#include <unordered_map>
#include <vector>

```

Include dependency graph for grammar_factory.hpp:



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



Classes

- struct [GrammarFactory](#)
Responsible for creating and managing grammar items and performing checks on grammars.
- struct [GrammarFactory::FactoryItem](#)
Represents an individual grammar item with its associated symbol table.

10.10 grammar_factory.hpp

[Go to the documentation of this file.](#)

```

00001  /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019  #pragma once

```



```

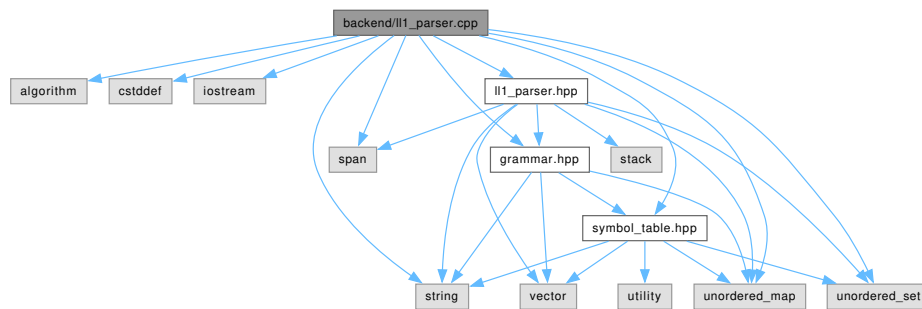
00020 #include "grammar.hpp"
00021 #include "symbol_table.hpp"
00022 #include <string>
00023 #include <unordered_map>
00024 #include <vector>
00025
00031 struct GrammarFactory {
00032
00038     struct FactoryItem {
00043         std::unordered_map<std::string, std::vector<production>> g_;
00044
00048         SymbolTable st_;
00049
00055         explicit FactoryItem(
00056             const std::unordered_map<std::string, std::vector<production>>&
00057             grammar);
00058     };
00059
00064     void Init();
00065
00072     Grammar PickOne(int level);
00073
00080     Grammar GenLL1Grammar(int level);
00087     Grammar GenSLR1Grammar(int level);
00088
00093     Grammar Lv1();
00094
00099     Grammar Lv2();
00100
00106     Grammar Lv3();
00107
00118     Grammar Lv4();
00119
00130     Grammar Lv5();
00131
00142     Grammar Lv6();
00143
00154     Grammar Lv7();
00155
00164     FactoryItem CreateLv2Item();
00165
00172     bool HasUnreachableSymbols(Grammar& grammar) const;
00173
00189     bool IsInfinite(Grammar& grammar) const;
00190
00197     bool HasDirectLeftRecursion(const Grammar& grammar) const;
00198
00204     bool HasIndirectLeftRecursion(const Grammar& grammar) const;
00205
00211     bool HasCycle(
00212         const std::unordered_map<std::string, std::unordered_set<std::string>&
00213         graph) const;
00214
00220     std::unordered_set<std::string>
00221     NullableSymbols(const Grammar& grammar) const;
00222
00242     void RemoveLeftRecursion(Grammar& grammar);
00244
00266     void LeftFactorize(Grammar& grammar);
00268
00280     std::vector<std::string>
00281     LongestCommonPrefix(const std::vector<production>& productions);
00282
00296     bool StartsWith(const production& prod,
00297         const std::vector<std::string>& prefix);
00298
00311     std::string GenerateNewNonTerminal(Grammar& grammar,
00312         const std::string& base);
00313
00326     void NormalizeNonTerminals(FactoryItem& item, const std::string& nt) const;
00327
00340     void AdjustTerminals(FactoryItem& base, const FactoryItem& cmb,
00341         const std::string& target_nt) const;
00342
00356     std::unordered_map<std::string, std::vector<production>>
00357     Merge(const FactoryItem& base, const FactoryItem& cmb) const;
00358
00363     std::vector<FactoryItem> items;
00364
00368     std::vector<std::string> terminal_alphabet_{ "a", "b", "c", "d", "e", "f",
00369         "g", "h", "i", "j", "k", "l" };
00370
00374     std::vector<std::string> non_terminal_alphabet_{ "A", "B", "C", "D",
00375         "E", "F", "G" };
00376 };

```

10.11 backend/ll1_parser.cpp File Reference

```
#include <algorithm>
#include <cstdint>
#include <iostream>
#include <span>
#include <string>
#include <unordered_map>
#include <unordered_set>
#include "grammar.hpp"
#include "ll1_parser.hpp"
#include "symbol_table.hpp"
```

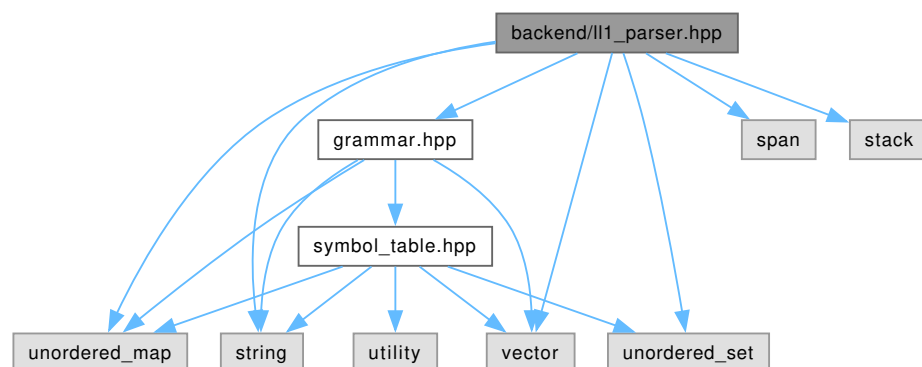
Include dependency graph for ll1_parser.cpp:



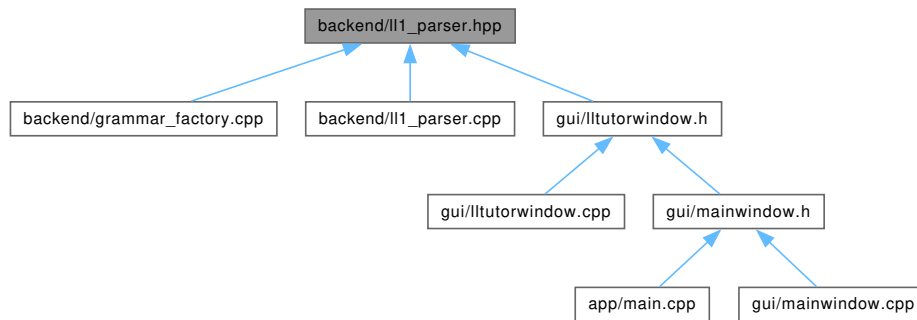
10.12 backend/ll1_parser.hpp File Reference

```
#include "grammar.hpp"
#include <span>
#include <stack>
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <vector>
```

Include dependency graph for ll1_parser.hpp:



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



Classes

- class **LL1Parser**

10.13 ll1_parser.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #pragma once
00020 #include "grammar.hpp"
00021 #include <span>
00022 #include <stack>
00023 #include <string>
00024 #include <unordered_map>
00025 #include <unordered_set>
00026 #include <vector>
00027
00028 class LL1Parser {
00029
00030     using l1l_table = std::unordered_map<
00031         std::string, std::unordered_map<std::string, std::vector<production>>>;
00032
00033 public:
00034     LL1Parser() = default;
00035     explicit LL1Parser(Grammar gr);
00036
00037     bool CreateLL1Table();
00038
00039     void First(std::span<const std::string> rule,
00040         std::unordered_set<std::string>& result);
00041
00042     void ComputeFirstSets();
00043
00044     void ComputeFollowSets();
00045
00046     std::unordered_set<std::string> Follow(const std::string& arg);
00047
00048     std::unordered_set<std::string>
00049     PredictionSymbols(const std::string& antecedent,
00050         const std::vector<std::string>& consequent);
00051
00052     l1l_table l1l_t_;
00053
00054     Grammar gr_;
00055

```

```

00195
00197     std::unordered_map<std::string, std::unordered_set<std::string>
00198         first_sets_;
00199
00201     std::unordered_map<std::string, std::unordered_set<std::string>
00202         follow_sets_;
00203 };

```

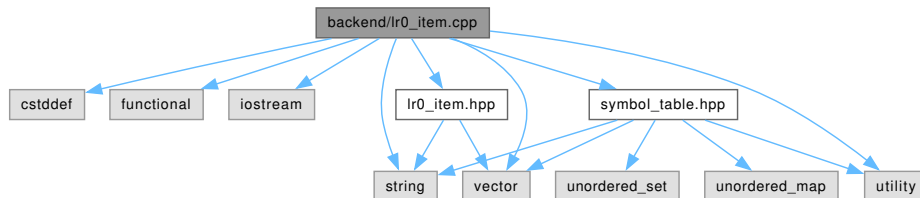
10.14 backend/lr0_item.cpp File Reference

```

#include <cstdint>
#include <functional>
#include <iostream>
#include <string>
#include <utility>
#include <vector>
#include "lr0_item.hpp"
#include "symbol_table.hpp"

```

Include dependency graph for lr0_item.cpp:



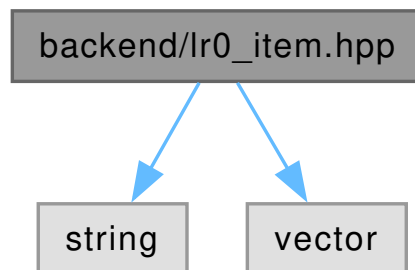
10.15 backend/lr0_item.hpp File Reference

```

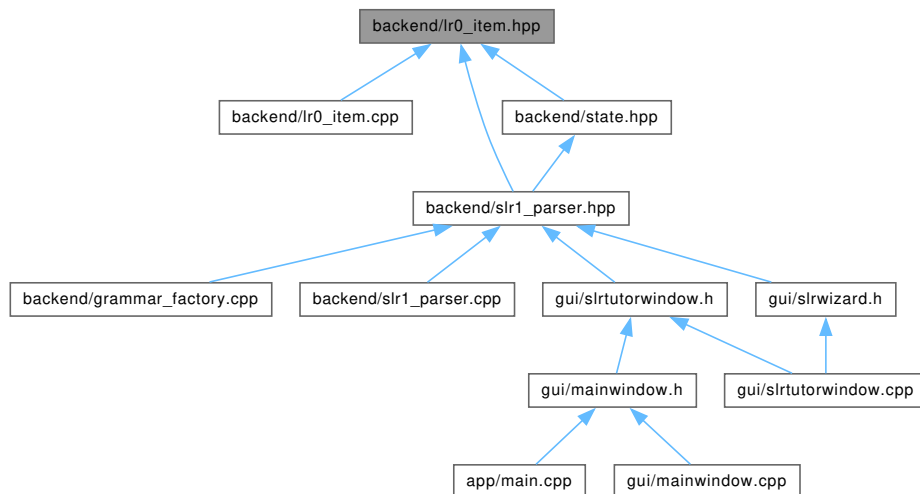
#include <string>
#include <vector>

```

Include dependency graph for lr0_item.hpp:



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



Classes

- struct [Lr0Item](#)

Represents an LR(0) item used in LR automata construction.

10.16 lr0_item.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #pragma once
00020 #include <string>
00021 #include <vector>
00022
00034
00035 struct Lr0Item {
00039     std::string antecedent_;
00040
00044     std::vector<std::string> consequent_;
00045
00049     std::string epsilon_;
00050
00054     std::string eol_;
00055
00059     unsigned int dot_ = 0;
00060
00068     Lr0Item(std::string antecedent, std::vector<std::string> consequent,
00069             std::string epsilon, std::string eol);
00070
00079     Lr0Item(std::string antecedent, std::vector<std::string> consequent,
00080             unsigned int dot, std::string epsilon, std::string eol);
00081
00087     std::string NextToDot() const;
00088
00093     void PrintItem() const;
00094

```

```

00100     std::string ToString() const;
00101
00105     void AdvanceDot();
00106
00111     bool IsComplete() const;
00112
00118     bool operator==(const Lr0Item& other) const;
00119 };
00120
00121 namespace std {
00122 template <> struct hash<Lr0Item> {
00123     size_t operator()(const Lr0Item& item) const;
00124 };
00125 } // namespace std

```

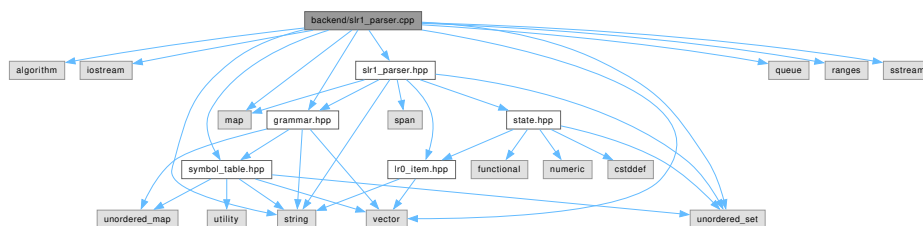
10.17 backend/slr1_parser.cpp File Reference

```

#include <algorithm>
#include <iostream>
#include <map>
#include <queue>
#include <ranges>
#include <sstream>
#include <string>
#include <unordered_set>
#include <vector>
#include "grammar.hpp"
#include "slr1_parser.hpp"
#include "symbol_table.hpp"

```

Include dependency graph for slr1_parser.cpp:



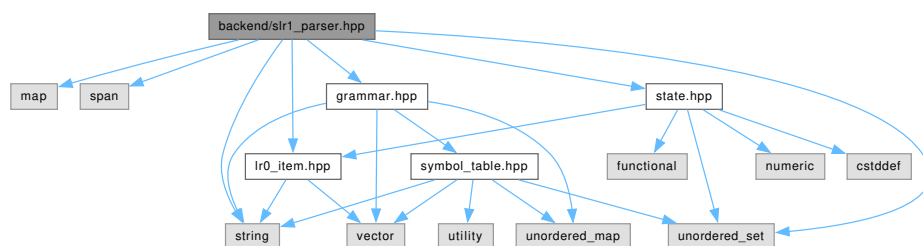
10.18 backend/slr1_parser.hpp File Reference

```

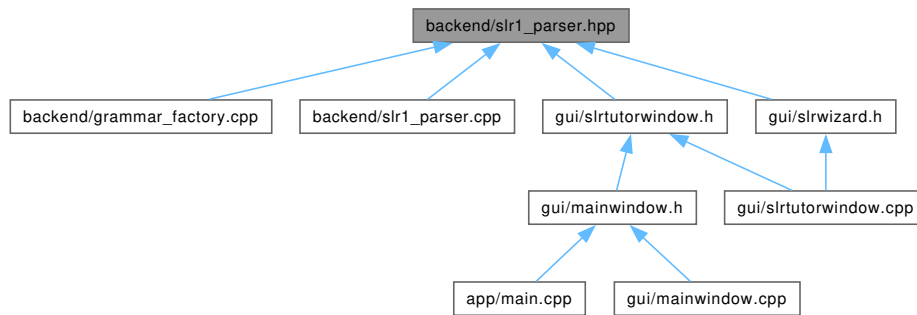
#include <map>
#include <span>
#include <string>
#include <unordered_set>
#include "grammar.hpp"
#include "lr0_item.hpp"
#include "state.hpp"

```

Include dependency graph for slr1_parser.hpp:



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



Classes

- class [SLR1Parser](#)
Implements an SLR(1) parser for context-free grammars.
- struct [SLR1Parser::s_action](#)

10.19 slr1_parser.hpp

[Go to the documentation of this file.](#)

```

00001  /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #pragma once
00020 #include <map>
00021 #include <span>
00022 #include <string>
00023 #include <unordered_set>
00024
00025 #include "grammar.hpp"
00026 #include "lr0_item.hpp"
00027 #include "state.hpp"
00028
00038 class SLR1Parser {
00039 public:
00051     enum class Action { Shift, Reduce, Accept, Empty };
00052
00064     struct s_action {
00065         const Lr0Item* item;
00066         Action          action;
00067     };
00068
00081     using action_table =
00082         std::map<unsigned int, std::map<std::string, SLR1Parser::s_action>;
00083
00096     using transition_table =
00097         std::map<unsigned int, std::map<std::string, unsigned int>;
00098
00099     SLR1Parser() = default;
00100     explicit SLR1Parser(Grammar gr);
00101
00110     std::unordered_set<Lr0Item> AllItems() const;
00111
00122     void Closure(std::unordered_set<Lr0Item>& items);
00123
00137     void ClosureUtil(std::unordered_set<Lr0Item>& items, unsigned int size,

```

```

00138         std::unordered_set<std::string>& visited);
00139
00150     std::unordered_set<Lr0Item> Delta(const std::unordered_set<Lr0Item>& items,
00151                                     const std::string& str);
00152
00165     bool SolveLRConflicts(const state& st);
00166
00193     void First(std::span<const std::string> rule,
00194               std::unordered_set<std::string>& result);
00195
00206     void ComputeFirstSets();
00207
00233     void ComputeFollowSets();
00234
00249     std::unordered_set<std::string> Follow(const std::string& arg);
00250
00263     void MakeInitialState();
00264
00282     bool MakeParser();
00283
00293     std::string PrintItems(const std::unordered_set<Lr0Item>& items) const;
00294
00296     Grammar gr_;
00297
00299     std::unordered_map<std::string, std::unordered_set<std::string>
00300                       first_sets_;
00301
00303     std::unordered_map<std::string, std::unordered_set<std::string>
00304                       follow_sets_;
00305
00308     action_table actions_;
00309
00312     transition_table transitions_;
00313
00315     std::unordered_set<state> states_;
00316 };

```

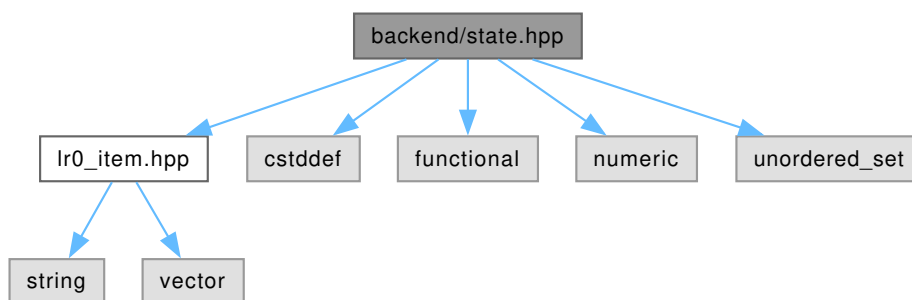
10.20 backend/state.hpp File Reference

```

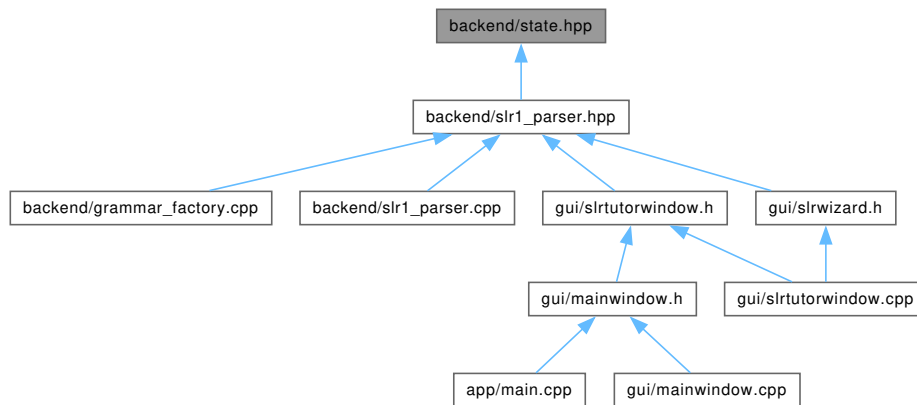
#include "lr0_item.hpp"
#include <cstddef>
#include <functional>
#include <numeric>
#include <unordered_set>

```

Include dependency graph for state.hpp:



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



Classes

- struct [state](#)

Represents a state in the LR(0) automaton.

10.21 state.hpp

[Go to the documentation of this file.](#)

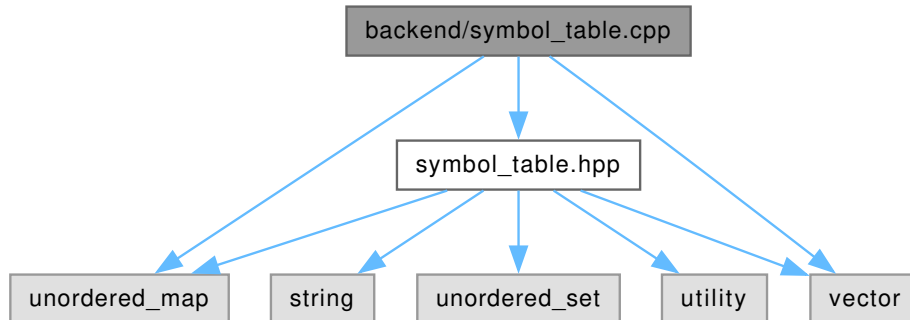
```

00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #pragma once
00020 #include "lr0_item.hpp"
00021 #include <cstdlib>
00022 #include <functional>
00023 #include <numeric>
00024 #include <unordered_set>
00025
00033 struct state {
00037     std::unordered_set<Lr0Item> items_;
00038
00042     unsigned int id_;
00043
00049     bool operator==(const state& other) const { return other.items_ == items_; }
00050 };
00051
00052 namespace std {
00053     template <> struct hash<state> {
00054         size_t operator()(const state& st) const {
00055             size_t seed =
00056                 std::accumulate(st.items_.begin(), st.items_.end(), 0,
00057                     [](size_t acc, const Lr0Item& item) {
00058                         return acc ^ (std::hash<Lr0Item>()(item));
00059                     });
00060             return seed;
00061         }
00062     };
00063 } // namespace std
  
```

10.22 backend/symbol_table.cpp File Reference

```
#include "symbol_table.hpp"
#include <unordered_map>
#include <vector>
```

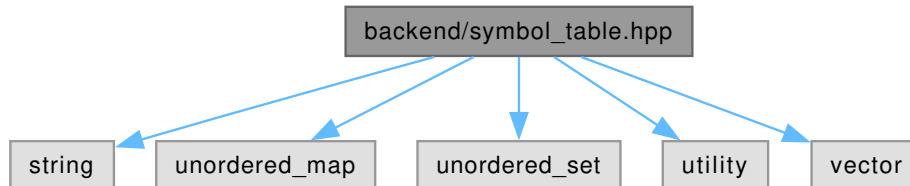
Include dependency graph for symbol_table.cpp:



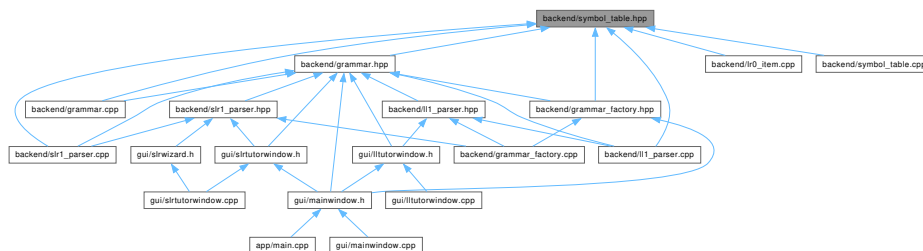
10.23 backend/symbol_table.hpp File Reference

```
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <utility>
#include <vector>
```

Include dependency graph for symbol_table.hpp:



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



Classes

- struct [SymbolTable](#)

Stores and manages grammar symbols, including their classification and special markers.

Enumerations

- enum class [symbol_type](#) { `NO_TERMINAL` , `TERMINAL` }

Represents the type of a grammar symbol.

10.23.1 Enumeration Type Documentation

10.23.1.1 symbol_type

enum class `symbol_type` [strong]

Represents the type of a grammar symbol.

This enum distinguishes between terminal and non-terminal symbols within the grammar and the symbol table.

Enumerator

NO_TERMINAL	
TERMINAL	

10.24 symbol_table.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #pragma once
00020 #include <string>
00021 #include <unordered_map>
00022 #include <unordered_set>
00023 #include <utility>
00024 #include <vector>
00025
00033 enum class symbol_type { NO_TERMINAL, TERMINAL };
00034
00045 struct SymbolTable {
00047     std::string EOL_{"$"};
00048
00051     std::string EPSILON_{"EPSILON"};
00052
00055     std::unordered_map<std::string, symbol_type> st_{
00056         {EOL_, symbol_type::TERMINAL}, {EPSILON_, symbol_type::TERMINAL}};
00057
00061     std::unordered_set<std::string> terminals_{EOL_};
00062
00066     std::unordered_set<std::string> terminals_wtho_eol_{};
00067
00071     std::unordered_set<std::string> non_terminals_;
00072
00079     void PutSymbol(const std::string& identifier, bool isTerminal);
00080
00087     bool In(const std::string& s) const;
00088
00095     bool IsTerminal(const std::string& s) const;
00096
00103     bool IsTerminalWthoEol(const std::string& s) const;
00104 };

```

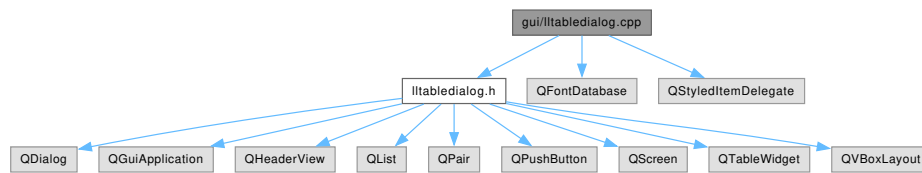
10.25 gui/ltabledialog.cpp File Reference

```

#include "ltabledialog.h"
#include <QFontDatabase>
#include <QStyledItemDelegate>

```

Include dependency graph for `lltabledialog.cpp`:



Classes

- class [CenterAlignDelegate](#)

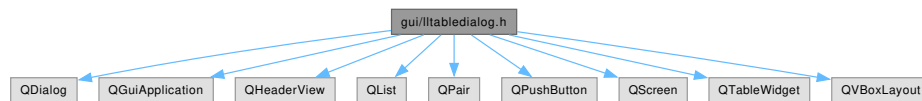
10.26 gui/lltabledialog.h File Reference

```

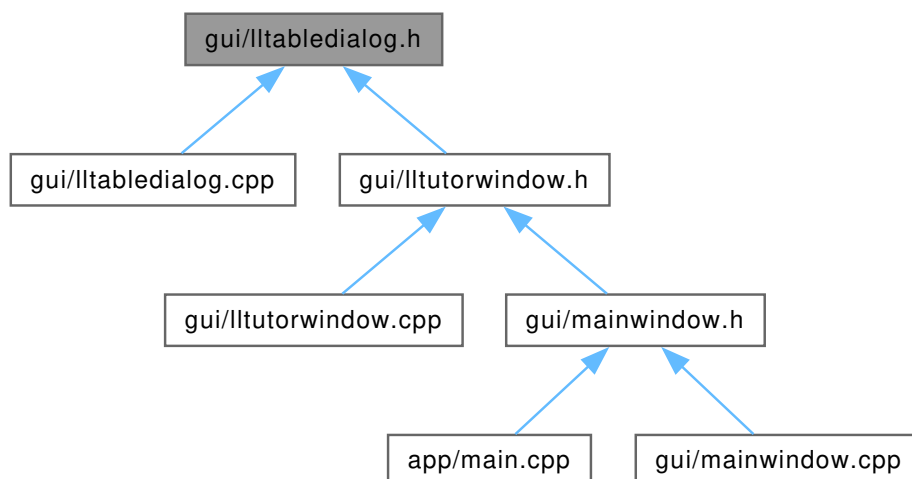
#include <QDialog>
#include <QGuiApplication>
#include <QHeaderView>
#include <QList>
#include <QPair>
#include <QPushButton>
#include <QScreen>
#include <QTableWidget>
#include <QVBoxLayout>

```

Include dependency graph for `lltabledialog.h`:



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



Classes

- class [LLTableDialog](#)
Dialog for filling and submitting an LL(1) parsing table.

10.27 lltabledialog.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #ifndef LLTABLEDIALOG_H
00020 #define LLTABLEDIALOG_H
00021
00022 #include <QDialog>
00023 #include <QGuiApplication>
00024 #include <QHeaderView>
00025 #include <QList>
00026 #include <QPair>
00027 #include <QPushButton>
00028 #include <QScreen>
00029 #include <QTableWidget>
00030 #include <QVBoxLayout>
00031
00041 class LLTableDialog : public QDialog {
00042     Q_OBJECT
00043     public:
00052     LLTableDialog(const QStringList& rowHeaders, const QStringList& colHeaders,
00053                  QWidget* parent,
00054                  QVector<QVector<QString>*> initialData = nullptr);
00055
00060     QVector<QVector<QString>> getTableData() const;
00061
00070     void setInitialData(const QVector<QVector<QString>>& data);
00071
00076     void highlightIncorrectCells(const QList<QPair<int, int>>& coords);
00077
00078     signals:
00083     void submitted(const QVector<QVector<QString>>& data);
00084
00085     private:
00086     QTableWidget* table;
00087     QPushButton* submitButton;
00088 };
00089
00090 #endif // LLTABLEDIALOG_H

```

10.28 gui/lltutorwindow.cpp File Reference

```

#include "lltutorwindow.h"
#include "tutorialmanager.h"
#include "ui_lltutorwindow.h"
#include <QAbstractButton>
#include <QFontDatabase>
#include <QRandomGenerator>
#include <QRegularExpression>
#include <QWheelEvent>

```

Include dependency graph for lltutorwindow.cpp:



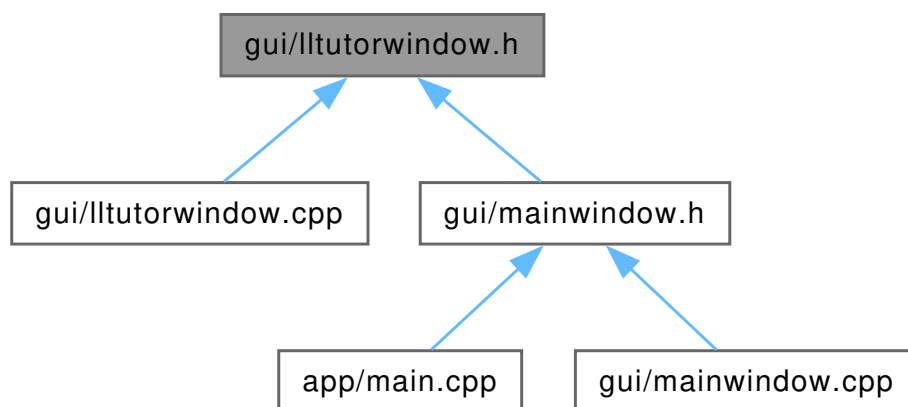
10.29 gui/lltutorwindow.h File Reference

```
#include <QAbstractItemView>
#include <QDialog>
#include <QFileDialog>
#include <QGraphicsColorizeEffect>
#include <QGraphicsScene>
#include <QGraphicsTextItem>
#include <QGraphicsView>
#include <QListWidgetItem>
#include <QMainWindow>
#include <QMessageBox>
#include <QPainter>
#include <QPropertyAnimation>
#include <QPushButton>
#include <QScrollBar>
#include <QShortcut>
#include <QTableWidget>
#include <QTextDocument>
#include <QTextEdit>
#include <QTime>
#include <QTimer>
#include <QTreeWidgetItem>
#include <QVBoxLayout>
#include <QtPrintSupport/QtPrinter>
#include "grammar.hpp"
#include "ll1_parser.hpp"
#include "lltabledialog.h"
```

Include dependency graph for lltutorwindow.h:



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



Classes

- class [LLTutorWindow](#)
Main window for the LL(1) interactive tutoring mode in [SyntaxTutor](#).
- struct [LLTutorWindow::TreeNode](#)
[TreeNode](#) structure used to build derivation trees.

Enumerations

- enum class [State](#) {
[A](#) , [A1](#) , [A2](#) , [A_prime](#) ,
[B](#) , [B1](#) , [B2](#) , [B_prime](#) ,
[C](#) , [C_prime](#) , [fin](#) }

10.29.1 Enumeration Type Documentation

10.29.1.1 State

```
enum class State [strong]
```

Enumerator

A	
A1	
A2	
A_prime	
B	
B1	
B2	
B_prime	
C	
C_prime	
fin	

10.30 lltutorwindow.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #ifndef LLTUTORWINDOW_H
00020 #define LLTUTORWINDOW_H
00021
00022 #include <QAbstractItemView>
00023 #include <QDialog>
00024 #include <QFileDialog>
00025 #include <QGraphicsColorizeEffect>
00026 #include <QGraphicsScene>
00027 #include <QGraphicsTextItem>
00028 #include <QGraphicsView>
00029 #include <QListWidgetItem>
00030 #include <QMainWindow>
00031 #include <QMessageBox>
00032 #include <QPainter>
00033 #include <QPropertyAnimation>
00034 #include <QPushButton>
00035 #include <QScrollBar>
00036 #include <QShortcut>
```

```

00037 #include <QTableWidget>
00038 #include <QTextDocument>
00039 #include <QTextEdit>
00040 #include <QTime>
00041 #include <QTimer>
00042 #include <QTreeWidgetItem>
00043 #include <QVBoxLayout>
00044 #include <QtPrintSupport/QPrinter>
00045
00046 #include "grammar.hpp"
00047 #include "ll1_parser.hpp"
00048 #include "ll1tabledialog.h"
00049
00050 class TutorialManager;
00051
00052 namespace Ui {
00053 class LLTutorWindow;
00054 }
00055
00056 // ===== LL(1) Tutor States =====
00057 enum class State { A, A1, A2, A_prime, B, B1, B2, B_prime, C, C_prime, fin };
00058
00059 // ===== LL(1) Tutor Main Class =====
00079 class LLTutorWindow : public QMainWindow {
00080     Q_OBJECT
00081
00082 public:
00083     // ===== Derivation Tree (used in TeachFirst) =====
00084     struct TreeNode {
00085         QString label;
00086         std::vector<std::unique_ptr<TreeNode>> children;
00087     };
00088
00092     // ===== Constructor / Destructor =====
00099     explicit LLTutorWindow(const Grammar& grammar,
00100         TutorialManager* tm = nullptr,
00101         QWidget* parent = nullptr);
00102     ~LLTutorWindow();
00103
00104     // ===== State Machine & Question Logic =====
00109     QString generateQuestion();
00110
00115     void updateState(bool isCorrect);
00116
00122     QString FormatGrammar(const Grammar& grammar);
00123
00124     // ===== UI Interaction =====
00125     void addMessage(const QString& text,
00126         bool isUser);
00127     void addWidgetMessage(QWidget* widget);
00128     void
00129     exportConversationToPdf(const QString& filePath);
00130     void showTable();
00131     void showTableForCPrime();
00132     void updateProgressPanel(); // Update progress panel
00133
00134     // ===== Visual Feedback / Animations =====
00135     void animateLabelPop(QLabel* label);
00136     void animateLabelColor(QLabel* label, const QColor& flashColor);
00137     void wrongAnimation();
00138     void
00139     wrongUserResponseAnimation();
00140     void markLastUserIncorrect();
00141
00142     // ===== Tree Generation (TeachFirst mode) =====
00143     void TeachFirstTree(const std::vector<std::string>& symbols,
00144         std::unordered_set<std::string>& first_set, int depth,
00145         std::unordered_set<std::string>& processing,
00146         QTreeWidgetItem* parent);
00147
00148     std::unique_ptr<TreeNode>
00149     buildTreeNode(const std::vector<std::string>& symbols,
00150         std::unordered_set<std::string>& first_set, int depth,
00151         std::vector<std::pair<std::string, std::vector<std::string>>>&
00152         active_derivations);
00153
00154     int computeSubtreeWidth(const std::unique_ptr<TreeNode>& node,
00155         int hSpacing);
00156     void drawTree(const std::unique_ptr<TreeNode>& root, QGraphicsScene* scene,
00157         QPointF pos, int hSpacing, int vSpacing);
00158
00159     void showTreeGraphics(
00160         std::unique_ptr<TreeNode> root); // Display derivation tree visually
00161
00162     // ===== User Response Verification =====
00163     bool verifyResponse(const QString& userResponse); // Delegates to current
00164         // state's verification

```



```

00165     bool verifyResponseForA(const QString& userResponse);
00166     bool verifyResponseForA1(const QString& userResponse);
00167     bool verifyResponseForA2(const QString& userResponse);
00168     bool verifyResponseForB(const QString& userResponse);
00169     bool verifyResponseForB1(const QString& userResponse);
00170     bool verifyResponseForB2(const QString& userResponse);
00171     bool verifyResponseForC(); // C is non-textual (checks internal table)
00172
00173     // ===== Expected Solutions (Auto-generated) =====
00174     QString      solution(const std::string& state);
00175     QStringList  solutionForA();
00176     QString      solutionForA1();
00177     QString      solutionForA2();
00178     QSet<QString> solutionForB();
00179     QSet<QString> solutionForB1();
00180     QSet<QString> solutionForB2();
00181
00182     // ===== Feedback (Corrective Explanations) =====
00183     QString feedback(); // Delegates by state
00184     QString feedbackForA();
00185     QString feedbackForA1();
00186     QString feedbackForA2();
00187     QString feedbackForAPrime();
00188     QString feedbackForB();
00189     QString feedbackForB1();
00190     QString feedbackForB2();
00191     QString feedbackForBPrime();
00192     QString feedbackForC();
00193     QString feedbackForCPrime();
00194     void feedbackForB1TreeWidget(); // TreeWidget of Teach (LL1 TeachFirst)
00195     void feedbackForB1TreeGraphics(); // Show derivation tree
00196     QString TeachFollow(const QString& nt);
00197     QString TeachPredictionSymbols(const QString& ant,
00198                                   const production& conseq);
00199     QString TeachLL1Table();
00200
00201     void handleTableSubmission(const QVector<QVector<QString>& raw,
00202                               const QStringList& colHeaders);
00203 private slots:
00204     void on_confirmButton_clicked();
00205     void on_userResponse_textChanged();
00206
00207 signals:
00208     void sessionFinished(int cntRight, int cntWrong);
00209
00210 protected:
00211     void closeEvent(QCloseEvent* event) override {
00212         emit sessionFinished(cntRightAnswers, cntWrongAnswers);
00213         QWidget::closeEvent(event);
00214     }
00215
00216     bool eventFilter(QObject* obj, QEvent* event) override;
00217
00218 private:
00219     // ===== Core Objects =====
00220     Ui::LLTutorWindow* ui;
00221     Grammar             grammar;
00222     LL1Parser          ll1;
00223
00224     // ===== State & Grammar Tracking =====
00225     State              currentState;
00226     size_t             currentRule = 0;
00227     const unsigned kMaxHighlightTries = 3;
00228     const unsigned kMaxTotalTries = 5;
00229     unsigned          lltries = 0;
00230     unsigned          cntRightAnswers = 0, cntWrongAnswers = 0;
00231
00232     using Cell = std::pair<QString, QString>;
00233     std::vector<Cell> lastWrongCells;
00234     LLTableDialog* currentDlg = nullptr;
00235
00236     QVector<QString> sortedNonTerminals;
00237     QVector<QPair<QString, QVector<QString>>> sortedGrammar;
00238     QString          formattedGrammar;
00239
00240     QMap<QString, QMap<QString, QVector<QString>>> lltable;
00241     QVector<QVector<QString>> rawTable;
00242     QSet<QString> solutionSet;
00243
00244     // ===== Conversation Logging =====
00245     struct MessageLog {
00246         QString message;
00247         bool isUser;
00248         bool isCorrect = true;
00249         MessageLog(const QString& message, bool isUser)
00250             : message(message), isUser(isUser) {}
00251         void toggleIsCorrect() { isCorrect = false; }

```

```

00252     };
00253
00254     QVector<MessageLog> conversationLog;
00255     QWidget*          lastUserMessage      = nullptr;
00256     qsize_t           lastUserMessageLogIdx = -1;
00257
00258     QMap<QString, QString> userCAB;
00259     QMap<QString, QString> userSIG;
00260     QMap<QString, QString> userSD;
00261
00262     // ===== Helper Conversions =====
00263     std::vector<std::string> qvectorToStdVector(const QVector<QString>& qvec);
00264     QVector<QString> stdVectorToQVector(const std::vector<std::string>& vec);
00265     QSet<QString>
00266     stdUnorderedSetToQSet(const std::unordered_set<std::string>& uset);
00267     std::unordered_set<std::string>
00268     qsetToStdUnorderedSet(const QSet<QString>& qset);
00269
00270     void setupTutorial();
00271
00272     void
00273     fillSortedGrammar(); // Populate sortedGrammar from internal representation
00274
00275     QPropertyAnimation* m_shakeAnimation =
00276         nullptr; // For interrupting userResponse animation if they spam enter
00277                 // key
00278
00279     TutorialManager* tm = nullptr;
00280
00281     const QRegularExpression kRe{"^\\s+|\\s+$"};
00282     const QRegularExpression kWhitespace{"\\s+"};
00283 };
00284
00285 #endif // LLTUTORWINDOW_H

```

10.31 gui/mainwindow.cpp File Reference

```

#include "mainwindow.h"
#include "tutorialmanager.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
#include <QPixmap>
#include <QProcess>

```

Include dependency graph for mainwindow.cpp:



10.32 gui/mainwindow.h File Reference

```

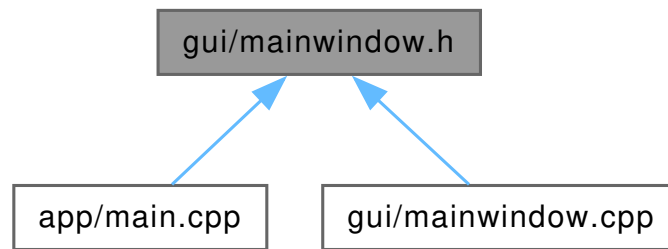
#include "grammar.hpp"
#include "grammar_factory.hpp"
#include "lltutorwindow.h"
#include "slrtutorwindow.h"
#include "tutorialmanager.h"
#include <QMainWindow>
#include <QSettings>

```

Include dependency graph for mainwindow.h:



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



Classes

- class [MainWindow](#)

Main application window of [SyntaxTutor](#), managing levels, exercises, and UI state.

10.33 mainwindow.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #ifndef MAINWINDOW_H
00020 #define MAINWINDOW_H
00021
00022 #include "grammar.hpp"
00023 #include "grammar_factory.hpp"
00024 #include "lltutorwindow.h"
00025 #include "slrtutorwindow.h"
00026 #include "tutorialmanager.h"
00027 #include <QMainWindow>
00028 #include <QSettings>
00029
00030 static const QVector<QString> levelColors = {
00031     "#2C3E50", // 1: Navy oscuro
00032     "#2980B9", // 2: Azul brillante
00033     "#16A085", // 3: Teal
00034     "#27AE60", // 4: Verde esmeralda
00035     "#8E44AD", // 5: Púrpura medio
00036     "#9B59B6", // 6: Púrpura claro
00037     "#E67E22", // 7: Naranja
00038     "#D35400", // 8: Naranja oscuro
00039     "#CD7F32", // 9: Bronce
00040     "#FFD700", // 10: Oro puro
00041 };
00042
00043 QT_BEGIN_NAMESPACE
00044 namespace Ui {
00045     class MainWindow;
00046 }
00047 QT_END_NAMESPACE
00048
00049 class MainWindow : public QMainWindow {
00050     Q_OBJECT
00051     Q_PROPERTY(unsigned userLevel READ userLevel WRITE setUserLevel NOTIFY
00052                 userLevelChanged)
00053
00054     public:
00055         MainWindow(QWidget* parent = nullptr);
  
```

```

00070
00072 ~MainWindow();
00073
00079 unsigned thresholdFor(unsigned level) { return BASE_THRESHOLD * level; }
00080
00084 unsigned userLevel() const { return m_userLevel; };
00085
00090 void setUserLevel(unsigned lvl) {
00091     unsigned clamped = qMin(lvl, MAX_LEVEL);
00092     if (m_userLevel == clamped)
00093         return;
00094     m_userLevel = clamped;
00095     emit userLevelChanged(clamped);
00096 }
00097
00098 private slots:
00102 void on_lv1Button_clicked(bool checked);
00103 void on_lv2Button_clicked(bool checked);
00104 void on_lv3Button_clicked(bool checked);
00105
00109 void on_pushButton_clicked();
00110
00114 void on_pushButton_2_clicked();
00115
00119 void on_tutorial_clicked();
00120
00124 void on_actionSobre_la_aplicaci_n_triggered();
00125
00129 void on_actionReferencia_LL_1_triggered();
00130
00134 void on_actionReferencia_SLR_1_triggered();
00135
00139 void on_idiom_clicked();
00140
00141 signals:
00146 void userLevelChanged(unsigned lvl);
00147
00152 void userLevelUp(unsigned newLevel);
00153
00154 private:
00158 void setupTutorial();
00159
00163 void restartTutorial();
00164
00170 void handleTutorFinished(int cntRight, int cntWrong);
00171
00175 void saveSettings();
00176
00180 void loadSettings();
00181
00182 Ui::MainWindow* ui;
00183 GrammarFactory factory;
00184 int level = 1;
00185 TutorialManager* tm = nullptr;
00186
00187 static constexpr unsigned MAX_LEVEL = 10;
00188 static constexpr unsigned MAX_SCORE = 999;
00189
00190 unsigned m_userLevel = 1;
00191 unsigned userScore = 0;
00192 QSettings settings;
00193
00194 const unsigned BASE_THRESHOLD = 10;
00195 };
00196 #endif // MAINWINDOW_H

```

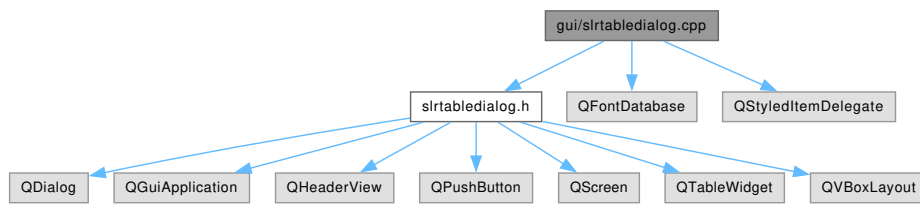
10.34 gui/slrtabledialog.cpp File Reference

```

#include "slrtabledialog.h"
#include <QFontDatabase>
#include <QStyledItemDelegate>

```

Include dependency graph for slrtabledialog.cpp:



Classes

- class [CenterAlignDelegate](#)

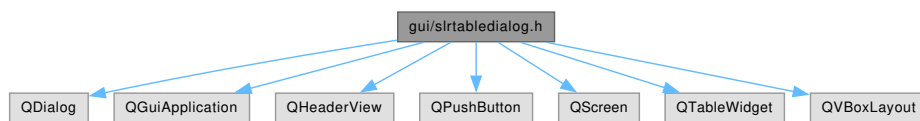
10.35 gui/slrtabledialog.h File Reference

```

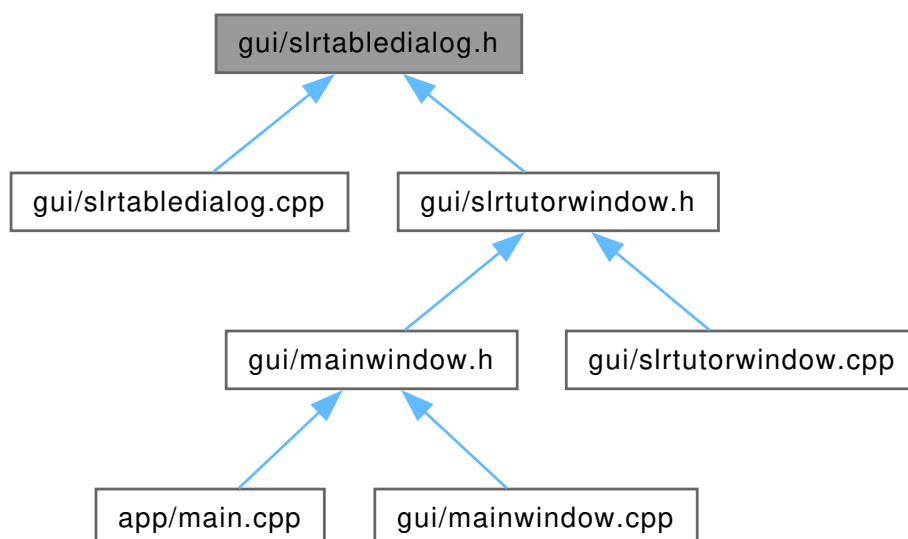
#include <QDialog>
#include <QGuiApplication>
#include <QHeaderView>
#include <QPushButton>
#include <QScreen>
#include <QTableWidget>
#include <QVBoxLayout>

```

Include dependency graph for slrtabledialog.h:



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



Classes

- class [SLRTableDialog](#)

Dialog window for completing and submitting an SLR(1) parsing table.

10.36 slrtabledialog.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #ifndef SLRTABLEDIALOG_H
00020 #define SLRTABLEDIALOG_H
00021
00022 #include <QDialog>
00023 #include <QGuiApplication>
00024 #include <QHeaderView>
00025 #include <QPushButton>
00026 #include <QScreen>
00027 #include <QTableWidget>
00028 #include <QVBoxLayout>
00029
00030 class SLRTableDialog : public QDialog {
00031     Q_OBJECT
00032 public:
00033     SLRTableDialog(int rowCount, int colCount, const QStringList& colHeaders,
00034                     QWidget* parent = nullptr,
00035                     QVector<QVector<QString>>* initialData = nullptr);
00036     QVector<QVector<QString>> getTableData() const;
00037     void setInitialData(const QVector<QVector<QString>>& data);
00038 private:
00039     QTableWidget* table;
00040     QPushButton* submitButton;
00041 };
00042 #endif // SLRTABLEDIALOG_H

```

10.37 gui/slrtutorwindow.cpp File Reference

```

#include "slrtutorwindow.h"
#include "tutorialmanager.h"
#include "ui_slrtutorwindow.h"
#include <QEasingCurve>
#include <QFontDatabase>
#include <sstream>
#include "slrwizard.h"

```

Include dependency graph for slrtutorwindow.cpp:



10.38 gui/slrtutorwindow.h File Reference

```

#include "UniqueQueue.h"
#include "grammar.hpp"
#include "slr1_parser.hpp"
#include "slrtabledialog.h"

```

```

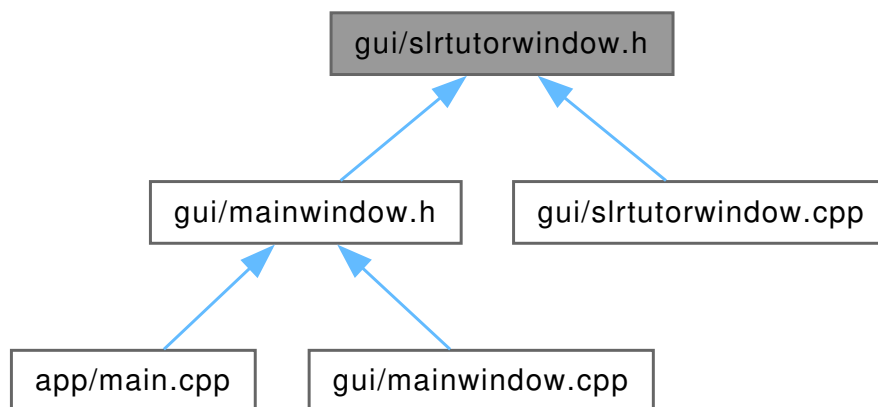
#include <QAbstractItemView>
#include <QDialog>
#include <QFileDialog>
#include <QGraphicsColorizeEffect>
#include <QListWidgetItem>
#include <QMainWindow>
#include <QMessageBox>
#include <QPropertyAnimation>
#include <QPushButton>
#include <QRegularExpression>
#include <QScrollBar>
#include <QShortcut>
#include <QTableWidget>
#include <QTextDocument>
#include <QTextEdit>
#include <QTime>
#include <QTimer>
#include <QVBoxLayout>
#include <QtPrintSupport/QtPrinter>

```

Include dependency graph for slrtutorwindow.h:



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



Classes

- class [SLRTutorWindow](#)

Main window for the SLR(1) interactive tutoring mode in [SyntaxTutor](#).

Enumerations

- enum class [StateSlr](#) {
[A](#) , [A1](#) , [A2](#) , [A3](#) ,
[A4](#) , [A_prime](#) , [B](#) , [C](#) ,
[CA](#) , [CB](#) , [D](#) , [D1](#) ,
[D2](#) , [D_prime](#) , [E](#) , [E1](#) ,
[E2](#) , [F](#) , [FA](#) , [G](#) ,
[H](#) , [H_prime](#) , [fin](#) }

10.38.1 Enumeration Type Documentation

10.38.1.1 StateSlr

```
enum class StateSlr [strong]
```

Enumerator

A	
A1	
A2	
A3	
A4	
A_prime	
B	
C	
CA	
CB	
D	
D1	
D2	
D_prime	
E	
E1	
E2	
F	
FA	
G	
H	
H_prime	
fin	

10.39 slrtutorwindow.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #ifndef SLRTUTORWINDOW_H
00020 #define SLRTUTORWINDOW_H
00021
00022 #include "UniqueQueue.h"
```



```

00023 #include "grammar.hpp"
00024 #include "slrl_parser.hpp"
00025 #include "slrtabledialog.h"
00026 #include <QAbstractItemView>
00027 #include <QDialog>
00028 #include <QFileDialog>
00029 #include <QGraphicsColorizeEffect>
00030 #include <QListWidgetItem>
00031 #include <QMainWindow>
00032 #include <QMessageBox>
00033 #include <QPropertyAnimation>
00034 #include <QPushButton>
00035 #include <QRegularExpression>
00036 #include <QScrollBar>
00037 #include <QShortcut>
00038 #include <QTableWidget>
00039 #include <QTextDocument>
00040 #include <QTextEdit>
00041 #include <QTime>
00042 #include <QTimer>
00043 #include <QVBoxLayout>
00044 #include <QtPrintSupport/QtPrinter>
00045
00046 namespace Ui {
00047 class SLRTutorWindow;
00048 }
00049
00050 // ===== SLR(1) Tutor States =====
00051 enum class StateSlr {
00052     A,
00053     A1,
00054     A2,
00055     A3,
00056     A4,
00057     A_prime,
00058     B,
00059     C,
00060     CA,
00061     CB,
00062     D,
00063     D1,
00064     D2,
00065     D_prime,
00066     E,
00067     E1,
00068     E2,
00069     F,
00070     FA,
00071     G,
00072     H,
00073     H_prime,
00074     fin
00075 };
00076
00077 class TutorialManager;
00078
00079 // ===== Main Tutor Class for SLR(1) =====
00094 class SLRTutorWindow : public QMainWindow {
00095     Q_OBJECT
00096
00097 public:
00098     // ===== Constructor / Destructor =====
00105     explicit SLRTutorWindow(const Grammar& g, TutorialManager* tm = nullptr,
00106                             QWidget* parent = nullptr);
00107     ~SLRTutorWindow();
00108
00109     // ===== Core Flow Control =====
00114     QString generateQuestion();
00115
00120     void updateState(bool isCorrect);
00121     QString
00122     FormatGrammar(const Grammar& grammar);
00123     void fillSortedGrammar();
00124
00125     // ===== UI Interaction =====
00126     void addMessage(const QString& text, bool isUser);
00127     void exportConversationToPdf(
00128         const QString& filePath);
00129     void showTable();
00130     void launchSLRWizard();
00131     void updateProgressPanel();
00132     void addUserState(unsigned id);
00133     void addUserTransition(unsigned fromId, const std::string& symbol,
00134                             unsigned toId); // Register a user-created transition
00135
00136     // ===== Visual Feedback & Animations =====
00137     void animateLabelPop(QLabel* label);

```

```

00138 void animateLabelColor(QLabel* label, const QColor& flashColor);
00139 void wrongAnimation(); // Label animation for incorrect answer
00140 void wrongUserResponseAnimation(); // Message widget animation for incorrect
00141 // answer
00142 void markLastUserIncorrect();
00143
00144 // ===== Response Verification =====
00145 bool verifyResponse(const QString& userResponse);
00146 bool verifyResponseForA(const QString& userResponse);
00147 bool verifyResponseForA1(const QString& userResponse);
00148 bool verifyResponseForA2(const QString& userResponse);
00149 bool verifyResponseForA3(const QString& userResponse);
00150 bool verifyResponseForA4(const QString& userResponse);
00151 bool verifyResponseForB(const QString& userResponse);
00152 bool verifyResponseForC(const QString& userResponse);
00153 bool verifyResponseForCA(const QString& userResponse);
00154 bool verifyResponseForCB(const QString& userResponse);
00155 bool verifyResponseForD(const QString& userResponse);
00156 bool verifyResponseForD1(const QString& userResponse);
00157 bool verifyResponseForD2(const QString& userResponse);
00158 bool verifyResponseForE(const QString& userResponse);
00159 bool verifyResponseForE1(const QString& userResponse);
00160 bool verifyResponseForE2(const QString& userResponse);
00161 bool verifyResponseForF(const QString& userResponse);
00162 bool verifyResponseForFA(const QString& userResponse);
00163 bool verifyResponseForG(const QString& userResponse);
00164 bool verifyResponseForH();
00165
00166 // ===== Correct Solutions (Auto-generated) =====
00167 QString solution(const std::string& state);
00168 std::unordered_set<Lr0Item> solutionForA();
00169 QString solutionForA1();
00170 QString solutionForA2();
00171 std::vector<std::pair<std::string, std::vector<std::string>>>
00172 solutionForA3();
00173 std::unordered_set<Lr0Item> solutionForA4();
00174 unsigned solutionForB();
00175 unsigned solutionForC();
00176 QStringList solutionForCA();
00177 std::unordered_set<Lr0Item> solutionForCB();
00178 QStringList solutionForD();
00179 QString solutionForD1();
00180 QString solutionForD2();
00181 std::ptrdiff_t solutionForE();
00182 QSet<unsigned> solutionForE1();
00183 QMap<unsigned, unsigned> solutionForE2();
00184 QSet<unsigned> solutionForF();
00185 QSet<QString> solutionForFA();
00186 QSet<QString> solutionForG();
00187
00188 // ===== Pedagogical Feedback =====
00189 QString feedback(); // Delegates to appropriate feedback based on state
00190 QString feedbackForA();
00191 QString feedbackForA1();
00192 QString feedbackForA2();
00193 QString feedbackForA3();
00194 QString feedbackForA4();
00195 QString feedbackForAPrime();
00196 QString feedbackForB();
00197 QString feedbackForB1();
00198 QString feedbackForB2();
00199 QString feedbackForBPrime();
00200 QString feedbackForC();
00201 QString feedbackForCA();
00202 QString feedbackForCB();
00203 QString feedbackForD();
00204 QString feedbackForD1();
00205 QString feedbackForD2();
00206 QString feedbackForDPrime();
00207 QString feedbackForE();
00208 QString feedbackForE1();
00209 QString feedbackForE2();
00210 QString feedbackForF();
00211 QString feedbackForFA();
00212 QString feedbackForG();
00213 QString TeachDeltaFunction(const std::unordered_set<Lr0Item>& items,
00214                             const QString& symbol);
00215 void TeachClosureStep(std::unordered_set<Lr0Item>& items, unsigned int size,
00216                       std::unordered_set<std::string>& visited, int depth,
00217                       QString& output);
00218 QString TeachClosure(const std::unordered_set<Lr0Item>& initialItems);
00219 private slots:
00220 void on_confirmButton_clicked();
00221 void on_userResponse_textChanged();
00222
00223 signals:
00224 void sessionFinished(int cntRight, int cntWrong);

```

```

00225
00226 protected:
00227     void closeEvent(QCloseEvent* event) override {
00228         emit sessionFinished(cntRightAnswers, cntWrongAnswers);
00229         QWidget::closeEvent(event);
00230     }
00231
00232 private:
00233     // ===== Helper Functions =====
00234     std::vector<std::string> qvectorToStdVector(const QVector<QString>& qvec);
00235     QVector<QString> stdVectorToQVector(const std::vector<std::string>& vec);
00236     QSet<QString>
00237     stdUnorderedSetToQSet(const std::unordered_set<std::string>& uset);
00238     std::unordered_set<std::string>
00239     qsetToStdUnorderedSet(const QSet<QString>& qset);
00240     std::unordered_set<Lr0Item> ingestUserItems(const QString& userResponse);
00241     std::vector<std::pair<std::string, std::vector<std::string>>>
00242         ingestUserRules(const QString& userResponse);
00243     void setupTutorial();
00244     // ===== Core Components =====
00245     Ui::SLRTutorWindow* ui;
00246     Grammar grammar;
00247     SLR1Parser srlr;
00248
00249     // ===== State and Grammar Tracking =====
00250     StateSlr currentState;
00251     QVector<QString> sortedNonTerminals;
00252     QVector<QPair<QString, QVector<QString>>> sortedGrammar;
00253     QString formattedGrammar;
00254
00255     unsigned cntRightAnswers = 0;
00256     unsigned cntWrongAnswers = 0;
00257
00258     // ===== State Machine Runtime Variables =====
00259     std::unordered_set<state> userMadeStates; // All states the user has created
00260     std::unordered_map<unsigned, std::unordered_map<std::string, unsigned>
00261         userMadeTransitions; // Transitions made by the user
00262     UniqueQueue<unsigned>
00263         statesIdQueue; // States to be processed in B-C-CA-CB loop
00264     unsigned currentStateId = 0;
00265     state currentSlrState;
00266
00267     QStringList followSymbols; // Used in CA-CB loop
00268     qsize_t currentFollowSymbolsIdx = 0;
00269     unsigned int nextStateId = 0;
00270
00271     QVector<const state*> statesWithLr0Conflict; // Populated in F
00272     std::queue<unsigned> conflictStatesIdQueue;
00273     unsigned currentConflictStateId = 0;
00274     state currentConflictState;
00275
00276     std::queue<unsigned>
00277         reduceStatesIdQueue; // States without conflicts but with reduce
00278     unsigned currentReduceStateId = 0;
00279     state currentReduceState;
00280
00281     struct ActionEntry {
00282         enum Type { Shift, Reduce, Accept, Goto } type;
00283         int target;
00284         static ActionEntry makeShift(int s) { return {Shift, s}; }
00285         static ActionEntry makeReduce(int r) { return {Reduce, r}; }
00286         static ActionEntry makeAccept() { return {Accept, 0}; }
00287         static ActionEntry makeGoto(int g) { return {Goto, g}; }
00288     };
00289
00290     QMap<int, QMap<QString, ActionEntry>> slrtable;
00291     QVector<QVector<QString>> rawTable;
00292
00293     // ===== Conversation Log =====
00294     struct MessageLog {
00295         QString message;
00296         bool isUser;
00297         bool isCorrect = true;
00298
00299         MessageLog(const QString& message, bool isUser)
00300             : message(message), isUser(isUser) {}
00301
00302         void toggleIsCorrect() { isCorrect = false; }
00303     };
00304
00305     QVector<MessageLog> conversationLog;
00306     QWidget* lastUserMessage = nullptr;
00307     qsize_t lastUserMessageLogIdx = -1;
00308
00309     QPropertyAnimation* m_shakeAnimation =
00310         nullptr; // For interrupting userResponse animation if they spam enter
00311         // key

```

```

00312
00313     TutorialManager* tm;
00314
00315     QRegularExpression      re{"^\\s+|\\s+$"};
00316     const QRegularExpression kWhitespace{"\\s+"};
00317 };
00318
00319 #endif // SLRTUTORWINDOW_H

```

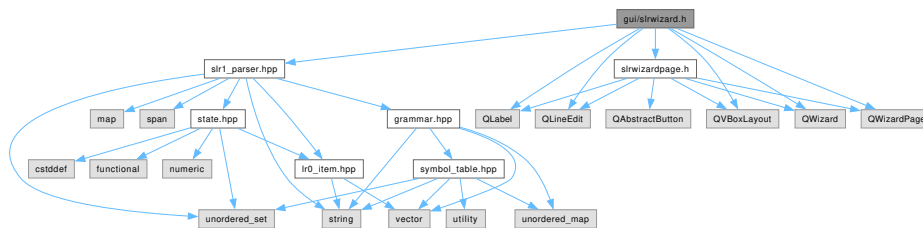
10.40 gui/slrwizard.h File Reference

```

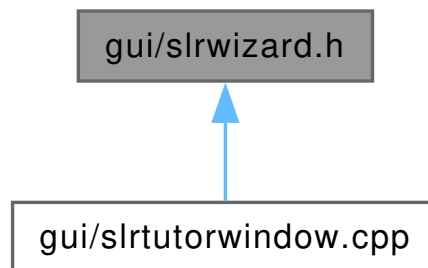
#include "slr1_parser.hpp"
#include "slrwizardpage.h"
#include <QLabel>
#include <QLineEdit>
#include <QVBoxLayout>
#include <QWizard>
#include <QWizardPage>

```

Include dependency graph for slrwizard.h:



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



Classes

- class [SLRWizard](#)

Interactive assistant that guides the student step-by-step through the SLR(1) parsing table.

10.41 slrwizard.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.

```

```

00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #ifndef SLRWIZARD_H
00020 #define SLRWIZARD_H
00021
00022 #include "slr1_parser.hpp"
00023 #include "slrwizardpage.h"
00024 #include <QLabel>
00025 #include <QLineEdit>
00026 #include <QVBoxLayout>
00027 #include <QWizard>
00028 #include <QWizardPage>
00029
00045 class SLRWizard : public QWizard {
00046     Q_OBJECT
00047     public:
00059     SLRWizard(SLR1Parser& parser, const QVector<QVector<QString>& rawTable,
00060             const QStringList& colHeaders,
00061             const QVector<QPair<QString, QVector<QString>>& sortedGrammar,
00062             QWidget* parent = nullptr)
00063     : QWizard(parent) {
00064         setWindowTitle(tr("Ayuda interactiva: Tabla SLR(1)"));
00065
00066         const int nTerm =
00067             parser.gr_.st_.terminals_.contains(parser.gr_.st_.EPSILON_)
00068             ? parser.gr_.st_.terminals_.size() - 1
00069             : parser.gr_.st_.terminals_.size();
00070         SLRWizardPage* last = nullptr;
00071         // Generar explicación y páginas
00072         int rows = rawTable.size();
00073         int cols = colHeaders.size();
00074         for (int i = 0; i < rows; ++i) {
00075             for (int j = 0; j < cols; ++j) {
00076                 QString sym = colHeaders[j];
00077                 QString expected;
00078                 QString explanation;
00079                 if (j < nTerm) {
00080                     auto itAct = parser.actions_.at(i).find(sym.toStdString());
00081                     SLR1Parser::s_action act =
00082                         (itAct != parser.actions_.at(i).end()
00083                          ? itAct->second
00084                          : SLR1Parser::s_action{nullptr,
00085                                                  SLR1Parser::Action::Empty});
00086                     switch (act.action) {
00087                         case SLR1Parser::Action::Shift: {
00088                             unsigned to =
00089                                 parser.transitions_.at(i).at(sym.toStdString());
00090                             expected = QString("s%1").arg(to);
00091                             explanation = tr("Estado %1: existe transición (%1, "
00092                                             "%2'). ¿A qué "
00093                                             "estado harías shift?")
00094                                 .arg(i)
00095                                 .arg(sym);
00096                             break;
00097                         }
00098                         case SLR1Parser::Action::Reduce: {
00099                             int idx = -1;
00100                             for (int k = 0; k < sortedGrammar.size(); ++k) {
00101                                 auto& rule = sortedGrammar[k];
00102                                 if (rule.first.toStdString() ==
00103                                     act.item->antecedent_ &&
00104                                     std::vectorToQVector(act.item->consequent_) ==
00105                                     rule.second) {
00106                                     idx = k;
00107                                     break;
00108                                 }
00109                             }
00110                             expected = QString("r%1").arg(idx);
00111                             // explicación con FOLLOW
00112                             std::unordered_set<std::string> F;
00113                             F = parser.Follow(act.item->antecedent_);
00114                             QStringList followList;
00115                             for (auto& t : F)
00116                                 followList << QString::fromStdString(t);
00117                             explanation = tr("Estado %1: contiene el ítem [%2 → "
00118                                             "...] y '%3' "
00119                                             "SIG(%2). ¿Qué regla usas para "
00120                                             "reducir (0, 1, ...)?")
00121                                 .arg(i)
00122                                 .arg(QString::fromStdString(
00123                                     act.item->antecedent_))
00124                                 .arg(colHeaders[j]);
00125                             break;
00126                         }
00127                     }
00128                 }
00129             }
00130         }
00131     }

```

```

00127         case SLR1Parser::Action::Accept:
00128             expected = "acc";
00129             explanation = tr("Estado %1: contiene [S → A · $]. "
00130                             "¿Qué palabra clave "
00131                             "usas para aceptar?")
00132                             .arg(i);
00133             break;
00134         case SLR1Parser::Action::Empty:
00135             default:
00136                 continue;
00137         }
00138     } else {
00139         // GOTO sobre no terminal
00140         auto nonT = sym.toStdString();
00141         if (!parser.transitions_.contains(i)) {
00142             continue;
00143         }
00144         auto itGo = parser.transitions_.at(i).find(nonT);
00145         if (itGo != parser.transitions_.at(i).end()) {
00146             expected = QString::number(itGo->second);
00147             explanation = tr("Estado %1: (%1, '%2') existe. ¿A "
00148                             "qué estado va "
00149                             "la transición? (pon solo el número)")
00150                             .arg(i)
00151                             .arg(sym);
00152         } else {
00153             continue;
00154         }
00155     }
00156
00157     SLRWizardPage* page =
00158         new SLRWizardPage(i, sym, explanation, expected, this);
00159     last = page;
00160     addPage(page);
00161 }
00162
00163 if (last) {
00164     last->setFinalPage(true);
00165 }
00166 }
00167
00174 QVector<QString> stdVectorToQVector(const std::vector<std::string>& vec) {
00175     QVector<QString> result;
00176     result.reserve(vec.size());
00177     for (const auto& str : vec) {
00178         result.push_back(QString::fromStdString(str));
00179     }
00180     return result;
00181 }
00182 };
00183
00184 #endif // SLRWIZARD_H

```

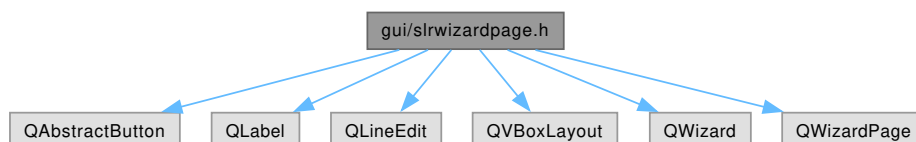
10.42 gui/slrwizardpage.h File Reference

```

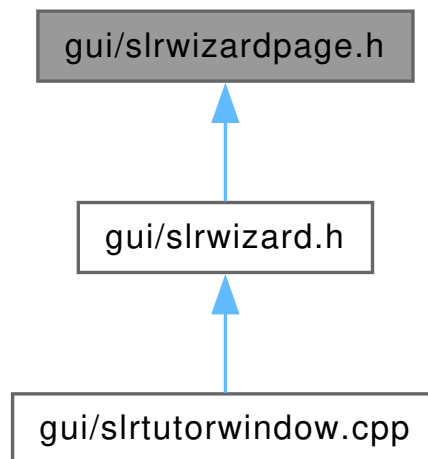
#include <QAbstractButton>
#include <QLabel>
#include <QLineEdit>
#include <QVBoxLayout>
#include <QWizard>
#include <QWizardPage>

```

Include dependency graph for slrwizardpage.h:



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



Classes

- class [SLRWizardPage](#)

A single step in the SLR(1) guided assistant for table construction.

10.43 slrwizardpage.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #ifndef SLRWIZARDPAGE_H
00020 #define SLRWIZARDPAGE_H
00021
00022 #include <QAbstractButton>
00023 #include <QLabel>
00024 #include <QLineEdit>
00025 #include <QVBoxLayout>
00026 #include <QWizard>
00027 #include <QWizardPage>
00028
00041 class SLRWizardPage : public QWizardPage {
00042     Q_OBJECT
00043     public:
00054         SLRWizardPage(int state, const QString& symbol, const QString& explanation,
00055                        const QString& expected, QWidget* parent = nullptr)
00056             : QWizardPage(parent), m_state(state), m_symbol(symbol),
00057               m_expected(expected) {
00058             setTitle(tr("Estado %1, símbolo '%2'").arg(state).arg(symbol));
00059
00060             QLabel* lbl = new QLabel(explanation, this);
00061             lbl->setWordWrap(true);
00062
00063             m_edit = new QLineEdit(this);
00064             m_edit->setPlaceholderText(
00065                 tr("Escribe tu respuesta (p.ej. s3, r2, acc, 5)"));
00066
00067             QVBoxLayout* layout = new QVBoxLayout(this);

```

```

00068         layout->addWidget(lbl);
00069         layout->addWidget(m_edit);
00070         setLayout(layout);
00071
00072         connect(m_edit, &QLineEdit::textChanged, this,
00073                 &SLRWizardPage::onTextChanged);
00074     }
00075     private slots:
00081     void onTextChanged(const QString& text) {
00082         bool correct = (text.trimmed() == m_expected);
00083         setComplete(correct);
00084         if (correct) {
00085             setSubTitle(
00086                 tr(" Respuesta correcta, pasa a la siguiente pregunta"));
00087         } else {
00088             setSubTitle(tr(" Incorrecto, revisa el enunciado. Consulta los "
00089                             "estados que has construido."));
00090         }
00091         wizard()->button(QWizard::NextButton)->setEnabled(correct);
00092     }
00093
00094     private:
00099     void setComplete(bool complete) {
00100         m_isComplete = complete;
00101         emit completeChanged();
00102     }
00103
00109     bool isComplete() const override { return m_isComplete; }
00110
00111     int         m_state;
00112     QString     m_symbol;
00113     QString     m_expected;
00114     QLineEdit* m_edit;
00115     bool        m_isComplete =
00116         false;
00117 };
00118
00119 #endif // SLRWIZARDPAGE_H

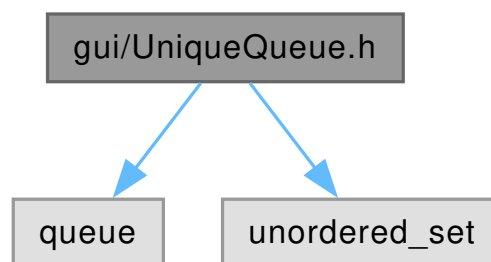
```

10.44 gui/UniqueQueue.h File Reference

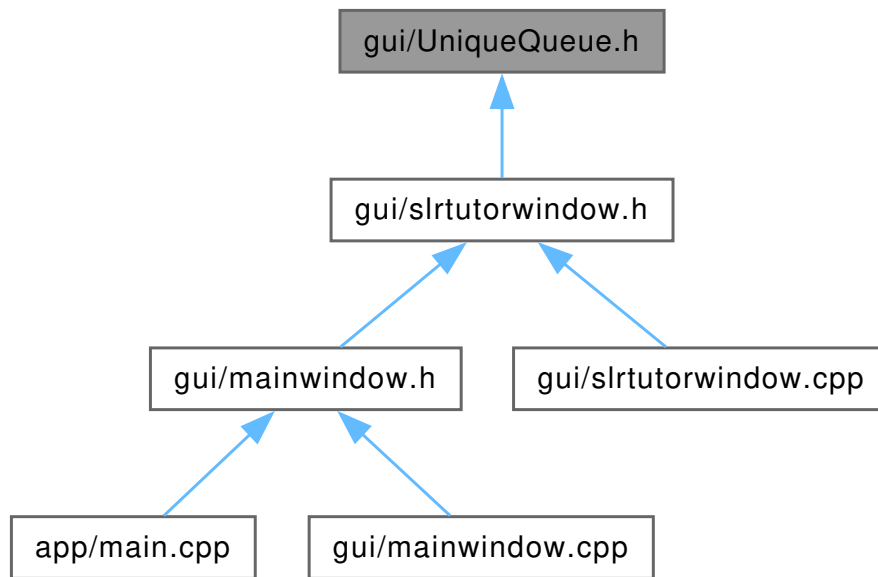
```
#include <queue>
```

```
#include <unordered_set>
```

Include dependency graph for UniqueQueue.h:



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



Classes

- class `UniqueQueue< T >`

A queue that ensures each element is inserted only once.

10.45 UniqueQueue.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #ifndef UNIQUEQUEUE_H
00020 #define UNIQUEQUEUE_H
00021 #include <queue>
00022 #include <unordered_set>
00023
00037 template <typename T> class UniqueQueue {
00038 public:
00043     void push(const T& value) {
00044         if (seen_.insert(value).second) {
00045             queue_.push(value);
00046         }
00047     }
00048
00052     void pop() {
00053         if (!queue_.empty()) {
00054             queue_.pop();
00055         }
00056     }
00057
00062     const T& front() const { return queue_.front(); }
00063
00068     bool empty() const { return queue_.empty(); }

```

```

00069
00073     void clear() {
00074         while (!queue_.empty())
00075             queue_.pop();
00076         seen_.clear();
00077     }
00078
00079     private:
00080         std::queue<T> queue_;
00081         std::unordered_set<T> seen_;
00082 };
00083 #endif // UNIQUEQUEUE_H

```

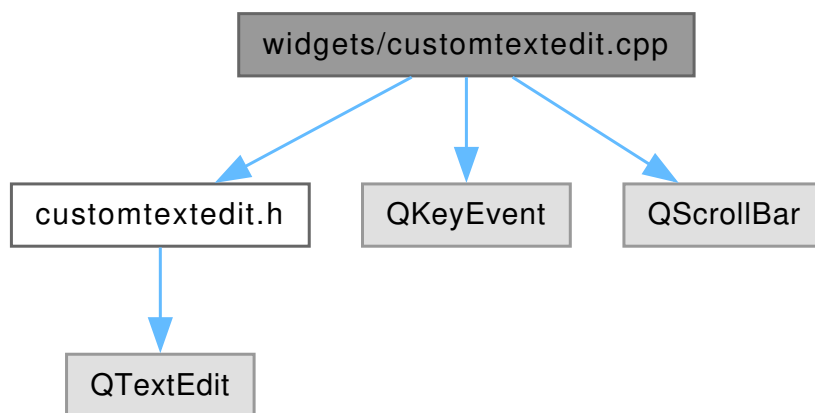
10.46 widgets/customtextedit.cpp File Reference

```

#include "customtextedit.h"
#include <QKeyEvent>
#include <QScrollBar>

```

Include dependency graph for customtextedit.cpp:



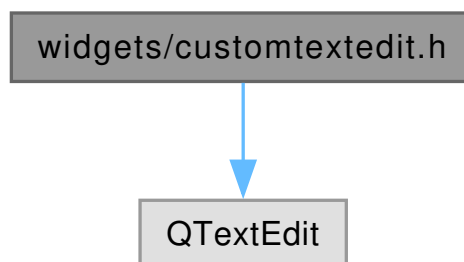
10.47 widgets/customtextedit.h File Reference

```

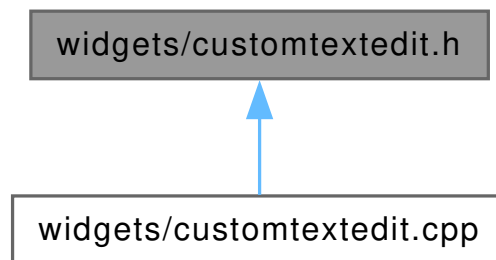
#include <QTextEdit>

```

Include dependency graph for customtextedit.h:



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



Classes

- class [CustomTextEdit](#)

10.48 customtextedit.h

[Go to the documentation of this file.](#)

```

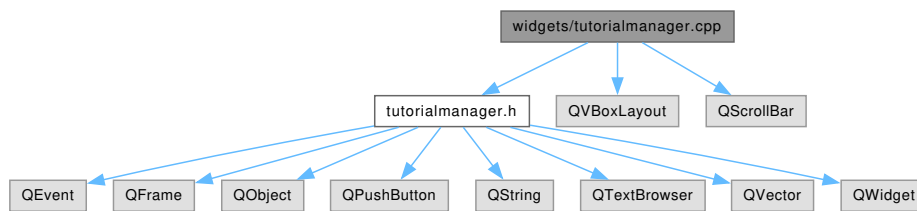
00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,
00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #ifndef CUSTOMTEXTEDIT_H
00020 #define CUSTOMTEXTEDIT_H
00021
00022 #include <QTextEdit>
00023
00024 class CustomTextEdit : public QTextEdit {
00025     Q_OBJECT
00026     public:
00027         explicit CustomTextEdit(QWidget* parent = nullptr);
00028
00029     signals:
00030         void sendRequested();
00031
00032     protected:
00033         void keyPressEvent(QKeyEvent* event) override;
00034 };
00035
00036 #endif // CUSTOMTEXTEDIT_H
  
```

10.49 widgets/tutorialmanager.cpp File Reference

```

#include "tutorialmanager.h"
#include <QVBoxLayout>
#include <QScrollBar>
  
```

Include dependency graph for tutorialmanager.cpp:



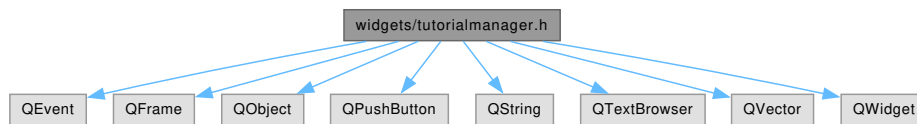
10.50 widgets/tutorialmanager.h File Reference

```

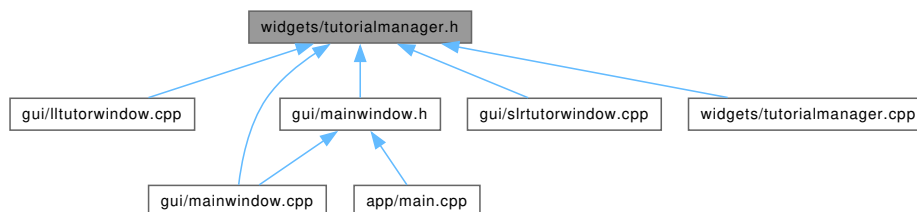
#include <QEvent>
#include <QFrame>
#include <QObject>
#include <QPushButton>
#include <QString>
#include <QTextBrowser>
#include <QVector>
#include <QWidget>

```

Include dependency graph for tutorialmanager.h:



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



Classes

- struct [TutorialStep](#)
Represents a single step in the tutorial sequence.
- class [TutorialManager](#)
Manages interactive tutorials by highlighting UI elements and guiding the user.

10.51 tutorialmanager.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003  * Copyright (C) 2025 Jose R. (jose-rzm)
00004  *
00005  * This program is free software: you can redistribute it and/or modify it
00006  * under the terms of the GNU General Public License as published by
00007  * the Free Software Foundation, either version 3 of the License, or
00008  * (at your option) any later version.
00009  *
00010  * This program is distributed in the hope that it will be useful,

```

```

00011  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013  * GNU General Public License for more details.
00014  *
00015  * You should have received a copy of the GNU General Public License
00016  * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017  */
00018
00019 #ifndef TUTORIALMANAGER_H
00020 #define TUTORIALMANAGER_H
00021
00022 #include <QEvent>
00023 #include <QFrame>
00024 #include <QObject>
00025 #include <QPushButton>
00026 #include <QString>
00027 #include <QTextBrowser>
00028 #include <QVector>
00029 #include <QWidget>
00030
00031 struct TutorialStep {
00032     QWidget* target;
00033     QString htmlText;
00034 };
00035
00036 class TutorialManager : public QObject {
00037     Q_OBJECT
00038 public:
00039     TutorialManager(QWidget* rootWindow);
00040
00041     void addStep(QWidget* target, const QString& htmlText);
00042
00043     void start();
00044
00045     void setRootWindow(QWidget* newRoot);
00046
00047     void clearSteps();
00048
00049     void hideOverlay();
00050
00051     void finishLL1();
00052
00053     void finishSLR1();
00054
00055 protected:
00056     bool eventFilter(QObject* obj, QEvent* ev) override;
00057
00058 signals:
00059     void stepStarted(int index);
00060
00061     void tutorialFinished();
00062
00063     void ll1Finished();
00064
00065     void slr1Finished();
00066
00067 public slots:
00068     void nextStep();
00069
00070 private:
00071     void showOverlay();
00072
00073     void repositionOverlay();
00074
00075     QWidget* m_root;
00076     QVector<TutorialStep> m_steps;
00077     int m_index = -1;
00078
00079     QWidget* m_overlay = nullptr;
00080     QFrame* m_highlight =
00081         nullptr;
00082     QTextBrowser* m_textBox = nullptr;
00083     QPushButton* m_nextBtn = nullptr;
00084 };
00085 #endif // TUTORIALMANAGER_H

```


Index

- AddProduction
 - Grammar, [30](#)
- AdjustTerminals
 - GrammarFactory, [33](#)
- app Directory Reference, [21](#)
- axiom_
 - Grammar, [31](#)
- backend Directory Reference, [21](#)
- CenterAlignDelegate, [25](#)
 - initStyleOption, [25](#)
- ComputeFirstSets
 - LL1Parser, [49](#)
- ComputeFollowSets
 - LL1Parser, [49](#)
- CreateLL1Table
 - LL1Parser, [50](#)
- CreateLv2Item
 - GrammarFactory, [34](#)
- current
 - SyntaxTutor::Version, [23](#)
- CustomTextEdit, [26](#)
 - CustomTextEdit, [26](#)
 - keyPressEvent, [27](#)
 - sendRequested, [27](#)
- Debug
 - Grammar, [30](#)
- FactoryItem
 - GrammarFactory::FactoryItem, [28](#)
- FilterRulesByConsequent
 - Grammar, [30](#)
- First
 - LL1Parser, [50](#)
- first_sets_
 - LL1Parser, [52](#)
- Follow
 - LL1Parser, [51](#)
- follow_sets_
 - LL1Parser, [52](#)
- g_
 - Grammar, [31](#)
 - GrammarFactory::FactoryItem, [28](#)
- GenerateNewNonTerminal
 - GrammarFactory, [34](#)
- GenLL1Grammar
 - GrammarFactory, [35](#)
- GenSLR1Grammar
 - GrammarFactory, [35](#)
- gr_
 - LL1Parser, [52](#)
- Grammar, [28](#)
 - AddProduction, [30](#)
 - axiom_, [31](#)
 - Debug, [30](#)
 - FilterRulesByConsequent, [30](#)
 - g_, [31](#)
 - Grammar, [29](#)
 - HasEmptyProduction, [30](#)
 - SetAxiom, [31](#)
 - Split, [31](#)
 - st_, [31](#)
- GrammarFactory, [32](#)
 - AdjustTerminals, [33](#)
 - CreateLv2Item, [34](#)
 - GenerateNewNonTerminal, [34](#)
 - GenLL1Grammar, [35](#)
 - GenSLR1Grammar, [35](#)
 - HasCycle, [36](#)
 - HasDirectLeftRecursion, [36](#)
 - HasIndirectLeftRecursion, [37](#)
 - HasUnreachableSymbols, [37](#)
 - Init, [38](#)
 - IsInfinite, [38](#)
 - items, [47](#)
 - LeftFactorize, [38](#)
 - LongestCommonPrefix, [39](#)
 - Lv1, [40](#)
 - Lv2, [40](#)
 - Lv3, [40](#)
 - Lv4, [41](#)
 - Lv5, [41](#)
 - Lv6, [42](#)
 - Lv7, [42](#)
 - Merge, [43](#)
 - non_terminal_alphabet_, [47](#)
 - NormalizeNonTerminals, [44](#)
 - NullableSymbols, [44](#)
 - PickOne, [45](#)
 - RemoveLeftRecursion, [45](#)
 - StartsWith, [46](#)
 - terminal_alphabet_, [47](#)
- GrammarFactory::FactoryItem, [27](#)
 - FactoryItem, [28](#)
 - g_, [28](#)
 - st_, [28](#)
- gui Directory Reference, [22](#)

- HasCycle
 - GrammarFactory, [36](#)
- HasDirectLeftRecursion
 - GrammarFactory, [36](#)
- HasEmptyProduction
 - Grammar, [30](#)
- HasIndirectLeftRecursion
 - GrammarFactory, [37](#)
- HasUnreachableSymbols
 - GrammarFactory, [37](#)
- Init
 - GrammarFactory, [38](#)
- initStyleOption
 - CenterAlignDelegate, [25](#)
- IsInfinite
 - GrammarFactory, [38](#)
- items
 - GrammarFactory, [47](#)
- keyPressEvent
 - CustomTextEdit, [27](#)
- LeftFactorize
 - GrammarFactory, [38](#)
- LL1Parser, [47](#)
 - ComputeFirstSets, [49](#)
 - ComputeFollowSets, [49](#)
 - CreateLL1Table, [50](#)
 - First, [50](#)
 - first_sets_, [52](#)
 - Follow, [51](#)
 - follow_sets_, [52](#)
 - gr_, [52](#)
 - LL1Parser, [48](#)
 - PredictionSymbols, [51](#)
- LongestCommonPrefix
 - GrammarFactory, [39](#)
- Lv1
 - GrammarFactory, [40](#)
- Lv2
 - GrammarFactory, [40](#)
- Lv3
 - GrammarFactory, [40](#)
- Lv4
 - GrammarFactory, [41](#)
- Lv5
 - GrammarFactory, [41](#)
- Lv6
 - GrammarFactory, [42](#)
- Lv7
 - GrammarFactory, [42](#)
- Merge
 - GrammarFactory, [43](#)
- non_terminal_alphabet_
 - GrammarFactory, [47](#)
- NormalizeNonTerminals
 - GrammarFactory, [44](#)
- NullableSymbols
 - GrammarFactory, [44](#)
- PickOne
 - GrammarFactory, [45](#)
- PredictionSymbols
 - LL1Parser, [51](#)
- raw
 - SyntaxTutor::Version, [23](#)
- RemoveLeftRecursion
 - GrammarFactory, [45](#)
- sendRequested
 - CustomTextEdit, [27](#)
- SetAxiom
 - Grammar, [31](#)
- Split
 - Grammar, [31](#)
- st_
 - Grammar, [31](#)
 - GrammarFactory::FactoryItem, [28](#)
- StartsWith
 - GrammarFactory, [46](#)
- SyntaxTutor, [23](#)
- SyntaxTutor: An interactive Tool for Learning Syntax Analysis, [1](#)
- SyntaxTutor::Version, [23](#)
 - current, [23](#)
 - raw, [23](#)
- terminal_alphabet_
 - GrammarFactory, [47](#)
- widgets Directory Reference, [22](#)