# DynGenPar

Generated by Doxygen 1.8.13

# Contents

# Chapter 1

# Main Page

## 1.1 Introduction

DynGenPar [1] is an innovative parser based on a new principle combining bottom-up and top-down features of traditional parsers. The most unique feature of the algorithm is the possibility to add rules to the grammar at almost any time, even during parsing. DynGenPar has the following characterizing properties:

*Dynamic* = The grammar is not hardcoded as in usual table-driven approaches, such as (Generalized) LR or Earley's algorithm. Instead, the algorithm works on a runtime representation of the grammar, which allows efficient handling of dynamic grammar changes. To decide when and how to shift or reduce, we use, instead of the usual LR tables, the initial graph, a graph which is easily updated as the grammar changes, along with some runtime top-down information.

*Generalized* = The algorithm exhaustively parses ambiguous grammars. In addition, epsilon productions are considered in order to support arbitrary CFGs. (Left recursion is naturally supported thanks to the bottom-up nature of the algorithm.) We use graph-structured (DAG-structured) stacks similar to the ones used in Tomita's Generalized LR algorithm. As additional generalizations, we also support Parallel Multiple Context-Free Grammars (PMCFGs) and next token constraints (useful, e.g. for scannerless parsing).

Due to the dynamic design, a parser generator is not needed. Instead, the parser can simply be used as a library.

DynGenPar supports dynamic grammar additions, incremental parsing, prediction, rule labels, custom parse actions, arbitrary token sources, hierarchical parsing, parallel multiple context-free grammars (PMCFGs), and next token constraints. See section Features for details.

## 1.2 Basic Usage

All the APIs provided by DynGenPar are in the DynGenPar namespace. You can use

```
using namespace DynGenPar;
```

to bring in the entire namespace.

The main class of DynGenPar is DynGenPar::Parser. To do any parsing, you will always have to instantiate at least one object of the DynGenPar::Parser class, or a subclass such as DynGenPar::PgfParser. Parsing is done through the DynGenPar::Parser::parse methods, either the fine-grained version with several arguments or the binding-friendly version which operates on a DynGenPar::ParseState object grouping all the arguments.

The parser operates on a stream of tokens. Those tokens have to be provided by a token source, i.e. a class implementing the DynGenPar::TokenSource interface. DynGenPar provides several ready-made token sources. You can also implement your own: Your class only has to derive from DynGenPar::TokenSource and implement the needed virtual methods, at least the pure virtual DynGenPar::TokenSource::readToken. In the following example, we will use the simplest token source, which operates directly on a list of tokens: the DynGenPar::ListTokenSource.

This is a basic example for how to use DynGenPar::Parser:

```
DynGenPar::ListTokenSource tokenSource;
DynGenPar::Parser parser(&tokenSource);

// We have to specify the grammar.
// In this example, the grammar is hardcoded.
parser.tokens << "a" << "b" << "c";
parser.rules["S"] << (Rule() << "A")
                  << (Rule() << "A" << "S");
parser.rules["A"] << (Rule() << "a")
                  << (Rule() << "b")
                  << (Rule() << "c");
parser.startCat = "S";

// Whenever we set the grammar directly, we have to call this function.
parser.initCaches();

// And here, we specify the sample input, also hardcoded.
tokenSource.inputTokens << "b" << "a";

// Now we parse the input, using the default arguments for
// DynGenPar::Parser::parse.
QList<DynGenPar::Match> matches = parser.parse();
```

The DynGenPar::Match class contains the results of the parsing process; in particular, the parsed tree(s) in a packed, i.e. DAG (directed acyclic graph) -structured, representation, see the DynGenPar::Match::tree field.

The following example shows how to iterate over a parse tree:

```
static void printSubtree(const DynGenPar::Node &node, int level,
                         QTextStream &stream);

// prints the parse tree given as an argument to stdout
static void printParseTree(const DynGenPar::Node &tree) {
  QTextStream stream(stdout);
  printSubtree(tree, 0, stream);
}

// recursive helper function for the above, which does the actual work
static void printSubtree(const DynGenPar::Node &node, int level,
                         QTextStream &stream) {
  for (int i=0; i<level; i++) stream << ' ';
  stream << node.cat << endl;
  switch (node.children.size()) {
    case 0: // no alternatives = error node
      for (int i=0; i<=level; i++) stream << ' ';
      stream << "ERROR" << endl;
      break;
    case 1: // 1 alternative, normal tree
      foreach (const DynGenPar::Node &child, node.children.first())
        printSubtree(child, level+1, stream);
      break;
    default: // multiple alternatives
      foreach (const DynGenPar::Alternative &subtree, node.
      children) {
        for (int i=0; i<=level; i++) stream << ' ';
        stream << "ALTERNATIVE" << endl;
        foreach (const DynGenPar::Node &child, subtree)
          printSubtree(child, level+2, stream);
      }
      break;
  }
}
```

You would call the above `printParseTree` function using e.g.

```
foreach (const DynGenPar::Match &m, matches) printParseTree(m.
      tree);
```

on the result of the previous example.

By default, category names, i.e. the DynGenPar::Cat typedef, are strings, which is convenient for debugging, but often not what is wanted in practice. For efficiency and/or interoperability, categories can be forced to be integers instead, by #defining DYNGENPAR_INTEGER_CATEGORIES. Some contexts such as the PGF support (pgf.h), the DynGenPar::FlexLexerTokenSource (flexlexertokensource.h) and the Java bindings always force this option.

## 1.3 Java Bindings

Java bindings for DynGenPar based on Qt Jambi are provided. Most of the DynGenPar API is available also to Java developers.

All the APIs provided by DynGenPar are in the `concise.DynGenPar` package. You can use

```
import concise.DynGenPar.*;
```

to import the entire package.

The following mappings and restrictions apply:

- The DynGenPar namespace is mapped to the `concise.DynGenPar` package.
- The global functions in the DynGenPar namespace are mapped to static methods of a `concise.DynGenPar.Util` class.
- The global constants and enumerations in the DynGenPar namespace are mapped as follows:
  - DynGenPar::PgfToken* to `concise.DynGenPar.Pgf.Token*`
  - DynGenPar::PreludeBind to `concise.DynGenPar.Pgf.PreludeBind`
  - DynGenPar::ByteToken* to `concise.DynGenPar.ByteTokenSource.*`
- DYNGENPAR_INTEGER_CATEGORIES is always defined, i.e. category identifiers are always integers, not strings. See DynGenPar::Cat.
- The internals of parse stacks cannot be accessed through Java. Parse stacks are only exposed in the form of opaque parse states (`concise.DynGenPar.ParseState`, i.e. DynGenPar::ParseState). The DynGenPar::ParseState::incrementalStacks member cannot be accessed from Java. In particular, only the overloads of DynGenPar::Parser::parse and of the Prediction methods in DynGenPar::Parser which take a DynGenPar::ParseState are available.
- Likewise, the DynGenPar::PseudoCatScopeData class and the DynGenPar::PseudoCatScope::data method that returns a pointer to it are not available.
- Public class/structure member fields such as DynGenPar::Parser::tokens are mapped to getters, e.g. `concise.DynGenPar.Parser.tokens`, and setters, e.g. `concise.DynGenPar.Parser.setTokens`. This is because Java only supports public member fields for Java code, it does not support native fields nor a property mechanism that would allow hiding the required function calls to access the C++ class's member field.
- Qt lists, hash tables etc. are mapped to their Java equivalents.

- Multi-valued hash tables are not directly supported in Java, therefore they are mapped to list-valued hash tables.
- List classes (classes derived from QList) can be appended to using a Java-style `add` method, and converted to an iterable Java list using a `toList` method. (In C++, the `toList` method is just an alias for a trivial cast and returns a writable reference to a QList. In Java, due to the required conversion, the returned list can be used for reading only.)
- Java does not support typedef, so unfortunately you have to spell out the actual types of things like DynGenPar::RuleSet.

This documentation documents the C++ API. With the exception of the above changes, the same interfaces can also be used from Java. In particular, it is transparently possible to implement virtual methods of C++ classes in derived Java classes, and thus interfaces required by the C++ code can also be implemented in Java.

In Java, the basic example for how to use DynGenPar::Parser would look as follows:

```
concise.DynGenPar.ListTokenSource tokenSource
  = new concise.DynGenPar.ListTokenSource();
concise.DynGenPar.Parser parser = new concise.DynGenPar.Parser(tokenSource);

// We have to specify the grammar.
// In this example, the grammar is hardcoded.
// Category identifiers must be integers, but char promotes to int.
// Unfortunately, the syntax is not quite as pretty as the C++ one.
parser.addToken('a');
parser.addToken('b');
parser.addToken('c');
java.util.HashMap<Integer, java.util.List<concise.DynGenPar.Rule> > rules
  = new java.util.HashMap<Integer,
                          java.util.List<concise.DynGenPar.Rule> >();
java.util.List<concise.DynGenPar.Rule> sRules
  = new java.util.ArrayList<concise.DynGenPar.Rule>();
concise.DynGenPar.Rule rule1 = new concise.DynGenPar.Rule();
rule1.add('A');
sRules.add(rule1);
concise.DynGenPar.Rule rule2 = new concise.DynGenPar.Rule();
rule2.add('A');
rule2.add('S');
sRules.add(rule2);
rules.put('S', sRules);
java.util.List<concise.DynGenPar.Rule> aRules
  = new java.util.ArrayList<concise.DynGenPar.Rule>();
concise.DynGenPar.Rule rule3 = new concise.DynGenPar.Rule();
rule3.add('a');
aRules.add(rule3);
concise.DynGenPar.Rule rule4 = new concise.DynGenPar.Rule();
rule4.add('b');
aRules.add(rule4);
concise.DynGenPar.Rule rule5 = new concise.DynGenPar.Rule();
rule5.add('c');
aRules.add(rule5);
rules.put('A', aRules);
parser.setRules(rules);
parser.setStartCat('S');

// Whenever we set the grammar directly, we have to call this function.
parser.initCaches();

// And here, we specify the sample input, also hardcoded.
java.util.List<Integer> inputTokens = new java.util.ArrayList<Integer>();
inputTokens.add('b');
inputTokens.add('a');
tokenSource.setInputTokens(inputTokens);

// Now we parse the input, using a default parse state.
concise.DynGenPar.ParseState parseState
  = new concise.DynGenPar.ParseState();
java.util.List<concise.DynGenPar.Match> matches = parser.parse(parseState);
```

The DynGenPar Java bindings are used in the `FMathL Concise` environment. [2]

## 1.4 Features

This section lists the advanced features supported by DynGenPar.

### 1.4.1 Dynamic Grammar Additions

The most unique feature of DynGenPar is the possibility to add rules to the grammar at almost any time, even during parsing. See DynGenPar::Parser::addRule.

### 1.4.2 Incremental Parsing

DynGenPar allows operating on its input incrementally, parsing interactively as input is produced and remembering its state. See DynGenPar::Parser::parse and DynGenPar::ParseState.

It is also possible to go back by resuming at a previous saved state.

### 1.4.3 Prediction

Possible continuations, i.e. tokens which can come next, can be predicted at any saved incremental parsing state, see DynGenPar::Parser::computePredictions and DynGenPar::Parser::computeConstrainedPredictions.

It is also possible to predict entire "literals", which are sequences of tokens appearing sequentially within a rule, see DynGenPar::Parser::computeMultiPredictions and DynGenPar::Parser::computeConstrainedMultiPredictions. This is particularly useful for scannerless parsing.

### 1.4.4 Rule Labels

CFG rules can have labels, which are reproduced in the produced parse tree. Those labels can be anything which can be stored in a QVariant, including strings, integers and pointers to arbitrary objects. See DynGenPar::Rule::Rule, DynGenPar::Rule::label and DynGenPar::Rule::setLabel.

### 1.4.5 Custom Parse Actions

It is also possible to attach an action to a rule, which will be executed when the rule is matched. Note that actions will currently only be executed for nonempty matches (and in particular, actions on epsilon rules will always be ignored) and that an action will be executed even if the detected match only appears in some of the considered parses and is later discarded due to the input that follows. An action must be a subclass of DynGenPar::Action implementing the pure virtual DynGenPar::Action::execute method. See DynGenPar::Action and DynGenPar::Rule::action.

### 1.4.6 Arbitrary Token Sources

DynGenPar accepts tokens from any source implementing the DynGenPar::TokenSource interface. Your class only has to derive from DynGenPar::TokenSource and implement the needed virtual methods, at least the pure virtual DynGenPar::TokenSource::readToken.

A number of ready-made token sources are also provided. See DynGenPar::ListTokenSource, DynGenPar::ByteTokenSource and DynGenPar::TextByteTokenSource.

#### 1.4.6.1 Flex Lexer Token Source

In particular, it is possible to use a lexer generated by Flex as a token source. See DynGenPar::FlexLexerTokenSource.

### 1.4.7 Hierarchical Parsing

A token source can return, along with a token, a complete parse (sub)tree to use in lieu of a leaf node in the resulting parse tree, making it possible to call, from your token source, another parser, or another instance of DynGenPar (which is fully reentrant), and to return the result as a single token for the higher-level grammar. See DynGenPar::TokenSource::tree.

### 1.4.8 Parallel Multiple Context-Free Grammars (PMCFGs)

PMCFGs (Parallel Multiple Context-Free Grammars) are an extension of context-free grammars used for natural language, e.g. by the Grammatical Framework GF. They are natively supported by DynGenPar. See DynGenPar::Pmcfg, DynGenPar::Parser::loadPmcfg, DynGenPar::Parser::addPmcfgRule and DynGenPar::parseTreeToPmcfgSyntaxTree.

#### 1.4.8.1 Grammatical Framework (GF) PGF Grammars

In particular, DynGenPar can import compiled grammars produced by the GF compiler, in the PGF (Portable Grammatical Framework) file format, automatically converting them to PMCFGs in standard form and providing a GF-compatible lexer. See DynGenPar::Pgf and DynGenPar::PgfParser.

### 1.4.9 Next Token Constraints

Rules can have constraints attached that require (expect) or forbid (taboo) certain tokens following the rule. This is particularly useful for scannerless parsing, where it allows one to implement the usual context-sensitive scannerless parsing primitives such as maximal matches. It can also be used to enforce grammatically correct word sequences, e.g. for the singular indefinite article ("a" vs. "an") in English. See DynGenPar::NextTokenConstraints and DynGenPar::Rule::nextTokenConstraints.

## 1.5 TODO

The following desirable features are currently not supported:

- some PMCFGs with infinitely ambiguous context-free approximations
- general context-sensitive constraints on rules (except PMCFG, next token)
- triggering parse actions on empty matches (including epsilon rules)
- error correction (we only have basic error detection)

(These features may or may not turn out to be required in practice.)

It is planned to add support for these features to the algorithm as needed, without changing the basic structure.

## 1.6 Copyright

### 1.6.1 Acknowledgments

Support by the Austrian Science Fund FWF under contract numbers P20631, P23554 and P22239 is gratefully acknowledged.

### 1.6.2 Licensing and Warranty Disclaimer



### 1.6.3 Citations

The publication [1] constitutes a suitable citation for academic works. Though not required by the software license, it is kindly requested to include that citation in any research papers using this program.

# Chapter 2

# Namespace Index

## 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 3

# Hierarchical Index

## 3.1 Class Hierarchy

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

# Chapter 4

# Class Index

## 4.1 Class List

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

# Chapter 5

# File Index

## 5.1 File List

Here is a list of all files with brief descriptions:

# Chapter 6

# Namespace Documentation

## 6.1 DynGenPar Namespace Reference

### Classes

- class AbstractLexerStateData

  *API for stateful lexers to save their state for rewinding.*
- class Action

  *interface for parser actions*
- class ActionDeserializer

  *interface for parser action deserializers*
- struct ActionInfo

  *data passed to parser actions*
- class Alternative
- class ByteTokenSource
- struct Cfg

  *An object representing a CFG (or a PMCFG in our internal representation)*
- struct ConstrainedMultiPrediction

  *multi-token predictions with next token constraints*
- class FlexLexerTokenSource
- struct FullRule

  *full rule as found in the initial graph*
- class Function

  *PMCFG function.*
- class LexerState
- class ListTokenSource
- struct Match

  *(possibly partial) match*
- struct MultiPrediction

  *multi-token predictions*
- struct NextTokenConstraints

  *rule constraints affecting the next token, for scannerless parsing*
- struct Node

  *node in the parse tree*
- class Parser

  *main class*

- struct ParseState

  *parse state struct, for bindings*
- struct Pgf

  *representation of the information in .pgf files in a format we can process*
- class PgfParser

  *parser for PGF grammars*
- struct Pmcfg

  *PMCFG.*
- struct PmcfgComponentInfo

  *attached to the parse trees as rule labels to allow obtaining syntax trees*
- class PseudoCatScope
- class PseudoCatScopeData
- class Rule
- class Sequence

  *component of a PMCFG function, a sequence of terms*
- class StackItem
- class StackItemData
- class StackItemType0

  *type 0 item: during match, we're waiting for a token to shift*
- class StackItemType1

  *type 1 item: during type 0 item processing, we're executing a reduce*
- class StackItemType2

  *type 2 item: during reduce, we're recursively executing another reduce*
- class StackItemType3

  *type 3 item: during matchRemaining, we're executing a match*
- class StackItemType4

  *type 4 item: during reduce, we're executing a matchRemaining*
- class StackItemType5

  *type 5 item: during match (of an MCFG constraint), we're executing a matchRemaining*
- class StackItemType6

  *type 6 item: during match, we're matching a P constraint*
- struct Term

  *term in the expression of a component of a PMCFG function*
- class TextByteLexerStateData

  *You should not have to use this class directly, ever.*
- class TextByteTokenSource
- struct TextPosition

  *text position*
- class TokenSource

## Typedefs

- typedef QString Cat

  *Category type: string or integer depending on DYNGENPAR_INTEGER_CATEGORIES.*
- typedef const Cat & CatArg

  *Category type (string or integer) when passed as an argument.*
- typedef QHash< Cat, QList< Rule > > RuleSet
- typedef QSet< Cat > TokenSet
- typedef QSet< Cat > Predictions
- typedef QMultiHash< QList< Cat >, MultiPrediction > MultiPredictions
- typedef QMultiHash< Cat, NextTokenConstraints > ConstrainedPredictions
- typedef QMultiHash< QList< Cat >, ConstrainedMultiPrediction > ConstrainedMultiPredictions

### Enumerations

- enum PgfReservedTokens {
  PgfTokenEpsilon, PgfTokenLexError, PgfTokenVar, PgfTokenFloat,
  PgfTokenInt, PgfTokenString }

### Functions

- uint qHash (const NextTokenConstraints &nextTokenConstraints)
  *simple hash function for next token constraints*
- Node parseTreeToPmcfgSyntaxTree (const Node &parseTree)
  *converts a parse tree obtained from a PMCFG to a PMCFG syntax tree*
- uint qHash (const PseudoCatScope &scope)

### Variables

- STATIC const char *const PreludeBind = "&+"

## 6.1.1 Typedef Documentation

#### 6.1.1.1 Cat

```
typedef QString DynGenPar::Cat
```

Category type: string or integer depending on DYNGENPAR_INTEGER_CATEGORIES.

If DYNGENPAR_INTEGER_CATEGORIES is defined, this typedef is defined as:

```
typedef int Cat;
```

instead.

Definition at line 71 of file dyngenpar.h.

#### 6.1.1.2 CatArg

```
typedef const Cat& DynGenPar::CatArg
```

Category type (string or integer) when passed as an argument.

If DYNGENPAR_INTEGER_CATEGORIES is defined, this typedef is defined as:

```
typedef int CatArg;
```

instead.

This typedef is used for maximum efficiency, to pass strings by reference, but integers by value.

Definition at line 83 of file dyngenpar.h.

#### 6.1.1.3 ConstrainedMultiPredictions

```
typedef QMultiHash<QList<Cat>, ConstrainedMultiPrediction> DynGenPar::ConstrainedMulti↩
Predictions
```

Definition at line 258 of file dyngenpar.h.

#### 6.1.1.4 ConstrainedPredictions

```
typedef QMultiHash<Cat, NextTokenConstraints> DynGenPar::ConstrainedPredictions
```

Definition at line 233 of file dyngenpar.h.

#### 6.1.1.5 MultiPredictions

```
typedef QMultiHash<QList<Cat>, MultiPrediction> DynGenPar::MultiPredictions
```

Definition at line 232 of file dyngenpar.h.

#### 6.1.1.6 Predictions

```
typedef QSet<Cat> DynGenPar::Predictions
```

Definition at line 214 of file dyngenpar.h.

#### 6.1.1.7 RuleSet

```
typedef QHash<Cat, QList<Rule> > DynGenPar::RuleSet
```

Definition at line 186 of file dyngenpar.h.

#### 6.1.1.8 TokenSet

```
typedef QSet<Cat> DynGenPar::TokenSet
```

Definition at line 187 of file dyngenpar.h.

### 6.1.2 Enumeration Type Documentation

#### 6.1.2.1 PgfReservedTokens

```
enum DynGenPar::PgfReservedTokens
```

**Enumerator**

| | |
|---|---|
| PgfTokenEpsilon | |
| PgfTokenLexError | |
| PgfTokenVar | |
| PgfTokenFloat | |
| PgfTokenInt | |
| PgfTokenString | |

Definition at line 41 of file pgf.h.

### 6.1.3 Function Documentation

#### 6.1.3.1 parseTreeToPmcfgSyntaxTree()

```
Node DynGenPar::parseTreeToPmcfgSyntaxTree (
            const Node & parseTree )
```

converts a parse tree obtained from a PMCFG to a PMCFG syntax tree

Definition at line 539 of file dyngenpar.cpp.

#### 6.1.3.2 qHash() [1/2]

```
uint DynGenPar::qHash (
            const PseudoCatScope & scope )  [inline]
```

Definition at line 392 of file dyngenpar.h.

#### 6.1.3.3 qHash() [2/2]

```
uint DynGenPar::qHash (
            const NextTokenConstraints & nextTokenConstraints )
```

simple hash function for next token constraints

Definition at line 452 of file dyngenpar.cpp.

### 6.1.4 Variable Documentation

#### 6.1.4.1 PreludeBind

```
STATIC const char* const DynGenPar::PreludeBind = "&+"
```

Definition at line 48 of file pgf.h.

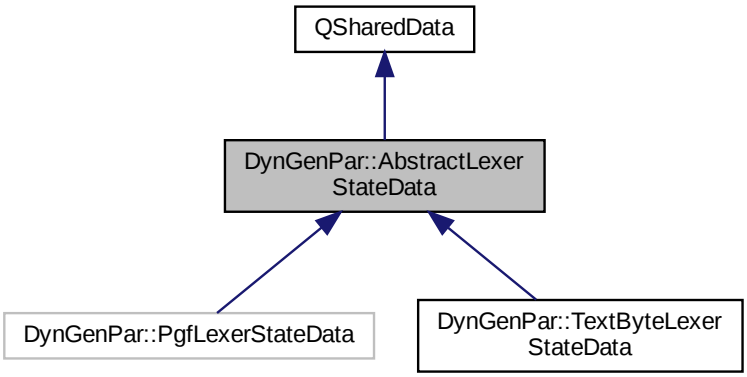
# Chapter 7

# Class Documentation

## 7.1 DynGenPar::AbstractLexerStateData Class Reference

API for stateful lexers to save their state for rewinding.

Inheritance diagram for DynGenPar::AbstractLexerStateData:




### Public Member Functions

- virtual ~AbstractLexerStateData ()
- virtual AbstractLexerStateData * clone ()=0

### 7.1.1 Detailed Description

API for stateful lexers to save their state for rewinding.

Definition at line 793 of file dyngenpar.h.

### 7.1.2 Constructor & Destructor Documentation

#### 7.1.2.1 ~AbstractLexerStateData()

```
virtual DynGenPar::AbstractLexerStateData::~AbstractLexerStateData ( )  [inline], [virtual]
```

Definition at line 795 of file dyngenpar.h.

### 7.1.3 Member Function Documentation

#### 7.1.3.1 clone()

```
virtual AbstractLexerStateData* DynGenPar::AbstractLexerStateData::clone ( )  [pure virtual]
```

Implemented in DynGenPar::TextByteLexerStateData.

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

- dyngenpar.h

## 7.2 DynGenPar::Action Class Reference

interface for parser actions

### Public Member Functions

- virtual ~Action ()
- virtual void execute (const ActionInfo &info)=0
- virtual int lookaheadTokens () const

  *the number of tokens to look ahead before deciding to execute the action*
- virtual void writeExternal (QDataStream &stream) const

  *implementation of the QDataStream operator<<*

### Static Public Member Functions

- static QDataStream & constructAndRead (QDataStream &stream, Action *&data)

  *implementation of the QDataStream operator>>*

### Static Protected Member Functions

- static void registerDeserializer (const QString &name, ActionDeserializer *deserializer)

### 7.2.1 Detailed Description

interface for parser actions

Definition at line 433 of file dyngenpar.h.

### 7.2.2 Constructor & Destructor Documentation

#### 7.2.2.1 ~Action()

```
virtual DynGenPar::Action::~Action ( )  [inline], [virtual]
```

Definition at line 442 of file dyngenpar.h.

### 7.2.3 Member Function Documentation

#### 7.2.3.1 constructAndRead()

```
static QDataStream& DynGenPar::Action::constructAndRead (
            QDataStream & stream,
            Action *& data )  [inline], [static]
```

implementation of the QDataStream operator>>

Definition at line 454 of file dyngenpar.h.

#### 7.2.3.2 execute()

```
virtual void DynGenPar::Action::execute (
            const ActionInfo & info )  [pure virtual]
```

#### 7.2.3.3 lookaheadTokens()

```
virtual int DynGenPar::Action::lookaheadTokens ( ) const  [inline], [virtual]
```

the number of tokens to look ahead before deciding to execute the action

defaults to 0, can be overridden by subclasses

Definition at line 447 of file dyngenpar.h.

#### 7.2.3.4 registerDeserializer()

```
static void DynGenPar::Action::registerDeserializer (
            const QString & name,
            ActionDeserializer * deserializer )  [inline], [static], [protected]
```

Definition at line 437 of file dyngenpar.h.

#### 7.2.3.5 writeExternal()

```
virtual void DynGenPar::Action::writeExternal (
            QDataStream & stream ) const  [inline], [virtual]
```

implementation of the QDataStream operator<<

Definition at line 449 of file dyngenpar.h.

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

- dyngenpar.h
- dyngenpar.cpp

## 7.3 DynGenPar::ActionDeserializer Class Reference

interface for parser action deserializers

### Public Member Functions

- virtual ~ActionDeserializer ()
- virtual Action * readAction (QDataStream &)=0

### 7.3.1 Detailed Description

interface for parser action deserializers

Definition at line 425 of file dyngenpar.h.

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 ~ActionDeserializer()

```
virtual DynGenPar::ActionDeserializer::~ActionDeserializer ( )  [inline], [virtual]
```

Definition at line 427 of file dyngenpar.h.

### 7.3.3 Member Function Documentation

#### 7.3.3.1 readAction()

```
virtual Action* DynGenPar::ActionDeserializer::readAction (
            QDataStream &  )  [pure virtual]
```

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

- dyngenpar.h

## 7.4 DynGenPar::ActionInfo Struct Reference

data passed to parser actions

### Public Member Functions

- ActionInfo ()
  *dummy default constructor for bindings*
- ActionInfo (const Node &t, Parser *p)

### Public Attributes

- Node tree
- Parser * parser

### 7.4.1 Detailed Description

data passed to parser actions

Definition at line 415 of file dyngenpar.h.

### 7.4.2 Constructor & Destructor Documentation

#### 7.4.2.1 ActionInfo() [1/2]

```
DynGenPar::ActionInfo::ActionInfo ( )  [inline]
```

dummy default constructor for bindings

Definition at line 417 of file dyngenpar.h.

#### 7.4.2.2 ActionInfo() [2/2]

```
DynGenPar::ActionInfo::ActionInfo (
            const Node & t,
            Parser * p )  [inline]
```

Definition at line 418 of file dyngenpar.h.

### 7.4.3 Member Data Documentation

#### 7.4.3.1 parser

```
Parser* DynGenPar::ActionInfo::parser
```

Definition at line 420 of file dyngenpar.h.

#### 7.4.3.2 tree

```
Node DynGenPar::ActionInfo::tree
```

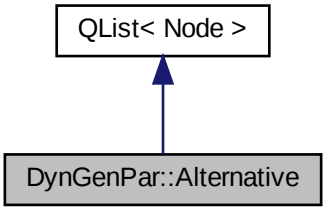
Definition at line 419 of file dyngenpar.h.

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

- dyngenpar.h

## 7.5 DynGenPar::Alternative Class Reference

Inheritance diagram for DynGenPar::Alternative:




### Public Member Functions

- Alternative ()
- Alternative (const QVariant &label)
- Alternative (const QList< DynGenPar::Node > &list)
- Alternative (const QList< DynGenPar::Node > &list, const QVariant &label)
- QVariant label () const
- void setLabel (const QVariant &label)
- Alternative & operator+= (const QList< DynGenPar::Node > &other)
- Alternative & operator+= (const DynGenPar::Node &value)
- Alternative & operator<< (const QList< DynGenPar::Node > &other)
- Alternative & operator<< (const DynGenPar::Node &value)
- void add (const DynGenPar::Node &value)

  *Java-style + the binding generator doesn't detect the inherited append.*
- QList< DynGenPar::Node > & toList ()

  *for bindings*

### 7.5.1 Detailed Description

Definition at line 278 of file dyngenpar.h.

### 7.5.2 Constructor & Destructor Documentation

#### 7.5.2.1 Alternative() [1/4]

```
DynGenPar::Alternative::Alternative ( )  [inline]
```

Definition at line 280 of file dyngenpar.h.

#### 7.5.2.2 Alternative() [2/4]

```
DynGenPar::Alternative::Alternative (
            const QVariant & label )  [inline], [explicit]
```

Definition at line 281 of file dyngenpar.h.

#### 7.5.2.3 Alternative() [3/4]

```
DynGenPar::Alternative::Alternative (
            const QList< DynGenPar::Node > & list )  [inline], [explicit]
```

Definition at line 283 of file dyngenpar.h.

#### 7.5.2.4 Alternative() [4/4]

```
DynGenPar::Alternative::Alternative (
            const QList< DynGenPar::Node > & list,
            const QVariant & label )  [inline]
```

Definition at line 285 of file dyngenpar.h.

### 7.5.3 Member Function Documentation

#### 7.5.3.1 add()

```
void DynGenPar::Alternative::add (
            const DynGenPar::Node & value )  [inline]
```

Java-style + the binding generator doesn't detect the inherited append.

Definition at line 306 of file dyngenpar.h.

#### 7.5.3.2 label()

```
QVariant DynGenPar::Alternative::label ( ) const  [inline]
```

Definition at line 287 of file dyngenpar.h.

#### 7.5.3.3 operator+=() [1/2]

```
Alternative& DynGenPar::Alternative::operator+= (
            const QList< DynGenPar::Node > & other )  [inline]
```

Definition at line 289 of file dyngenpar.h.

#### 7.5.3.4 operator+=() [2/2]

```
Alternative& DynGenPar::Alternative::operator+= (
            const DynGenPar::Node & value )  [inline]
```

Definition at line 293 of file dyngenpar.h.

#### 7.5.3.5 operator<<() [1/2]

```
Alternative& DynGenPar::Alternative::operator<< (
            const QList< DynGenPar::Node > & other )  [inline]
```

Definition at line 297 of file dyngenpar.h.

#### 7.5.3.6 operator<<() [2/2]

```
Alternative& DynGenPar::Alternative::operator<< (
            const DynGenPar::Node & value )  [inline]
```

Definition at line 301 of file dyngenpar.h.

#### 7.5.3.7 setLabel()

```
void DynGenPar::Alternative::setLabel (
            const QVariant & label )  [inline]
```

Definition at line 288 of file dyngenpar.h.

#### 7.5.3.8 toList()

```
QList<DynGenPar::Node>& DynGenPar::Alternative::toList ( )  [inline]
```

for bindings

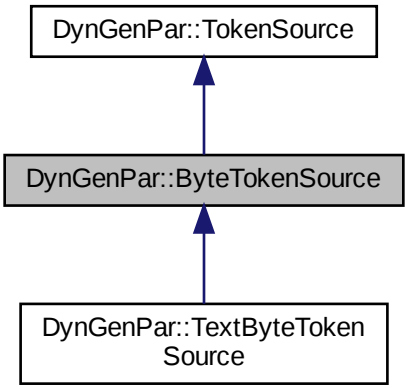
Definition at line 310 of file dyngenpar.h.

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

- dyngenpar.h

## 7.6 DynGenPar::ByteTokenSource Class Reference

Inheritance diagram for DynGenPar::ByteTokenSource:




### Public Member Functions

- ByteTokenSource ()
- ByteTokenSource (const QString &fileName)
- virtual ~ByteTokenSource ()
- virtual bool rewindTo (int pos, const LexerState &=LexerState())

  *rewind to an older position (requires buffering)*
- void setInputStdin ()
- void setInputFile (const QString &fileName)
- void setInputBytes (const QByteArray &bytes)
- void setInputString (const QString &string)
- void setInputBuffer (QByteArray *buffer)

### Protected Member Functions

- virtual Cat readToken ()

  *get the next token from the input, to be implemented by subclasses*
- virtual void reset ()

### Protected Attributes

- QIODevice * stream

## 7.6.1 Detailed Description

Definition at line 44 of file bytetokensource.h.

## 7.6.2 Constructor & Destructor Documentation

### 7.6.2.1 ByteTokenSource() [1/2]

```
DynGenPar::ByteTokenSource::ByteTokenSource ( )  [inline]
```

Definition at line 46 of file bytetokensource.h.

### 7.6.2.2 ByteTokenSource() [2/2]

```
DynGenPar::ByteTokenSource::ByteTokenSource (
            const QString & fileName )  [inline]
```

Definition at line 49 of file bytetokensource.h.

### 7.6.2.3 ~ByteTokenSource()

```
virtual DynGenPar::ByteTokenSource::~ByteTokenSource ( )  [inline], [virtual]
```

Definition at line 53 of file bytetokensource.h.

## 7.6.3 Member Function Documentation

#### 7.6.3.1 readToken()

```
virtual Cat DynGenPar::ByteTokenSource::readToken ( )  [inline], [protected], [virtual]
```

get the next token from the input, to be implemented by subclasses

Implements DynGenPar::TokenSource.

Reimplemented in DynGenPar::TextByteTokenSource.

Definition at line 95 of file bytetokensource.h.

#### 7.6.3.2 reset()

```
virtual void DynGenPar::ByteTokenSource::reset ( )  [inline], [protected], [virtual]
```

Reimplemented in DynGenPar::TextByteTokenSource.

Definition at line 102 of file bytetokensource.h.

#### 7.6.3.3 rewindTo()

```
virtual bool DynGenPar::ByteTokenSource::rewindTo (
            int pos,
            const LexerState &  = LexerState() )  [inline], [virtual]
```

rewind to an older position (requires buffering)

**Returns**

`true` if successful, `false` otherwise

in all cases, destroys the saved parse tree

By default, only succeeds if the position is the current one, otherwise always returns `false`. Can be overridden by subclasses.

Some subclasses (e.g., file-backed ones) can wind both forward and backward, some can only rewind, some cannot wind at all. The method will return `false` if seeking to the new position is not possible.

Reimplemented from DynGenPar::TokenSource.

Reimplemented in DynGenPar::TextByteTokenSource.

Definition at line 54 of file bytetokensource.h.

#### 7.6.3.4 setInputBuffer()

```
void DynGenPar::ByteTokenSource::setInputBuffer (
            QByteArray * buffer )  [inline]
```

Definition at line 88 of file bytetokensource.h.

#### 7.6.3.5 setInputBytes()

```
void DynGenPar::ByteTokenSource::setInputBytes (
            const QByteArray & bytes )  [inline]
```

Definition at line 78 of file bytetokensource.h.

#### 7.6.3.6 setInputFile()

```
void DynGenPar::ByteTokenSource::setInputFile (
            const QString & fileName )  [inline]
```

Definition at line 72 of file bytetokensource.h.

#### 7.6.3.7 setInputStdin()

```
void DynGenPar::ByteTokenSource::setInputStdin ( )  [inline]
```

Definition at line 66 of file bytetokensource.h.

#### 7.6.3.8 setInputString()

```
void DynGenPar::ByteTokenSource::setInputString (
            const QString & string )  [inline]
```

Definition at line 85 of file bytetokensource.h.

### 7.6.4 Member Data Documentation

#### 7.6.4.1 stream

```
QIODevice* DynGenPar::ByteTokenSource::stream  [protected]
```

Definition at line 101 of file bytetokensource.h.

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

- bytetokensource.h

## 7.7 DynGenPar::Cfg Struct Reference

An object representing a CFG (or a PMCFG in our internal representation)

### Public Member Functions

- Cfg ()
  *dummy default constructor for bindings*
- Cfg (const RuleSet &r, const TokenSet &t, CatArg sc)
- bool isToken (CatArg cat) const
- void addToken (CatArg cat)
- QDataStream & writeExternal (QDataStream &stream) const
  *implementation of the QDataStream operator<<*
- QDataStream & readExternal (QDataStream &stream)
  *implementation of the QDataStream operator>>*

### Public Attributes

- RuleSet rules
- TokenSet tokens
- Cat startCat

### 7.7.1 Detailed Description

An object representing a CFG (or a PMCFG in our internal representation)

This allows passing it around more easily and loading it into the parser in one step.

Definition at line 193 of file dyngenpar.h.

### 7.7.2 Constructor & Destructor Documentation

#### 7.7.2.1 Cfg() [1/2]

```
DynGenPar::Cfg::Cfg ( )  [inline]
```

dummy default constructor for bindings

Definition at line 195 of file dyngenpar.h.

#### 7.7.2.2 Cfg() [2/2]

```
DynGenPar::Cfg::Cfg (
            const RuleSet & r,
            const TokenSet & t,
            CatArg sc )  [inline]
```

Definition at line 196 of file dyngenpar.h.

### 7.7.3 Member Function Documentation

#### 7.7.3.1 addToken()

```
void DynGenPar::Cfg::addToken (
            CatArg cat )  [inline]
```

Definition at line 202 of file dyngenpar.h.

#### 7.7.3.2 isToken()

```
bool DynGenPar::Cfg::isToken (
            CatArg cat ) const  [inline]
```

Definition at line 201 of file dyngenpar.h.

#### 7.7.3.3 readExternal()

```
QDataStream& DynGenPar::Cfg::readExternal (
            QDataStream & stream )  [inline]
```

implementation of the QDataStream operator>>

Definition at line 208 of file dyngenpar.h.

#### 7.7.3.4 writeExternal()

```
QDataStream& DynGenPar::Cfg::writeExternal (
            QDataStream & stream ) const  [inline]
```

implementation of the QDataStream operator<<

Definition at line 204 of file dyngenpar.h.

### 7.7.4 Member Data Documentation

#### 7.7.4.1 rules

```
RuleSet DynGenPar::Cfg::rules
```

Definition at line 198 of file dyngenpar.h.

#### 7.7.4.2 startCat

```
Cat DynGenPar::Cfg::startCat
```

Definition at line 200 of file dyngenpar.h.

#### 7.7.4.3 tokens

```
TokenSet DynGenPar::Cfg::tokens
```

Definition at line 199 of file dyngenpar.h.

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

- dyngenpar.h

## 7.8 DynGenPar::ConstrainedMultiPrediction Struct Reference

multi-token predictions with next token constraints

### Public Member Functions

- ConstrainedMultiPrediction ()

  *dummy default constructor for bindings*
- ConstrainedMultiPrediction (const QList< Cat > &fullLit, CatArg c)
- ConstrainedMultiPrediction (const QList< Cat > &fullLit, CatArg c, NextTokenConstraints ntc)
- bool operator== (const ConstrainedMultiPrediction &other) const

  *needed for QList, QMultiHash*

### Public Attributes

- QList< Cat > fullLiteral

  *the entire literal completed by the prediction*
- Cat cat

  *the nonterminal generating the literal / the nonterminal itself*
- NextTokenConstraints nextTokenConstraints

  *only for nonterminals*

## 7.8.1 Detailed Description

multi-token predictions with next token constraints

Definition at line 236 of file dyngenpar.h.

## 7.8.2 Constructor & Destructor Documentation

### 7.8.2.1 ConstrainedMultiPrediction() [1/3]

```
DynGenPar::ConstrainedMultiPrediction::ConstrainedMultiPrediction ( )  [inline]
```

dummy default constructor for bindings

Definition at line 238 of file dyngenpar.h.

### 7.8.2.2 ConstrainedMultiPrediction() [2/3]

```
DynGenPar::ConstrainedMultiPrediction::ConstrainedMultiPrediction (
            const QList< Cat > & fullLit,
            CatArg c )  [inline]
```

Definition at line 240 of file dyngenpar.h.

#### 7.8.2.3 ConstrainedMultiPrediction() [3/3]

```
DynGenPar::ConstrainedMultiPrediction::ConstrainedMultiPrediction (
            const QList< Cat > & fullLit,
            CatArg c,
            NextTokenConstraints ntc )  [inline]
```

Definition at line 242 of file dyngenpar.h.

### 7.8.3 Member Function Documentation

#### 7.8.3.1 operator==()

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

needed for QList, QMultiHash

Definition at line 250 of file dyngenpar.h.

### 7.8.4 Member Data Documentation

#### 7.8.4.1 cat

```
Cat DynGenPar::ConstrainedMultiPrediction::cat
```

the nonterminal generating the literal / the nonterminal itself

Definition at line 246 of file dyngenpar.h.

#### 7.8.4.2 fullLiteral

```
QList<Cat> DynGenPar::ConstrainedMultiPrediction::fullLiteral
```

the entire literal completed by the prediction

Definition at line 245 of file dyngenpar.h.

#### 7.8.4.3 nextTokenConstraints

```
NextTokenConstraints DynGenPar::ConstrainedMultiPrediction::nextTokenConstraints
```

only for nonterminals

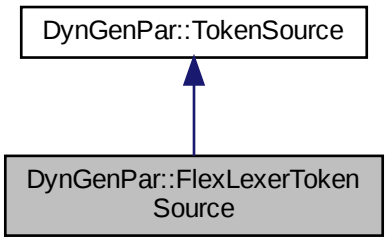
Definition at line 247 of file dyngenpar.h.

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

- dyngenpar.h

## 7.9 DynGenPar::FlexLexerTokenSource Class Reference

Inheritance diagram for DynGenPar::FlexLexerTokenSource:




### Public Member Functions

- FlexLexerTokenSource (FlexLexer ∗lexer)
- virtual ∼FlexLexerTokenSource ()

### Protected Member Functions

- virtual Cat readToken ()

  *get the next token from the input, to be implemented by subclasses*

### Protected Attributes

- FlexLexer ∗ flexLexer

### 7.9.1 Detailed Description

Definition at line 31 of file flexlexertokensource.h.

### 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 FlexLexerTokenSource()

```
DynGenPar::FlexLexerTokenSource::FlexLexerTokenSource (
            FlexLexer * lexer )  [inline]
```

Definition at line 33 of file flexlexertokensource.h.

#### 7.9.2.2 ~FlexLexerTokenSource()

```
virtual DynGenPar::FlexLexerTokenSource::~FlexLexerTokenSource ( )  [inline], [virtual]
```

Definition at line 34 of file flexlexertokensource.h.

### 7.9.3 Member Function Documentation

#### 7.9.3.1 readToken()

```
virtual Cat DynGenPar::FlexLexerTokenSource::readToken ( )  [inline], [protected], [virtual]
```

get the next token from the input, to be implemented by subclasses

Implements DynGenPar::TokenSource.

Definition at line 36 of file flexlexertokensource.h.

### 7.9.4 Member Data Documentation

#### 7.9.4.1 flexLexer

```
FlexLexer* DynGenPar::FlexLexerTokenSource::flexLexer  [protected]
```

Definition at line 39 of file flexlexertokensource.h.

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

- flexlexertokensource.h

## 7.10 DynGenPar::FullRule Struct Reference

full rule as found in the initial graph

### Public Member Functions

- FullRule ()
  *dummy default constructor for bindings*
- FullRule (CatArg c, const Rule &r, int epsSkipped, int n)

### Public Attributes

- Cat cat
- Rule rule
- int epsilonsSkipped
- int ruleno
  *needed for PMCFGs (to match components of rules to each other)*

### 7.10.1 Detailed Description

full rule as found in the initial graph

Definition at line 261 of file dyngenpar.h.

### 7.10.2 Constructor & Destructor Documentation

#### 7.10.2.1 FullRule() [1/2]

```
DynGenPar::FullRule::FullRule ( )  [inline]
```

dummy default constructor for bindings

Definition at line 263 of file dyngenpar.h.

#### 7.10.2.2 FullRule() [2/2]

```
DynGenPar::FullRule::FullRule (
            CatArg c,
            const Rule & r,
            int epsSkipped,
            int n )  [inline]
```

Definition at line 264 of file dyngenpar.h.

### 7.10.3 Member Data Documentation

#### 7.10.3.1 cat

```
Cat DynGenPar::FullRule::cat
```

Definition at line 266 of file dyngenpar.h.

#### 7.10.3.2 epsilonsSkipped

```
int DynGenPar::FullRule::epsilonsSkipped
```

Definition at line 268 of file dyngenpar.h.

#### 7.10.3.3 rule

```
Rule DynGenPar::FullRule::rule
```

Definition at line 267 of file dyngenpar.h.

#### 7.10.3.4 ruleno

```
int DynGenPar::FullRule::ruleno
```

needed for PMCFGs (to match components of rules to each other)

always set to 0 for context-free (1-dimensional) categories

Definition at line 272 of file dyngenpar.h.

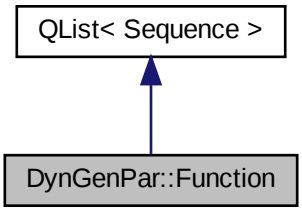
The documentation for this struct was generated from the following file:

- dyngenpar.h

## 7.11 DynGenPar::Function Class Reference

PMCFG function.

Inheritance diagram for DynGenPar::Function:




### Public Member Functions

- Function ()
- Function (const QList< Sequence > &list)
- Function & operator+= (const QList< Sequence > &other)
- Function & operator+= (const Sequence &value)
- Function & operator<< (const QList< Sequence > &other)
- Function & operator<< (const Sequence &value)
- void add (const Sequence &value)

  *Java-style (for consistency, even though append is detected here)*
- QList< Sequence > & toList ()

  *for bindings*

### 7.11.1 Detailed Description

PMCFG function.

Definition at line 1015 of file dyngenpar.h.

### 7.11.2 Constructor & Destructor Documentation

#### 7.11.2.1 Function() [1/2]

```
DynGenPar::Function::Function ( )  [inline]
```

Definition at line 1017 of file dyngenpar.h.

#### 7.11.2.2 Function() [2/2]

```
DynGenPar::Function::Function (
            const QList< Sequence > & list )  [inline], [explicit]
```

Definition at line 1018 of file dyngenpar.h.

### 7.11.3 Member Function Documentation

#### 7.11.3.1 add()

```
void DynGenPar::Function::add (
            const Sequence & value )  [inline]
```

Java-style (for consistency, even though append is detected here)

Definition at line 1036 of file dyngenpar.h.

#### 7.11.3.2 operator+=() [1/2]

```
Function& DynGenPar::Function::operator+= (
            const QList< Sequence > & other )  [inline]
```

Definition at line 1019 of file dyngenpar.h.

#### 7.11.3.3 operator+=() [2/2]

```
Function& DynGenPar::Function::operator+= (
            const Sequence & value )  [inline]
```

Definition at line 1023 of file dyngenpar.h.

#### 7.11.3.4 operator<<() [1/2]

```
Function& DynGenPar::Function::operator<< (
            const QList< Sequence > & other )  [inline]
```

Definition at line 1027 of file dyngenpar.h.

#### 7.11.3.5 operator<<() [2/2]

```
Function& DynGenPar::Function::operator<< (
            const Sequence & value )  [inline]
```

Definition at line 1031 of file dyngenpar.h.

#### 7.11.3.6 toList()

```
QList<Sequence>& DynGenPar::Function::toList ( )  [inline]
```

for bindings

Definition at line 1040 of file dyngenpar.h.

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

- dyngenpar.h

## 7.12 DynGenPar::LexerState Class Reference

### Public Member Functions

- LexerState ()
- LexerState (AbstractLexerStateData *data)
- void clear ()
- bool isNull () const
- const AbstractLexerStateData * data () const
- bool operator== (const LexerState &other) const

### 7.12.1 Detailed Description

Definition at line 800 of file dyngenpar.h.

### 7.12.2 Constructor & Destructor Documentation

#### 7.12.2.1 LexerState() [1/2]

```
DynGenPar::LexerState::LexerState ( )  [inline]
```

Definition at line 802 of file dyngenpar.h.

#### 7.12.2.2 LexerState() [2/2]

```
DynGenPar::LexerState::LexerState (
            AbstractLexerStateData * data )  [inline]
```

Definition at line 803 of file dyngenpar.h.

### 7.12.3 Member Function Documentation

#### 7.12.3.1 clear()

```
void DynGenPar::LexerState::clear ( )  [inline]
```

Definition at line 804 of file dyngenpar.h.

#### 7.12.3.2 data()

```
const AbstractLexerStateData* DynGenPar::LexerState::data ( ) const  [inline]
```

Definition at line 806 of file dyngenpar.h.

#### 7.12.3.3 isNull()

```
bool DynGenPar::LexerState::isNull ( ) const  [inline]
```

Definition at line 805 of file dyngenpar.h.

#### 7.12.3.4 operator==()

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

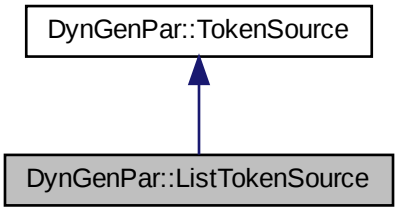
Definition at line 807 of file dyngenpar.h.

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

- dyngenpar.h

## 7.13 DynGenPar::ListTokenSource Class Reference

Inheritance diagram for DynGenPar::ListTokenSource:




### Public Member Functions

- ListTokenSource ()
- virtual ~ListTokenSource ()
- virtual bool rewindTo (int pos, const LexerState &=LexerState())

  *overridden because lists can be rewound*

### Public Attributes

- QList< Cat > inputTokens

### Protected Member Functions

- virtual Cat readToken ()

  *just fetch the next token from the list*

### Additional Inherited Members

### 7.13.1 Detailed Description

Definition at line 906 of file dyngenpar.h.

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 ListTokenSource()

```
DynGenPar::ListTokenSource::ListTokenSource ( )  [inline]
```

Definition at line 908 of file dyngenpar.h.

#### 7.13.2.2 ~ListTokenSource()

```
virtual DynGenPar::ListTokenSource::~ListTokenSource ( )  [inline], [virtual]
```

Definition at line 909 of file dyngenpar.h.

### 7.13.3 Member Function Documentation

#### 7.13.3.1 readToken()

```
virtual Cat DynGenPar::ListTokenSource::readToken ( )  [inline], [protected], [virtual]
```

just fetch the next token from the list

Implements DynGenPar::TokenSource.

Definition at line 917 of file dyngenpar.h.

#### 7.13.3.2 rewindTo()

```
virtual bool DynGenPar::ListTokenSource::rewindTo (
            int pos,
            const LexerState &  = LexerState() )  [inline], [virtual]
```

overridden because lists can be rewound

Reimplemented from DynGenPar::TokenSource.

Definition at line 912 of file dyngenpar.h.

### 7.13.4 Member Data Documentation

#### 7.13.4.1 inputTokens

```
QList<Cat> DynGenPar::ListTokenSource::inputTokens
```

Definition at line 910 of file dyngenpar.h.

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

- dyngenpar.h

## 7.14 DynGenPar::Match Struct Reference

(possibly partial) match

### Public Member Functions

- Match ()
  *dummy default constructor for bindings*
- Match (int l, Node t, int n, PseudoCatScope s)
- Match (int l, Node t, int n, PseudoCatScope s, const NextTokenConstraints &nt)

### Public Attributes

- int len
- Node tree
- int ruleno
  *used for PMCFGs*
- PseudoCatScope scope
- NextTokenConstraints nextTokenConstraints

### 7.14.1 Detailed Description

(possibly partial) match

Definition at line 397 of file dyngenpar.h.

### 7.14.2 Constructor & Destructor Documentation

#### 7.14.2.1 Match() [1/3]

```
DynGenPar::Match::Match ( )  [inline]
```

dummy default constructor for bindings

Definition at line 399 of file dyngenpar.h.

#### 7.14.2.2 Match() [2/3]

```
DynGenPar::Match::Match (
            int l,
            Node t,
            int n,
            PseudoCatScope s )  [inline]
```

Definition at line 400 of file dyngenpar.h.

#### 7.14.2.3 Match() [3/3]

```
DynGenPar::Match::Match (
            int l,
            Node t,
            int n,
            PseudoCatScope s,
            const NextTokenConstraints & nt )  [inline]
```

Definition at line 402 of file dyngenpar.h.

### 7.14.3 Member Data Documentation

#### 7.14.3.1 len

```
int DynGenPar::Match::len
```

Definition at line 404 of file dyngenpar.h.

#### 7.14.3.2 nextTokenConstraints

```
NextTokenConstraints DynGenPar::Match::nextTokenConstraints
```

Definition at line 409 of file dyngenpar.h.

#### 7.14.3.3 ruleno

```
int DynGenPar::Match::ruleno
```

used for PMCFGs

set to 0 where not needed to allow unification

Definition at line 406 of file dyngenpar.h.

#### 7.14.3.4 scope

```
PseudoCatScope DynGenPar::Match::scope
```

Definition at line 408 of file dyngenpar.h.

#### 7.14.3.5 tree

```
Node DynGenPar::Match::tree
```

Definition at line 405 of file dyngenpar.h.

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

- dyngenpar.h

## 7.15 DynGenPar::MultiPrediction Struct Reference

multi-token predictions

### Public Member Functions

- MultiPrediction ()
  *dummy default constructor for bindings*
- MultiPrediction (const QList< Cat > &fullLit, CatArg c)
- bool operator== (const MultiPrediction &other) const
  *needed for QList, QMultiHash*

### Public Attributes

- QList< Cat > fullLiteral
  *the entire literal completed by the prediction*
- Cat cat
  *the nonterminal generating the literal*

### 7.15.1 Detailed Description

multi-token predictions

Definition at line 217 of file dyngenpar.h.

### 7.15.2 Constructor & Destructor Documentation

#### 7.15.2.1 MultiPrediction() [1/2]

```
DynGenPar::MultiPrediction::MultiPrediction ( )  [inline]
```

dummy default constructor for bindings

Definition at line 219 of file dyngenpar.h.

#### 7.15.2.2 MultiPrediction() [2/2]

```
DynGenPar::MultiPrediction::MultiPrediction (
            const QList< Cat > & fullLit,
            CatArg c )  [inline]
```

Definition at line 220 of file dyngenpar.h.

### 7.15.3 Member Function Documentation

#### 7.15.3.1 operator==()

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

needed for QList, QMultiHash

Definition at line 226 of file dyngenpar.h.

### 7.15.4 Member Data Documentation

#### 7.15.4.1 cat

```
Cat DynGenPar::MultiPrediction::cat
```

the nonterminal generating the literal

Definition at line 223 of file dyngenpar.h.

#### 7.15.4.2 fullLiteral

```
QList<Cat> DynGenPar::MultiPrediction::fullLiteral
```

the entire literal completed by the prediction

Definition at line 222 of file dyngenpar.h.

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

- dyngenpar.h

## 7.16 DynGenPar::NextTokenConstraints Struct Reference

rule constraints affecting the next token, for scannerless parsing

### Public Member Functions

- bool operator== (const NextTokenConstraints &other) const
  *needed for hash tables*
- QDataStream & writeExternal (QDataStream &stream) const
  *implementation of the QDataStream operator<<*
- QDataStream & readExternal (QDataStream &stream)
  *implementation of the QDataStream operator>>*

### Public Attributes

- QList< Cat > expect
  *list of context-free categories the next token MUST match*
- QList< Cat > taboo
  *list of context-free categories the next token MUST NOT match*

### 7.16.1 Detailed Description

rule constraints affecting the next token, for scannerless parsing

Definition at line 87 of file dyngenpar.h.

### 7.16.2 Member Function Documentation

#### 7.16.2.1 operator==()

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

needed for hash tables

Definition at line 105 of file dyngenpar.h.

#### 7.16.2.2 readExternal()

```
QDataStream& DynGenPar::NextTokenConstraints::readExternal (
            QDataStream & stream )  [inline]
```

implementation of the QDataStream operator>>

Definition at line 113 of file dyngenpar.h.

#### 7.16.2.3 writeExternal()

```
QDataStream& DynGenPar::NextTokenConstraints::writeExternal (
            QDataStream & stream ) const  [inline]
```

implementation of the QDataStream operator<<

Definition at line 109 of file dyngenpar.h.

### 7.16.3 Member Data Documentation

#### 7.16.3.1 expect

```
QList<Cat> DynGenPar::NextTokenConstraints::expect
```

list of context-free categories the next token MUST match

The categories in this list may be nonterminals or tokens. But they MUST be context-free. In other words, they must not be PMCFG pseudo-categories, and none of the rules used to derive them may contain any PMCFG pseudo-categories or next token constraints. (In particular, it is not possible to nest next token constraints.)

Definition at line 95 of file dyngenpar.h.

#### 7.16.3.2 taboo

`QList<Cat> DynGenPar::NextTokenConstraints::taboo`

list of context-free categories the next token MUST NOT match

The categories in this list may be nonterminals or tokens. But they MUST be context-free. In other words, they must not be PMCFG pseudo-categories, and none of the rules used to derive them may contain any PMCFG pseudo-categories or next token constraints. (In particular, it is not possible to nest next token constraints.)

Definition at line 103 of file dyngenpar.h.

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

- dyngenpar.h

## 7.17 DynGenPar::Node Struct Reference

node in the parse tree

### Public Member Functions

- Node ()
  *error node*
- Node (CatArg c)
- Node (CatArg c, const QVariant &d)
- bool operator== (const Node &other) const
  *needed for QList*

### Public Attributes

- Cat cat
- QVariant data
- QList< Alternative > children

### 7.17.1 Detailed Description

node in the parse tree

Definition at line 320 of file dyngenpar.h.

### 7.17.2 Constructor & Destructor Documentation

#### 7.17.2.1 Node() [1/3]

```
DynGenPar::Node::Node ( )  [inline]
```

error node

Definition at line 321 of file dyngenpar.h.

#### 7.17.2.2 Node() [2/3]

```
DynGenPar::Node::Node (
            CatArg c )  [inline]
```

Definition at line 322 of file dyngenpar.h.

#### 7.17.2.3 Node() [3/3]

```
DynGenPar::Node::Node (
            CatArg c,
            const QVariant & d )  [inline]
```

Definition at line 323 of file dyngenpar.h.

### 7.17.3 Member Function Documentation

#### 7.17.3.1 operator==()

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

needed for QList

Definition at line 330 of file dyngenpar.h.

### 7.17.4 Member Data Documentation

#### 7.17.4.1 cat

```
Cat DynGenPar::Node::cat
```

Definition at line 326 of file dyngenpar.h.

#### 7.17.4.2 children

```
QList<Alternative> DynGenPar::Node::children
```

Definition at line 328 of file dyngenpar.h.

#### 7.17.4.3 data

```
QVariant DynGenPar::Node::data
```

Definition at line 327 of file dyngenpar.h.

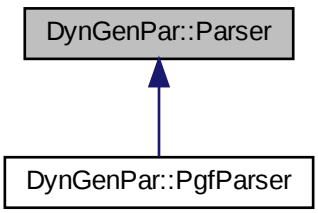
The documentation for this struct was generated from the following file:

- dyngenpar.h

## 7.18 DynGenPar::Parser Class Reference

main class

Inheritance diagram for DynGenPar::Parser:



## Public Member Functions

- Parser (TokenSource *tokenSource)
- virtual ~Parser ()
- bool isToken (CatArg cat) const
- void addToken (CatArg cat)
- bool isLiteral (const QList< Cat > &list) const

  *is a given list of categories a literal?*
- void initCaches ()

  *clears all caches, then computes the nullable categories and the initial graph*
- void addRule (CatArg cat, const Rule &rule)

  *adds a new rule to the grammar, updates the nullable categories and the initial graph and clears the other caches*
- void loadCfg (const Cfg &cfg)
- Cfg getCfg ()

  *get a Cfg object back from the parser, for serialization*
- bool loadPmcfg (const Pmcfg &pmcfg)

  *loads a PMCFG in standard form, converting it to the internal representation*
- bool addPmcfgRule (Pmcfg &pmcfg, CatArg cat, const Rule &rule)

  *adds a new rule to the grammar (both the PMCFG and the internal representation), updates the nullable categories and the initial graph and clears the other caches*
- QList< Match > parse (int *errorPos=0, Cat *errorToken=0, int *incrementalPos=0, QList< StackItem > *incrementalStacks=0, QList< Match > *incrementalMatches=0, LexerState *lexerState=0)

  *parse the input string*
- QList< Match > parse (ParseState *parseState)

  *overloaded version using ParseState, for bindings*
- void saveState (ParseState *parseState)

  *saves the current parser state, only meaningful during a parse operation*
- Predictions computePredictions (const QList< StackItem > &stacks) const

  *compute a set of predictions from the stacks produced by an incremental parse*
- Predictions computePredictions (const ParseState &parseState) const

  *overloaded version using ParseState, for bindings*
- QHash< Cat, QSet< Cat > > expandNonterminalPrediction (CatArg cat) const

  *expand a nonterminal prediction to the possible initial tokens and the nonterminals they immediately reduce to (for categorization), using recursive descent*
- QHash< Cat, QSet< Cat > > expandNonterminalPredictionC (CatArg cat)

  *expand a nonterminal prediction to the possible initial tokens and the nonterminals they immediately reduce to (for categorization), using recursive descent*
- MultiPredictions computeMultiPredictions (const QList< StackItem > &stacks) const

  *compute a set of multi-token predictions from the stacks produced by an incremental parse*
- MultiPredictions computeMultiPredictions (const ParseState &parseState) const

  *overloaded version using ParseState, for bindings*
- QHash< Cat, QSet< QList< Cat > > > expandNonterminalPredictionMulti (CatArg cat) const

  *expand a nonterminal prediction to the possible initial nonempty literals (strings of one or more tokens) and the nonterminals they immediately reduce to (for categorization), using recursive descent*
- QHash< Cat, QSet< QList< Cat > > > expandNonterminalPredictionMultiC (CatArg cat)

  *expand a nonterminal prediction to the possible initial nonempty literals (strings of one or more tokens) and the nonterminals they immediately reduce to (for categorization), using recursive descent*
- ConstrainedPredictions computeConstrainedPredictions (const QList< StackItem > &stacks) const

  *compute a set of predictions from the stacks produced by an incremental parse*
- ConstrainedPredictions computeConstrainedPredictions (const ParseState &parseState) const

  *overloaded version using ParseState, for bindings*
- QHash< Cat, QSet< Cat > > expandNonterminalPredictionC (CatArg cat, const NextTokenConstraints &nextTokenConstraints)

*expand a nonterminal prediction to the possible initial tokens and the nonterminals they immediately reduce to (for categorization), using recursive descent*

- QHash< Cat, QSet< Cat > > expandNonterminalPredictionC (CatArg cat, const QList< NextTokenConstraints > &nextTokenConstraintsList)

  *expand a nonterminal prediction to the possible initial tokens and the nonterminals they immediately reduce to (for categorization), using recursive descent*
- ConstrainedMultiPredictions computeConstrainedMultiPredictions (const QList< StackItem > &stacks) const

  *compute a set of multi-token predictions from the stacks produced by an incremental parse*
- ConstrainedMultiPredictions computeConstrainedMultiPredictions (const ParseState &parseState) const

  *overloaded version using ParseState, for bindings*
- QHash< Cat, QSet< QList< Cat > > > expandNonterminalPredictionMultiC (CatArg cat, const NextTokenConstraints &nextTokenConstraints)

  *expand a nonterminal prediction to the possible initial nonempty literals (strings of one or more tokens) and the nonterminals they immediately reduce to (for categorization), using recursive descent*
- QHash< Cat, QSet< QList< Cat > > > expandNonterminalPredictionMultiC (CatArg cat, const QList< NextTokenConstraints > &nextTokenConstraintsList)

  *expand a nonterminal prediction to the possible initial nonempty literals (strings of one or more tokens) and the nonterminals they immediately reduce to (for categorization), using recursive descent*

## Public Attributes

### grammar

- RuleSet rules

  *grammar rules*
- TokenSet tokens

  *tokens*
- Cat startCat

  *start category*

### additional information needed for PMCFGs

- QHash< Cat, QPair< Cat, QList< Cat > > > pseudoCats

  *pseudo-categories, used to represent PMCFGs internally*
- QHash< Cat, QPair< Cat, int > > componentCats

  *maps categories which represent components of a multi-component category to the category and component index they represent*
- QHash< Cat, QList< Cat > > catComponents

  *maps multi-component categories to the list of their components*

## Protected Attributes

- TokenSource * inputSource

  *input source*

### 7.18.1 Detailed Description

main class

Definition at line 1158 of file dyngenpar.h.

### 7.18.2 Constructor & Destructor Documentation

#### 7.18.2.1 Parser()

```
DynGenPar::Parser::Parser (
            TokenSource * tokenSource )  [inline]
```

Definition at line 1161 of file dyngenpar.h.

#### 7.18.2.2 ~Parser()

```
virtual DynGenPar::Parser::~Parser ( )  [inline], [virtual]
```

Definition at line 1163 of file dyngenpar.h.

### 7.18.3 Member Function Documentation

#### 7.18.3.1 addPmcfgRule()

```
bool DynGenPar::Parser::addPmcfgRule (
            Pmcfg & pmcfg,
            CatArg cat,
            const Rule & rule )
```

adds a new rule to the grammar (both the PMCFG and the internal representation), updates the nullable categories and the initial graph and clears the other caches

**Returns**

`true` on success, `false` on failure

**Note**

Functions can be added by simply calling Pmcfg::addFunction on the *pmcfg* object (for named functions) or appending to the *pmcfg* object's Pmcfg::functions member (for unnamed functions).

Definition at line 974 of file dyngenpar.cpp.

#### 7.18.3.2 addRule()

```
void DynGenPar::Parser::addRule (
            CatArg cat,
            const Rule & rule )
```

adds a new rule to the grammar, updates the nullable categories and the initial graph and clears the other caches

Definition at line 689 of file dyngenpar.cpp.

#### 7.18.3.3 addToken()

```
void DynGenPar::Parser::addToken (
            CatArg cat )  [inline]
```

Definition at line 1165 of file dyngenpar.h.

#### 7.18.3.4 computeConstrainedMultiPredictions() [1/2]

```
ConstrainedMultiPredictions DynGenPar::Parser::computeConstrainedMultiPredictions (
            const QList< StackItem > & stacks ) const
```

compute a set of multi-token predictions from the stacks produced by an incremental parse

**Returns**

a table of extended categories which are valid continuations for the current input

An extended category is either a nonterminal or a "literal", meaning a nonempty list of tokens appearing in sequence in a rule.

The table is represented as a (possibly multi-valued) hash table mapping a list of categories (containing either exactly one nonterminal or at least one terminal) to

1. another list of categories representing the completed literal in case of a literal (e.g. if the complete literal is "abc" and the user already entered "a", the entry will have key "bc" and value "abc"), and reproducing the key otherwise,
2. the nonterminal the literal appears in (or the nonterminal itself if the predicted category is a nonterminal) and
3. for nonterminals, associated next token constraints.

For terminal/literal predictions, the constraints are immediately validated. For nonterminal predictions, this must be done during expansion.

Definition at line 3299 of file dyngenpar.cpp.

#### 7.18.3.5 computeConstrainedMultiPredictions() [2/2]

```
ConstrainedMultiPredictions DynGenPar::Parser::computeConstrainedMultiPredictions (
            const ParseState & parseState ) const  [inline]
```

overloaded version using ParseState, for bindings

Definition at line 1240 of file dyngenpar.h.

#### 7.18.3.6 computeConstrainedPredictions() [1/2]

```
ConstrainedPredictions DynGenPar::Parser::computeConstrainedPredictions (
            const QList< StackItem > & stacks ) const
```

compute a set of predictions from the stacks produced by an incremental parse

**Returns**

a table of (terminal or nonterminal) categories which are valid continuations for the current input and associated next token constraints

For terminal predictions, the constraints are immediately validated. For nonterminal predictions, this must be done during expansion.

**Warning**

This prediction method does not support multi-token literals.

Definition at line 3147 of file dyngenpar.cpp.

#### 7.18.3.7 computeConstrainedPredictions() [2/2]

```
ConstrainedPredictions DynGenPar::Parser::computeConstrainedPredictions (
            const ParseState & parseState ) const  [inline]
```

overloaded version using ParseState, for bindings

Definition at line 1229 of file dyngenpar.h.

#### 7.18.3.8 computeMultiPredictions() [1/2]

```
MultiPredictions DynGenPar::Parser::computeMultiPredictions (
            const QList< StackItem > & stacks ) const
```

compute a set of multi-token predictions from the stacks produced by an incremental parse

**Returns**

a table of extended categories which are valid continuations for the current input.

An extended category is either a nonterminal or a "literal", meaning a nonempty list of tokens appearing in sequence in a rule.

The table is represented as a (possibly multi-valued) hash table mapping a list of categories (containing either exactly one nonterminal or at least one terminal) to

1. another list of categories representing the completed literal in case of a literal (e.g. if the complete literal is "abc" and the user already entered "a", the entry will have key "bc" and value "abc"), and reproducing the key otherwise, and
2. the nonterminal the literal appears in (or the nonterminal itself if the predicted category is a nonterminal).

**Warning**

This prediction method does not support next token constraints.

Definition at line 2806 of file dyngenpar.cpp.

#### 7.18.3.9 computeMultiPredictions() [2/2]

```
MultiPredictions DynGenPar::Parser::computeMultiPredictions (
            const ParseState & parseState ) const  [inline]
```

overloaded version using ParseState, for bindings

Definition at line 1218 of file dyngenpar.h.

#### 7.18.3.10 computePredictions() [1/2]

```
Predictions DynGenPar::Parser::computePredictions (
            const QList< StackItem > & stacks ) const
```

compute a set of predictions from the stacks produced by an incremental parse

**Returns**

a set of (terminal or nonterminal) categories which are valid continuations for the current input

**Warning**

This prediction method does not support multi-token literals nor next token constraints.

Definition at line 2515 of file dyngenpar.cpp.

#### 7.18.3.11 computePredictions() [2/2]

```
Predictions DynGenPar::Parser::computePredictions (
            const ParseState & parseState ) const  [inline]
```

overloaded version using ParseState, for bindings

Definition at line 1210 of file dyngenpar.h.

#### 7.18.3.12 expandNonterminalPrediction()

```
QHash< Cat, QSet< Cat > > DynGenPar::Parser::expandNonterminalPrediction (
            CatArg cat ) const
```

expand a nonterminal prediction to the possible initial tokens and the nonterminals they immediately reduce to (for categorization), using recursive descent

only follow the leftmost branch and ignore left recursion because it does not affect the starting tokens

also expand each category only once because we do not need the full parse trees, only the last category

**Warning**

This expansion method does not honor context-sensitive constraints (PMCFG constraints, next token constraints) attached to the skipped epsilon matches.

Definition at line 2575 of file dyngenpar.cpp.

#### 7.18.3.13 expandNonterminalPredictionC() [1/3]

```
QHash< Cat, QSet< Cat > > DynGenPar::Parser::expandNonterminalPredictionC (
            CatArg cat )
```

expand a nonterminal prediction to the possible initial tokens and the nonterminals they immediately reduce to (for categorization), using recursive descent

only follow the leftmost branch and ignore left recursion because it does not affect the starting tokens

also match all the nullable categories encountered to epsilon, and collect and enforce any context-sensitive constraints

Definition at line 2774 of file dyngenpar.cpp.

#### 7.18.3.14 expandNonterminalPredictionC() [2/3]

```
QHash< Cat, QSet< Cat > > DynGenPar::Parser::expandNonterminalPredictionC (
            CatArg cat,
            const NextTokenConstraints & nextTokenConstraints )
```

expand a nonterminal prediction to the possible initial tokens and the nonterminals they immediately reduce to (for categorization), using recursive descent

only follow the leftmost branch and ignore left recursion because it does not affect the starting tokens

also match all the nullable categories encountered to epsilon, and collect and enforce any context-sensitive constraints

This overload also enforces the next token constraints passed as a second argument.

Definition at line 3215 of file dyngenpar.cpp.

#### 7.18.3.15 expandNonterminalPredictionC() [3/3]

```
QHash< Cat, QSet< Cat > > DynGenPar::Parser::expandNonterminalPredictionC (
            CatArg cat,
            const QList< NextTokenConstraints > & nextTokenConstraintsList )
```

expand a nonterminal prediction to the possible initial tokens and the nonterminals they immediately reduce to (for categorization), using recursive descent

only follow the leftmost branch and ignore left recursion because it does not affect the starting tokens

also match all the nullable categories encountered to epsilon, and collect and enforce any context-sensitive constraints

This overload also enforces the disjunctive (inclusive OR) list of next token constraints passed as a second argument, i.e. if any of the next token constraint sets in *nextTokenConstraintsList* matches, the prediction is accepted.

Definition at line 3242 of file dyngenpar.cpp.

#### 7.18.3.16 expandNonterminalPredictionMulti()

```
QHash< Cat, QSet< QList< Cat > > > DynGenPar::Parser::expandNonterminalPredictionMulti (
            CatArg cat ) const
```

expand a nonterminal prediction to the possible initial nonempty literals (strings of one or more tokens) and the nonterminals they immediately reduce to (for categorization), using recursive descent

only follow the leftmost branch and ignore left recursion because it does not affect the starting tokens

also expand each category only once because we do not need the full parse trees, only the last category

**Warning**

This expansion method does not honor context-sensitive constraints (PMCFG constraints, next token constraints) attached to the skipped epsilon matches.

Definition at line 2918 of file dyngenpar.cpp.

#### 7.18.3.17 expandNonterminalPredictionMultiC() [1/3]

```
QHash< Cat, QSet< QList< Cat > > > DynGenPar::Parser::expandNonterminalPredictionMultiC (
            CatArg cat )
```

expand a nonterminal prediction to the possible initial nonempty literals (strings of one or more tokens) and the nonterminals they immediately reduce to (for categorization), using recursive descent

only follow the leftmost branch and ignore left recursion because it does not affect the starting tokens

also match all the nullable categories encountered to epsilon, and collect and enforce any context-sensitive constraints

Definition at line 3124 of file dyngenpar.cpp.

#### 7.18.3.18 expandNonterminalPredictionMultiC() [2/3]

```
QHash< Cat, QSet< QList< Cat > > > DynGenPar::Parser::expandNonterminalPredictionMultiC (
            CatArg cat,
            const NextTokenConstraints & nextTokenConstraints )
```

expand a nonterminal prediction to the possible initial nonempty literals (strings of one or more tokens) and the nonterminals they immediately reduce to (for categorization), using recursive descent

only follow the leftmost branch and ignore left recursion because it does not affect the starting tokens

also match all the nullable categories encountered to epsilon, and collect and enforce any context-sensitive constraints

This overload also enforces the next token constraints passed as a second argument.

Definition at line 3397 of file dyngenpar.cpp.

#### 7.18.3.19 expandNonterminalPredictionMultiC() [3/3]

```
QHash< Cat, QSet< QList< Cat > > > DynGenPar::Parser::expandNonterminalPredictionMultiC (
            CatArg cat,
            const QList< NextTokenConstraints > & nextTokenConstraintsList )
```

expand a nonterminal prediction to the possible initial nonempty literals (strings of one or more tokens) and the nonterminals they immediately reduce to (for categorization), using recursive descent

only follow the leftmost branch and ignore left recursion because it does not affect the starting tokens

also match all the nullable categories encountered to epsilon, and collect and enforce any context-sensitive constraints

This overload also enforces the disjunctive (inclusive OR) list of next token constraints passed as a second argument, i.e. if any of the next token constraint sets in *nextTokenConstraintsList* matches, the prediction is accepted.

Definition at line 3425 of file dyngenpar.cpp.

#### 7.18.3.20 getCfg()

```
Cfg DynGenPar::Parser::getCfg ( )  [inline]
```

get a Cfg object back from the parser, for serialization

Definition at line 1176 of file dyngenpar.h.

#### 7.18.3.21 initCaches()

```
void DynGenPar::Parser::initCaches (
            void  )
```

clears all caches, then computes the nullable categories and the initial graph

should be called after each direct grammar modification (addRule takes care of updating the caches, which is more efficient than clearing.)

Definition at line 578 of file dyngenpar.cpp.

#### 7.18.3.22 isLiteral()

```
bool DynGenPar::Parser::isLiteral (
            const QList< Cat > & list ) const
```

is a given list of categories a literal?

**Returns**

`true` if the given *list* of categories is a literal, i.e. contains only tokens, `false` otherwise

Definition at line 553 of file dyngenpar.cpp.

#### 7.18.3.23 isToken()

```
bool DynGenPar::Parser::isToken (
            CatArg cat ) const  [inline]
```

Definition at line 1164 of file dyngenpar.h.

#### 7.18.3.24 loadCfg()

```
void DynGenPar::Parser::loadCfg (
            const Cfg & cfg )  [inline]
```

Definition at line 1169 of file dyngenpar.h.

#### 7.18.3.25 loadPmcfg()

```
bool DynGenPar::Parser::loadPmcfg (
            const Pmcfg & pmcfg )
```

loads a PMCFG in standard form, converting it to the internal representation

Rules containing categories which are neither tokens nor have rules are discarded, as they're unreachable and as we cannot transform them without knowing the dimension of the unused categories.

**Returns**

`true` on success, `false` on failure

**Warning**

The parser rules may be in an inconsistent state if the loading failed.

Definition at line 932 of file dyngenpar.cpp.

#### 7.18.3.26 parse() [1/2]

```
QList< Match > DynGenPar::Parser::parse (
            int * errorPos = 0,
            Cat * errorToken = 0,
            int * incrementalPos = 0,
            QList< StackItem > * incrementalStacks = 0,
            QList< Match > * incrementalMatches = 0,
            LexerState * lexerState = 0 )
```

parse the input string

**Returns**

the list of matches

**Parameters**

| | | |
|---|---|---|
| out | *errorPos* | if non-NULL, is filled with −1 on success and with the number of accepted tokens before the error occurred on error (Caution: This might not be a good position indicator to show to the end user. For some token sources, lexerState contains a more user-centric position indicator which can be obtained through that token source's API.) |
| out | *errorToken* | if non-NULL, is filled with 0 (epsilon) on success and with the token triggering the error on error. |
| in,out | *incrementalPos* | should be NULL for a non-incremental parse, a pointer to a negative integer to start an incremental parse or a pointer to a nonnegative integer to continue an incremental parse. It will be set to the current end of input if non-NULL. |
| in,out | *incrementalStacks* | should be NULL for a non-incremental, non-predictive parse or a pointer to QList<StackItem> (initially empty) for an incremental parse or when needed for prediction. It represents the internal parser states. Normally, this will be the list of stacks at the end of the parsing process. However, if an error occurred, that list would always be empty, so instead, we return the list of stacks before the token triggering the error, thus allowing to use the prediction functionality to report what token would have been expected instead of the faulty one. (An empty list of stacks means that the input was expected to end before the faulty token.) |
| in,out | *incrementalMatches* | should be NULL for a non-incremental parse. For an incremental parse, it can be NULL if you do not need to get your matches back a second time when there is no new input. If set to non-NULL, it will be returned as is if there is no new input in an incremental parse, or updated to the current return value otherwise. |
| in,out | *lexerState* | if non-NULL, is filled with the lexer state at the end of the (incremental) parsing process. (In case of an error, it is filled with the lexer state where the error occurred, i.e. before shifting the faulty token, to allow reporting error positions accurately.) It is useful to allow rewinding to a previous position with a stateful lexer. It can be NULL for a non-incremental or a sequential incremental parse (i.e. if rewinding is not needed) or if a stateless token source (stateless lexer, token buffer etc.) is used. When starting a new incremental parse, the lexer state pointed to should be a null (default-constructed) LexerState. If the *lexerState* pointer is NULL, all rewind operations for the lexer will be passed a null (default-constructed) LexerState; stateful lexers will then fail any rewind operations. |

**Note**

An "error" is defined as a place at which it is no longer possible to continue parsing. This does not include incomplete input, i.e. input which can be continued to valid input, but does not form valid input by itself. If an empty list of matches is returned without an error being flagged, this means that the input was incomplete. Use one of the prediction functions (e.g. computePredictions) to list possible continuations ("expected ..."). Incremental parsing can be used to process additional input given by the user.

Definition at line 2437 of file dyngenpar.cpp.

#### 7.18.3.27 parse() [2/2]

```
QList<Match> DynGenPar::Parser::parse (
            ParseState * parseState )  [inline]
```

overloaded version using ParseState, for bindings

Definition at line 1185 of file dyngenpar.h.

#### 7.18.3.28 saveState()

```
void DynGenPar::Parser::saveState (
            ParseState * parseState )  [inline]
```

saves the current parser state, only meaningful during a parse operation

This method is intended to be used in callbacks executed within parse. When called with no running parse operation, *parseState* will be reset.

Definition at line 1196 of file dyngenpar.h.

### 7.18.4 Member Data Documentation

#### 7.18.4.1 catComponents

```
QHash<Cat, QList<Cat> > DynGenPar::Parser::catComponents
```

maps multi-component categories to the list of their components

used during the import of PMCFG rules in the standard representation

can be left out if the internal representation is used

Definition at line 1300 of file dyngenpar.h.

#### 7.18.4.2 componentCats

```
QHash<Cat, QPair<Cat, int> > DynGenPar::Parser::componentCats
```

maps categories which represent components of a multi-component category to the category and component index they represent

also used to look up whether a category is a component of a multi-component category

Definition at line 1294 of file dyngenpar.h.

#### 7.18.4.3 inputSource

```
TokenSource* DynGenPar::Parser::inputSource  [protected]
```

input source

Definition at line 1305 of file dyngenpar.h.

#### 7.18.4.4 pseudoCats

```
QHash<Cat, QPair<Cat, QList<Cat> > > DynGenPar::Parser::pseudoCats
```

pseudo-categories, used to represent PMCFGs internally

maps a pseudo-category to:

1. the actual component of the multidimensional category this pseudo-category stands for - just substituting this for the pseudo-category results in the context-free approximation of the PMCFG
2. the list of pseudo-categories resulting from the use of the SAME argument (not just the same category) in the same context - all those pseudo-categories, when used in the same context, have to be expanded using matching rules; we use top-down expansion for all except the first encountered one to guarantee the same expansion as the one obtained while reducing the first one

Note that, when parsing a PMCFG, the initial graph and the set of nullable categories are the ones for the context-free approximation. The PMCFG constraints are only evaluated during the matching resp. reducing steps.

Also note that tokens are always 1-dimensional, so tokens may not be pseudo-categories nor the actual component for a pseudo-category.

Definition at line 1288 of file dyngenpar.h.

#### 7.18.4.5 rules

```
RuleSet DynGenPar::Parser::rules
```

grammar rules

**Warning**

Modifying the grammar directly requires initCaches, use addRule if possible.

Definition at line 1258 of file dyngenpar.h.

#### 7.18.4.6 startCat

```
Cat DynGenPar::Parser::startCat
```

start category

Definition at line 1262 of file dyngenpar.h.

#### 7.18.4.7 tokens

```
TokenSet DynGenPar::Parser::tokens
```

tokens

Definition at line 1260 of file dyngenpar.h.

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

- dyngenpar.h
- dyngenpar.cpp

## 7.19 DynGenPar::ParseState Struct Reference

parse state struct, for bindings

### Public Member Functions

- ParseState ()
- ParseState (const ParseState &other)
- void reset ()

### Public Attributes

- int errorPos
- Cat errorToken
- int incrementalPos
- QList< StackItem > incrementalStacks
- QList< Match > incrementalMatches
- LexerState lexerState

### 7.19.1 Detailed Description

parse state struct, for bindings

Definition at line 1131 of file dyngenpar.h.

### 7.19.2 Constructor & Destructor Documentation

#### 7.19.2.1 ParseState() [1/2]

```
DynGenPar::ParseState::ParseState ( )  [inline]
```

Definition at line 1132 of file dyngenpar.h.

#### 7.19.2.2 ParseState() [2/2]

```
DynGenPar::ParseState::ParseState (
            const ParseState & other )  [inline]
```

Definition at line 1136 of file dyngenpar.h.

### 7.19.3 Member Function Documentation

#### 7.19.3.1 reset()

```
void DynGenPar::ParseState::reset ( )  [inline]
```

Definition at line 1150 of file dyngenpar.h.

### 7.19.4 Member Data Documentation

#### 7.19.4.1 errorPos

```
int DynGenPar::ParseState::errorPos
```

Definition at line 1143 of file dyngenpar.h.

#### 7.19.4.2 errorToken

```
Cat DynGenPar::ParseState::errorToken
```

Definition at line 1144 of file dyngenpar.h.

#### 7.19.4.3 incrementalMatches

```
QList<Match> DynGenPar::ParseState::incrementalMatches
```

Definition at line 1147 of file dyngenpar.h.

#### 7.19.4.4 incrementalPos

```
int DynGenPar::ParseState::incrementalPos
```

Definition at line 1145 of file dyngenpar.h.

#### 7.19.4.5 incrementalStacks

```
QList<StackItem> DynGenPar::ParseState::incrementalStacks
```

Definition at line 1146 of file dyngenpar.h.

#### 7.19.4.6 lexerState

```
LexerState DynGenPar::ParseState::lexerState
```

Definition at line 1148 of file dyngenpar.h.

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

- dyngenpar.h

## 7.20 DynGenPar::Pgf Struct Reference

representation of the information in .pgf files in a format we can process

### Public Member Functions

- Pgf ()
  *dummy default constructor for bindings*
- Pgf (const QString &fileName, const QString &concreteName=QString())
  *constructor, loads concrete syntax from .pgf file*

### Public Attributes

- Pmcfg pmcfg
  *the PMCFG (in (almost) standard form)*
- QStringList catNames
  *names of categories, in general not unique*
- QStringList functionNames
  *names of functions, in general not unique*
- QHash< QString, int > tokenHash
  *hash table for quick token lexing*
- QList< QPair< QString, int > > suffixes
  *list of &+ suffixes with their IDs*
- QHash< QString, QStringList > componentNames
  *names of category components*
- int firstFunction
  *the function ID of the first non-coercion function*

## 7.20.1 Detailed Description

representation of the information in .pgf files in a format we can process

Definition at line 52 of file pgf.h.

## 7.20.2 Constructor & Destructor Documentation

### 7.20.2.1 Pgf() [1/2]

```
DynGenPar::Pgf::Pgf ( )  [inline]
```

dummy default constructor for bindings

Definition at line 54 of file pgf.h.

### 7.20.2.2 Pgf() [2/2]

```
DynGenPar::Pgf::Pgf (
            const QString & fileName,
            const QString & concreteName = QString() )
```

constructor, loads concrete syntax from .pgf file

Loads a .pgf file from disk and imports it into a format we can process.

Only one concrete grammar is loaded. If the .pgf file contains more than one concrete grammar, the name of the concrete grammar to load must be specified (otherwise a fatal error is raised).

Definition at line 602 of file pgf.cpp.

### 7.20.3 Member Data Documentation

#### 7.20.3.1 catNames

```
QStringList DynGenPar::Pgf::catNames
```

names of categories, in general not unique

Definition at line 59 of file pgf.h.

#### 7.20.3.2 componentNames

```
QHash<QString, QStringList> DynGenPar::Pgf::componentNames
```

names of category components

Definition at line 64 of file pgf.h.

#### 7.20.3.3 firstFunction

```
int DynGenPar::Pgf::firstFunction
```

the function ID of the first non-coercion function

Definition at line 65 of file pgf.h.

#### 7.20.3.4 functionNames

```
QStringList DynGenPar::Pgf::functionNames
```

names of functions, in general not unique

Definition at line 60 of file pgf.h.

#### 7.20.3.5 pmcfg

```
Pmcfg DynGenPar::Pgf::pmcfg
```

the PMCFG (in (almost) standard form)

Definition at line 58 of file pgf.h.

#### 7.20.3.6 suffixes

```
QList<QPair<QString, int> > DynGenPar::Pgf::suffixes
```

list of &+ suffixes with their IDs

Definition at line 62 of file pgf.h.

#### 7.20.3.7 tokenHash

```
QHash<QString, int> DynGenPar::Pgf::tokenHash
```

hash table for quick token lexing

Definition at line 61 of file pgf.h.

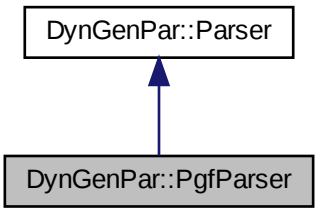
The documentation for this struct was generated from the following files:

- pgf.h
- pgf.cpp

## 7.21 DynGenPar::PgfParser Class Reference

parser for PGF grammars

Inheritance diagram for DynGenPar::PgfParser:




### Public Member Functions

- PgfParser (const Pgf &p)
- PgfParser (const QString &fileName, const QString &concreteName=QString())
- virtual ~PgfParser ()
- void setInputStdin ()
- void setInputFile (const QString &fileName)
- void setInputBytes (const QByteArray &bytes)
- void setInputString (const QString &string)
- void setInputBuffer (QByteArray *buffer)
- QString catName (int cat) const
- QString functionName (int id) const
- void filterCoercionsFromSyntaxTree (Node &tree) const

### Public Attributes

- Pgf pgf

### Additional Inherited Members

## 7.21.1 Detailed Description

parser for PGF grammars

Definition at line 70 of file pgf.h.

## 7.21.2 Constructor & Destructor Documentation

### 7.21.2.1 PgfParser() [1/2]

```
DynGenPar::PgfParser::PgfParser (
            const Pgf & p )
```

Definition at line 1299 of file pgf.cpp.

### 7.21.2.2 PgfParser() [2/2]

```
DynGenPar::PgfParser::PgfParser (
            const QString & fileName,
            const QString & concreteName = QString() )
```

Definition at line 1304 of file pgf.cpp.

### 7.21.2.3 ~PgfParser()

```
virtual DynGenPar::PgfParser::~PgfParser ( )  [inline], [virtual]
```

Definition at line 74 of file pgf.h.

## 7.21.3 Member Function Documentation

#### 7.21.3.1 catName()

```
QString DynGenPar::PgfParser::catName (
            int cat ) const
```

Definition at line 1335 of file pgf.cpp.

#### 7.21.3.2 filterCoercionsFromSyntaxTree()

```
void DynGenPar::PgfParser::filterCoercionsFromSyntaxTree (
            Node & tree ) const
```

Definition at line 1355 of file pgf.cpp.

#### 7.21.3.3 functionName()

```
QString DynGenPar::PgfParser::functionName (
            int id ) const  [inline]
```

Definition at line 81 of file pgf.h.

#### 7.21.3.4 setInputBuffer()

```
void DynGenPar::PgfParser::setInputBuffer (
            QByteArray * buffer )
```

Definition at line 1330 of file pgf.cpp.

#### 7.21.3.5 setInputBytes()

```
void DynGenPar::PgfParser::setInputBytes (
            const QByteArray & bytes )
```

Definition at line 1320 of file pgf.cpp.

#### 7.21.3.6 setInputFile()

```
void DynGenPar::PgfParser::setInputFile (
            const QString & fileName )
```

Definition at line 1315 of file pgf.cpp.

#### 7.21.3.7 setInputStdin()

```
void DynGenPar::PgfParser::setInputStdin ( )
```

Definition at line 1310 of file pgf.cpp.

#### 7.21.3.8 setInputString()

```
void DynGenPar::PgfParser::setInputString (
            const QString & string )
```

Definition at line 1325 of file pgf.cpp.

### 7.21.4 Member Data Documentation

#### 7.21.4.1 pgf

```
Pgf DynGenPar::PgfParser::pgf
```

Definition at line 83 of file pgf.h.

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

- pgf.h
- pgf.cpp

## 7.22 DynGenPar::Pmcfg Struct Reference

PMCFG.

### Public Attributes

**grammar**

- QList< Function > functions
  *list of PMCFG functions*
- RuleSet rules
  *set of PMCFG rules*
- TokenSet tokens
  *set of true tokens*
- Cat startCat
  *start category*
- RuleSet cfRules
  *optional context-free rules*

### function name tables (optional)

- QHash< int, QString > functionNames
- QHash< QString, int > functionIndices
- void addFunction (const QString &name, const Function &function)
- Function lookupFunction (const QVariant &id) const

## 7.22.1 Detailed Description

PMCFG.

Definition at line 1047 of file dyngenpar.h.

## 7.22.2 Member Function Documentation

### 7.22.2.1 addFunction()

```
void DynGenPar::Pmcfg::addFunction (
            const QString & name,
            const Function & function )  [inline]
```

Definition at line 1095 of file dyngenpar.h.

### 7.22.2.2 lookupFunction()

```
Function DynGenPar::Pmcfg::lookupFunction (
            const QVariant & id ) const  [inline]
```

Definition at line 1101 of file dyngenpar.h.

## 7.22.3 Member Data Documentation

### 7.22.3.1 cfRules

```
RuleSet DynGenPar::Pmcfg::cfRules
```

optional context-free rules

allows specifying rules for context-free categories which can be used as "token" terms in PMCFG functions, e.g. A -> "a" | "an"

**Warning**

> The exact expansion used will be reflected only in the parse tree, not in the PMCFG syntax tree. Do not use this feature if you need to know which exact expression was used.

Definition at line 1088 of file dyngenpar.h.

#### 7.22.3.2 functionIndices

```
QHash<QString, int> DynGenPar::Pmcfg::functionIndices
```

Definition at line 1094 of file dyngenpar.h.

#### 7.22.3.3 functionNames

```
QHash<int, QString> DynGenPar::Pmcfg::functionNames
```

Definition at line 1093 of file dyngenpar.h.

#### 7.22.3.4 functions

```
QList<Function> DynGenPar::Pmcfg::functions
```

list of PMCFG functions

This list does not store function names. They can be optionally used and are stored in functionNames and functionIndices.

Definition at line 1054 of file dyngenpar.h.

#### 7.22.3.5 rules

```
RuleSet DynGenPar::Pmcfg::rules
```

set of PMCFG rules

Rules should be labeled with the index of the function in functions or its name as found in functionNames and functionIndices. The expression of the rule is interpreted as the argument list for the function. For example:

```
Rule(1) << "A" << "B"; // calls function #1 with (A, B) as parameters
Rule("f") << "A" << "B"; // calls f(A, B)
```

In a standard PMCFG, the argument list may contain only PMCFG nonterminals. The syntax tree then only contains the function and the syntax trees for each argument. This implementation also allows tokens and context-free nonterminals as function arguments, in which case the syntax tree will contain the parse tree for the context-free argument. In particular, in the case of a token, the data attached to the token is retained.

Definition at line 1073 of file dyngenpar.h.

#### 7.22.3.6 startCat

```
Cat DynGenPar::Pmcfg::startCat
```

start category

The start category must be 1-dimensional.

Definition at line 1079 of file dyngenpar.h.

#### 7.22.3.7 tokens

```
TokenSet DynGenPar::Pmcfg::tokens
```

set of true tokens

(must NOT contain context-free nonterminals)

Definition at line 1076 of file dyngenpar.h.

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

- dyngenpar.h

## 7.23 DynGenPar::PmcfgComponentInfo Struct Reference

attached to the parse trees as rule labels to allow obtaining syntax trees

### Public Member Functions

- PmcfgComponentInfo ()
- PmcfgComponentInfo (const Rule &rule)

### Public Attributes

- Rule pmcfgRule
- QVector< QVector< int > > argPositions

### 7.23.1 Detailed Description

attached to the parse trees as rule labels to allow obtaining syntax trees

Definition at line 1113 of file dyngenpar.h.

### 7.23.2 Constructor & Destructor Documentation

#### 7.23.2.1 PmcfgComponentInfo() [1/2]

```
DynGenPar::PmcfgComponentInfo::PmcfgComponentInfo ( )  [inline]
```

Definition at line 1114 of file dyngenpar.h.

#### 7.23.2.2 PmcfgComponentInfo() [2/2]

```
DynGenPar::PmcfgComponentInfo::PmcfgComponentInfo (
            const Rule & rule )  [inline]
```

Definition at line 1115 of file dyngenpar.h.

### 7.23.3 Member Data Documentation

#### 7.23.3.1 argPositions

```
QVector<QVector<int> > DynGenPar::PmcfgComponentInfo::argPositions
```

must be the same size as pmcfgRule (even if the last arguments are not used)

Definition at line 1118 of file dyngenpar.h.

#### 7.23.3.2 pmcfgRule

```
Rule DynGenPar::PmcfgComponentInfo::pmcfgRule
```

Definition at line 1117 of file dyngenpar.h.

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

- dyngenpar.h

## 7.24 DynGenPar::PseudoCatScope Class Reference

### Public Member Functions

- PseudoCatScope ()
- QHash< Cat, QPair< QPair< Node, NextTokenConstraints >, int > > & pConstraints ()
- QHash< Cat, QPair< int, PseudoCatScope > > & mcfgConstraints ()
- bool hasPConstraint (CatArg cat) const
- bool hasMcfgConstraint (CatArg cat) const
- QPair< QPair< Node, NextTokenConstraints >, int > pConstraint (CatArg cat) const
- QPair< int, PseudoCatScope > mcfgConstraint (CatArg cat) const
- bool isNull () const
- const PseudoCatScopeData * data () const

  *needed for hash tables*
- bool operator== (const PseudoCatScope &other) const

### 7.24.1 Detailed Description

Definition at line 355 of file dyngenpar.h.

### 7.24.2 Constructor & Destructor Documentation

#### 7.24.2.1 PseudoCatScope()

```
DynGenPar::PseudoCatScope::PseudoCatScope ( )  [inline]
```

Definition at line 357 of file dyngenpar.h.

### 7.24.3 Member Function Documentation

#### 7.24.3.1 data()

```
const PseudoCatScopeData* DynGenPar::PseudoCatScope::data ( ) const  [inline]
```

needed for hash tables

Definition at line 382 of file dyngenpar.h.

#### 7.24.3.2 hasMcfgConstraint()

```
bool DynGenPar::PseudoCatScope::hasMcfgConstraint (
            CatArg cat ) const  [inline]
```

Definition at line 369 of file dyngenpar.h.

#### 7.24.3.3 hasPConstraint()

```
bool DynGenPar::PseudoCatScope::hasPConstraint (
            CatArg cat ) const  [inline]
```

Definition at line 366 of file dyngenpar.h.

#### 7.24.3.4 isNull()

```
bool DynGenPar::PseudoCatScope::isNull ( ) const  [inline]
```

Definition at line 380 of file dyngenpar.h.

#### 7.24.3.5 mcfgConstraint()

```
QPair<int, PseudoCatScope> DynGenPar::PseudoCatScope::mcfgConstraint (
            CatArg cat ) const  [inline]
```

Definition at line 377 of file dyngenpar.h.

#### 7.24.3.6 mcfgConstraints()

```
QHash<Cat, QPair<int, PseudoCatScope> >& DynGenPar::PseudoCatScope::mcfgConstraints ( )  [inline]
```

Definition at line 362 of file dyngenpar.h.

#### 7.24.3.7 operator==()

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

Definition at line 383 of file dyngenpar.h.

#### 7.24.3.8 pConstraint()

```
QPair<QPair<Node, NextTokenConstraints>, int> DynGenPar::PseudoCatScope::pConstraint (
            CatArg cat ) const  [inline]
```

Definition at line 372 of file dyngenpar.h.

#### 7.24.3.9 pConstraints()

```
QHash<Cat, QPair<QPair<Node, NextTokenConstraints>, int> >& DynGenPar::PseudoCatScope::p↩
Constraints ( )  [inline]
```

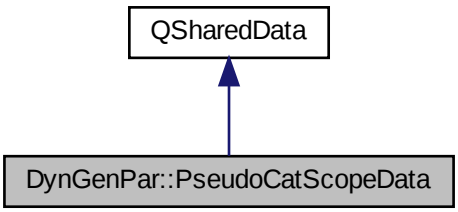
Definition at line 358 of file dyngenpar.h.

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

- dyngenpar.h

## 7.25 DynGenPar::PseudoCatScopeData Class Reference

Inheritance diagram for DynGenPar::PseudoCatScopeData:




### Public Attributes

- QHash< Cat, QPair< QPair< Node, NextTokenConstraints >, int > > pConstraints
  *hash table recording parallel constraints*
- QHash< Cat, QPair< int, PseudoCatScope > > mcfgConstraints
  *hash table recording MCFG constraints*

### 7.25.1 Detailed Description

Definition at line 338 of file dyngenpar.h.

### 7.25.2 Member Data Documentation

#### 7.25.2.1 mcfgConstraints

`QHash<Cat, QPair<int, PseudoCatScope> > DynGenPar::PseudoCatScopeData::mcfgConstraints`

hash table recording MCFG constraints

record the rule number to use for a pseudo-category to match the one used for the first encountered pseudo-category, and the scope to reuse

Definition at line 351 of file dyngenpar.h.

#### 7.25.2.2 pConstraints

`QHash<Cat, QPair<QPair<Node, NextTokenConstraints>, int> > DynGenPar::PseudoCatScopeData::pConstraints`

hash table recording parallel constraints

record the tree for each pseudo-category and, for efficiency, the length of the matched string so we can match the exact same token string if the exact same pseudo-category is used again: this is the "parallel" in PMCFGs; also record the next token constraints

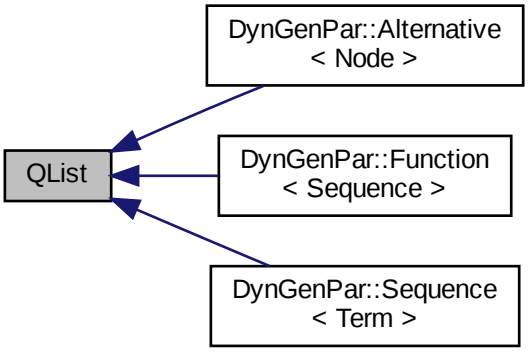
Definition at line 346 of file dyngenpar.h.

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

- dyngenpar.h

## 7.26 QList Class Reference

Inheritance diagram for QList:

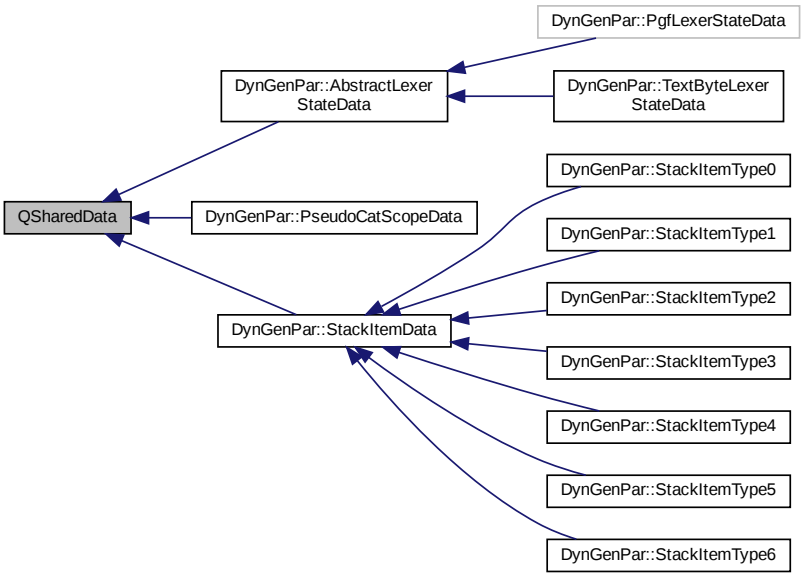



The documentation for this class was generated from the following file:

- dyngenpar.h

## 7.27 QSharedData Class Reference

Inheritance diagram for QSharedData:

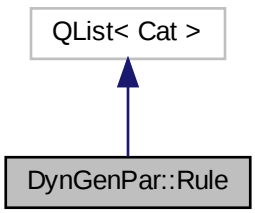



The documentation for this class was generated from the following file:

- dyngenpar.h

## 7.28 DynGenPar::Rule Class Reference

Inheritance diagram for DynGenPar::Rule:



### Public Member Functions

- Rule ()
- Rule (const QVariant &label)
- Rule (const QList< Cat > &list)
- Rule (const QList< Cat > &list, const QVariant &label)
- QVariant label () const
- void setLabel (const QVariant &label)
- Rule & operator+= (const QList< Cat > &other)
- Rule & operator+= (const Cat &value)
- Rule & operator<< (const QList< Cat > &other)
- Rule & operator<< (const Cat &value)
- void add (const Cat &value)

  *Java-style + the binding generator doesn't detect the inherited append.*
- QList< Cat > & toList ()

  *for bindings*
- QDataStream & writeExternal (QDataStream &stream, bool writeLabel=true, bool writeAction=true) const

  *implementation of the QDataStream operator<<*
- QDataStream & readExternal (QDataStream &stream)

  *implementation of the QDataStream operator>>*

### Public Attributes

- NextTokenConstraints nextTokenConstraints
- Action * action

### Static Public Attributes

- static bool serializeLabels = true

  *whether the operator<<(QDataStream &, const Rule &) should serialize labels*
- static bool serializeActions = true

  *whether the operator<<(QDataStream &, const Rule &) should serialize actions*

## 7.28.1 Detailed Description

Definition at line 129 of file dyngenpar.h.

## 7.28.2 Constructor & Destructor Documentation

#### 7.28.2.1 Rule() [1/4]

```
DynGenPar::Rule::Rule ( )  [inline]
```

Definition at line 133 of file dyngenpar.h.

#### 7.28.2.2 Rule() [2/4]

```
DynGenPar::Rule::Rule (
            const QVariant & label )  [inline], [explicit]
```

Definition at line 134 of file dyngenpar.h.

#### 7.28.2.3 Rule() [3/4]

```
DynGenPar::Rule::Rule (
            const QList< Cat > & list )  [inline], [explicit]
```

Definition at line 136 of file dyngenpar.h.

#### 7.28.2.4 Rule() [4/4]

```
DynGenPar::Rule::Rule (
            const QList< Cat > & list,
            const QVariant & label )  [inline]
```

Definition at line 138 of file dyngenpar.h.

### 7.28.3 Member Function Documentation

#### 7.28.3.1 add()

```
void DynGenPar::Rule::add (
            const Cat & value )  [inline]
```

Java-style + the binding generator doesn't detect the inherited append.

Definition at line 161 of file dyngenpar.h.

#### 7.28.3.2 label()

```
QVariant DynGenPar::Rule::label ( ) const  [inline]
```

Definition at line 140 of file dyngenpar.h.

#### 7.28.3.3 operator+=() [1/2]

```
Rule& DynGenPar::Rule::operator+= (
            const QList< Cat > & other )  [inline]
```

Definition at line 144 of file dyngenpar.h.

#### 7.28.3.4 operator+=() [2/2]

```
Rule& DynGenPar::Rule::operator+= (
            const Cat & value )  [inline]
```

Definition at line 148 of file dyngenpar.h.

#### 7.28.3.5 operator<<() [1/2]

```
Rule& DynGenPar::Rule::operator<< (
            const QList< Cat > & other )  [inline]
```

Definition at line 152 of file dyngenpar.h.

#### 7.28.3.6 operator<<() [2/2]

```
Rule& DynGenPar::Rule::operator<< (
            const Cat & value )  [inline]
```

Definition at line 156 of file dyngenpar.h.

#### 7.28.3.7 readExternal()

```
QDataStream& DynGenPar::Rule::readExternal (
            QDataStream & stream )  [inline]
```

implementation of the QDataStream operator>>

Definition at line 176 of file dyngenpar.h.

#### 7.28.3.8 setLabel()

```
void DynGenPar::Rule::setLabel (
            const QVariant & label )  [inline]
```

Definition at line 141 of file dyngenpar.h.

#### 7.28.3.9 toList()

```
QList<Cat>& DynGenPar::Rule::toList ( )  [inline]
```

for bindings

Definition at line 165 of file dyngenpar.h.

#### 7.28.3.10 writeExternal()

```
QDataStream& DynGenPar::Rule::writeExternal (
            QDataStream & stream,
            bool writeLabel = true,
            bool writeAction = true ) const  [inline]
```

implementation of the QDataStream operator<<

Definition at line 167 of file dyngenpar.h.

### 7.28.4 Member Data Documentation

#### 7.28.4.1 action

```
Action* DynGenPar::Rule::action
```

Definition at line 143 of file dyngenpar.h.

#### 7.28.4.2 nextTokenConstraints

```
NextTokenConstraints DynGenPar::Rule::nextTokenConstraints
```

Definition at line 142 of file dyngenpar.h.

#### 7.28.4.3 serializeActions

```
bool DynGenPar::Rule::serializeActions = true  [static]
```

whether the operator<<(QDataStream &, const Rule &) should serialize actions

Definition at line 132 of file dyngenpar.h.

#### 7.28.4.4 serializeLabels

```
bool DynGenPar::Rule::serializeLabels = true  [static]
```

whether the operator<<(QDataStream &, const Rule &) should serialize labels

Definition at line 131 of file dyngenpar.h.

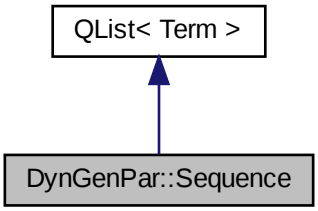
The documentation for this class was generated from the following files:

- dyngenpar.h
- dyngenpar.cpp

## 7.29 DynGenPar::Sequence Class Reference

component of a PMCFG function, a sequence of terms

Inheritance diagram for DynGenPar::Sequence:




### Public Member Functions

- Sequence ()
- Sequence (const NextTokenConstraints &ntc)
- Sequence (const QList< Term > &list)
- Sequence (const QList< Term > &list, const NextTokenConstraints &ntc)
- Sequence & operator+= (const QList< Term > &other)
- Sequence & operator+= (const Term &value)
- Sequence & operator<< (const QList< Term > &other)
- Sequence & operator<< (const Term &value)
- void add (const Term &value)

  *Java-style (for consistency, even though append is detected here)*
- QList< Term > & toList ()

  *for bindings*

### Public Attributes

- NextTokenConstraints nextTokenConstraints

## 7.29.1 Detailed Description

component of a PMCFG function, a sequence of terms

Definition at line 976 of file dyngenpar.h.

## 7.29.2 Constructor & Destructor Documentation

### 7.29.2.1 Sequence() [1/4]

```
DynGenPar::Sequence::Sequence ( )  [inline]
```

Definition at line 980 of file dyngenpar.h.

### 7.29.2.2 Sequence() [2/4]

```
DynGenPar::Sequence::Sequence (
            const NextTokenConstraints & ntc )  [inline], [explicit]
```

Definition at line 981 of file dyngenpar.h.

### 7.29.2.3 Sequence() [3/4]

```
DynGenPar::Sequence::Sequence (
            const QList< Term > & list )  [inline], [explicit]
```

Definition at line 983 of file dyngenpar.h.

### 7.29.2.4 Sequence() [4/4]

```
DynGenPar::Sequence::Sequence (
            const QList< Term > & list,
            const NextTokenConstraints & ntc )  [inline]
```

Definition at line 985 of file dyngenpar.h.

### 7.29.3 Member Function Documentation

#### 7.29.3.1 add()

```
void DynGenPar::Sequence::add (
            const Term & value )  [inline]
```

Java-style (for consistency, even though append is detected here)

Definition at line 1004 of file dyngenpar.h.

#### 7.29.3.2 operator+=() [1/2]

```
Sequence& DynGenPar::Sequence::operator+= (
            const QList< Term > & other )  [inline]
```

Definition at line 987 of file dyngenpar.h.

#### 7.29.3.3 operator+=() [2/2]

```
Sequence& DynGenPar::Sequence::operator+= (
            const Term & value )  [inline]
```

Definition at line 991 of file dyngenpar.h.

#### 7.29.3.4 operator<<() [1/2]

```
Sequence& DynGenPar::Sequence::operator<< (
            const QList< Term > & other )  [inline]
```

Definition at line 995 of file dyngenpar.h.

#### 7.29.3.5 operator<<() [2/2]

```
Sequence& DynGenPar::Sequence::operator<< (
            const Term & value )  [inline]
```

Definition at line 999 of file dyngenpar.h.

#### 7.29.3.6 toList()

```
QList<Term>& DynGenPar::Sequence::toList ( )  [inline]
```

for bindings

Definition at line 1008 of file dyngenpar.h.

### 7.29.4 Member Data Documentation

#### 7.29.4.1 nextTokenConstraints

```
NextTokenConstraints DynGenPar::Sequence::nextTokenConstraints
```

Definition at line 978 of file dyngenpar.h.

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

- dyngenpar.h

## 7.30 DynGenPar::StackItem Class Reference

### Public Member Functions

- StackItem ()
  *invalid type*
- StackItem (const QList< StackItem > &parents, CatArg cat, CatArg effCat, int pos, const PseudoCatScope &scope)
  *type 0*
- StackItem (const QList< StackItem > &parents, CatArg cat, CatArg effCat, const PseudoCatScope &scope)
  *type 1*
- StackItem (const StackItem &parent, int dummy)
  *type 2*
- StackItem (const StackItem &parent, const Rule &rule, int len, int curr, int i, const Node &tree, int ruleno, const NextTokenConstraints &nextTokenConstraints)
  *type 3*
- StackItem (const StackItem &parent, CatArg target, int pos, int len)
  *type 4*
- StackItem (const StackItem &parent, CatArg cat, const PseudoCatScope &scope)
  *type 5*
- StackItem (const StackItem &parent, const QList< Node > &leaves, int i, const Node &tree, const Pseudo↩ CatScope &scope, const NextTokenConstraints &nextTokenConstraints)
  *type 6*
- int type () const
- int depth () const
- void addParent (const StackItem &parent)
- void setParents (const QList< StackItem > &parents)
- const StackItemData * data () const
- bool operator< (const StackItem &other) const

### 7.30.1 Detailed Description

Definition at line 488 of file dyngenpar.h.

### 7.30.2 Constructor & Destructor Documentation

#### 7.30.2.1 StackItem() [1/8]

```
DynGenPar::StackItem::StackItem ( )  [inline]
```

invalid type

Definition at line 490 of file dyngenpar.h.

#### 7.30.2.2 StackItem() [2/8]

```
DynGenPar::StackItem::StackItem (
            const QList< StackItem > & parents,
            CatArg cat,
            CatArg effCat,
            int pos,
            const PseudoCatScope & scope )  [inline]
```

type 0

Definition at line 743 of file dyngenpar.h.

#### 7.30.2.3 StackItem() [3/8]

```
DynGenPar::StackItem::StackItem (
            const QList< StackItem > & parents,
            CatArg cat,
            CatArg effCat,
            const PseudoCatScope & scope )  [inline]
```

type 1

Definition at line 750 of file dyngenpar.h.

#### 7.30.2.4 StackItem() [4/8]

```
DynGenPar::StackItem::StackItem (
            const StackItem & parent,
            int dummy )  [inline]
```

type 2

Definition at line 756 of file dyngenpar.h.

#### 7.30.2.5 StackItem() [5/8]

```
DynGenPar::StackItem::StackItem (
            const StackItem & parent,
            const Rule & rule,
            int len,
            int curr,
            int i,
            const Node & tree,
            int ruleno,
            const NextTokenConstraints & nextTokenConstraints )  [inline]
```

type 3

Definition at line 761 of file dyngenpar.h.

#### 7.30.2.6 StackItem() [6/8]

```
DynGenPar::StackItem::StackItem (
            const StackItem & parent,
            CatArg target,
            int pos,
            int len )  [inline]
```

type 4

Definition at line 769 of file dyngenpar.h.

#### 7.30.2.7 StackItem() [7/8]

```
DynGenPar::StackItem::StackItem (
            const StackItem & parent,
            CatArg cat,
            const PseudoCatScope & scope )  [inline]
```

type 5

Definition at line 775 of file dyngenpar.h.

#### 7.30.2.8 StackItem() [8/8]

```
DynGenPar::StackItem::StackItem (
            const StackItem & parent,
            const QList< Node > & leaves,
            int i,
            const Node & tree,
            const PseudoCatScope & scope,
            const NextTokenConstraints & nextTokenConstraints )  [inline]
```

type 6

Definition at line 781 of file dyngenpar.h.

### 7.30.3 Member Function Documentation

#### 7.30.3.1 addParent()

```
void DynGenPar::StackItem::addParent (
            const StackItem & parent )  [inline]
```

Definition at line 508 of file dyngenpar.h.

#### 7.30.3.2 data()

```
const StackItemData* DynGenPar::StackItem::data ( ) const  [inline]
```

Definition at line 510 of file dyngenpar.h.

#### 7.30.3.3 depth()

```
int DynGenPar::StackItem::depth ( ) const  [inline]
```

Definition at line 507 of file dyngenpar.h.

#### 7.30.3.4 operator<()

```
bool DynGenPar::StackItem::operator< (
            const StackItem & other ) const  [inline]
```

Definition at line 511 of file dyngenpar.h.

#### 7.30.3.5 setParents()

```
void DynGenPar::StackItem::setParents (
            const QList< StackItem > & parents )  [inline]
```

Definition at line 509 of file dyngenpar.h.

#### 7.30.3.6 type()

```
int DynGenPar::StackItem::type ( ) const  [inline]
```

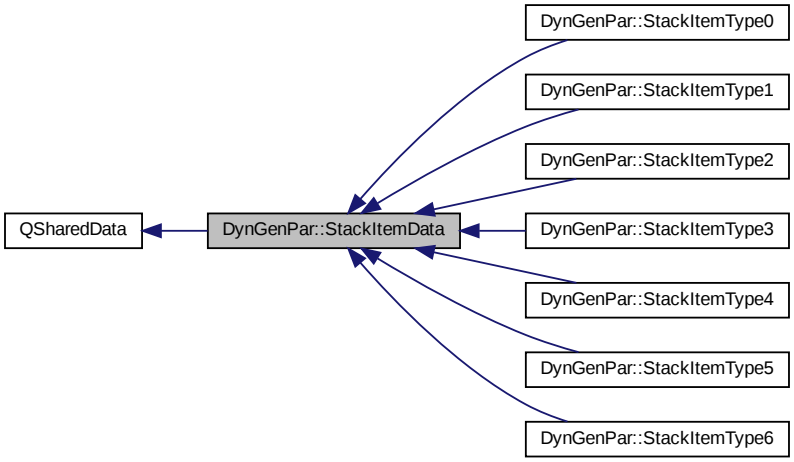
Definition at line 506 of file dyngenpar.h.

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

- dyngenpar.h

## 7.31 DynGenPar::StackItemData Class Reference

Inheritance diagram for DynGenPar::StackItemData:




### Public Member Functions

- StackItemData (int depth)
- virtual ~StackItemData ()
- virtual StackItemData * clone ()=0
- virtual int type () const =0
- int depth () const
- virtual void addParent (const StackItem &parent)=0
- virtual void setParents (const QList< StackItem > &parents)=0

### Protected Attributes

- int m_depth

## 7.31.1 Detailed Description

Definition at line 474 of file dyngenpar.h.

## 7.31.2 Constructor & Destructor Documentation

### 7.31.2.1 StackItemData()

```
DynGenPar::StackItemData::StackItemData (
            int depth )  [inline]
```

Definition at line 476 of file dyngenpar.h.

### 7.31.2.2 ~StackItemData()

```
virtual DynGenPar::StackItemData::~StackItemData ( )  [inline], [virtual]
```

Definition at line 477 of file dyngenpar.h.

## 7.31.3 Member Function Documentation

### 7.31.3.1 addParent()

```
virtual void DynGenPar::StackItemData::addParent (
            const StackItem & parent )  [pure virtual]
```

Implemented in DynGenPar::StackItemType6, DynGenPar::StackItemType5, DynGenPar::StackItemType4, DynGenPar::StackItemType3, DynGenPar::StackItemType2, DynGenPar::StackItemType1, and DynGenPar::StackItemType0.

#### 7.31.3.2 clone()

```
virtual StackItemData* DynGenPar::StackItemData::clone ( )  [pure virtual]
```

Implemented in DynGenPar::StackItemType6, DynGenPar::StackItemType5, DynGenPar::StackItemType4, DynGenPar::StackItemType3, DynGenPar::StackItemType2, DynGenPar::StackItemType1, and DynGenPar::StackItemType0.

#### 7.31.3.3 depth()

```
int DynGenPar::StackItemData::depth ( ) const  [inline]
```

Definition at line 480 of file dyngenpar.h.

#### 7.31.3.4 setParents()

```
virtual void DynGenPar::StackItemData::setParents (
            const QList< StackItem > & parents )  [pure virtual]
```

Implemented in DynGenPar::StackItemType6, DynGenPar::StackItemType5, DynGenPar::StackItemType4, DynGenPar::StackItemType3, DynGenPar::StackItemType2, DynGenPar::StackItemType1, and DynGenPar::StackItemType0.

#### 7.31.3.5 type()

```
virtual int DynGenPar::StackItemData::type ( ) const  [pure virtual]
```

Implemented in DynGenPar::StackItemType6, DynGenPar::StackItemType5, DynGenPar::StackItemType4, DynGenPar::StackItemType3, DynGenPar::StackItemType2, DynGenPar::StackItemType1, and DynGenPar::StackItemType0.

### 7.31.4 Member Data Documentation

#### 7.31.4.1 m_depth

```
int DynGenPar::StackItemData::m_depth  [protected]
```

Definition at line 484 of file dyngenpar.h.

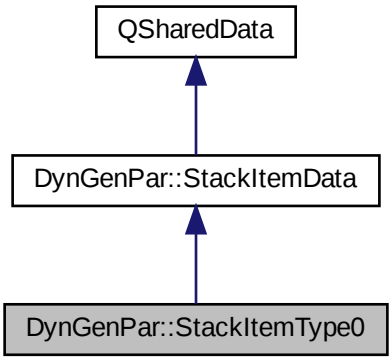
The documentation for this class was generated from the following file:

- dyngenpar.h

## 7.32 DynGenPar::StackItemType0 Class Reference

type 0 item: during match, we're waiting for a token to shift

Inheritance diagram for DynGenPar::StackItemType0:




### Public Member Functions

- StackItemType0 (const QList< StackItem > &parents, CatArg cat, CatArg effCat, int pos, const PseudoCatScope &scope)
- virtual ~StackItemType0 ()
- virtual StackItemData * clone ()
- virtual int type () const
- virtual void addParent (const StackItem &parent)
- virtual void setParents (const QList< StackItem > &parents)
- const QList< StackItem > & parents () const
- Cat cat () const
- Cat effCat () const
- int pos () const
- PseudoCatScope scope () const

### Additional Inherited Members

### 7.32.1 Detailed Description

type 0 item: during match, we're waiting for a token to shift

Definition at line 520 of file dyngenpar.h.

### 7.32.2 Constructor & Destructor Documentation

#### 7.32.2.1 StackItemType0()

```
DynGenPar::StackItemType0::StackItemType0 (
            const QList< StackItem > & parents,
            CatArg cat,
            CatArg effCat,
            int pos,
            const PseudoCatScope & scope )  [inline]
```

Definition at line 522 of file dyngenpar.h.

#### 7.32.2.2 ~StackItemType0()

```
virtual DynGenPar::StackItemType0::~StackItemType0 ( )  [inline], [virtual]
```

Definition at line 531 of file dyngenpar.h.

### 7.32.3 Member Function Documentation

#### 7.32.3.1 addParent()

```
virtual void DynGenPar::StackItemType0::addParent (
            const StackItem & parent )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 534 of file dyngenpar.h.

#### 7.32.3.2 cat()

```
Cat DynGenPar::StackItemType0::cat ( ) const  [inline]
```

Definition at line 548 of file dyngenpar.h.

#### 7.32.3.3 clone()

```
virtual StackItemData* DynGenPar::StackItemType0::clone ( )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 532 of file dyngenpar.h.

#### 7.32.3.4 effCat()

```
Cat DynGenPar::StackItemType0::effCat ( ) const  [inline]
```

Definition at line 549 of file dyngenpar.h.

#### 7.32.3.5 parents()

```
const QList<StackItem>& DynGenPar::StackItemType0::parents ( ) const  [inline]
```

Definition at line 547 of file dyngenpar.h.

#### 7.32.3.6 pos()

```
int DynGenPar::StackItemType0::pos ( ) const  [inline]
```

Definition at line 550 of file dyngenpar.h.

#### 7.32.3.7 scope()

```
PseudoCatScope DynGenPar::StackItemType0::scope ( ) const  [inline]
```

Definition at line 551 of file dyngenpar.h.

#### 7.32.3.8 setParents()

```
virtual void DynGenPar::StackItemType0::setParents (
            const QList< StackItem > & parents )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 539 of file dyngenpar.h.

#### 7.32.3.9 type()

```
virtual int DynGenPar::StackItemType0::type ( ) const  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 533 of file dyngenpar.h.

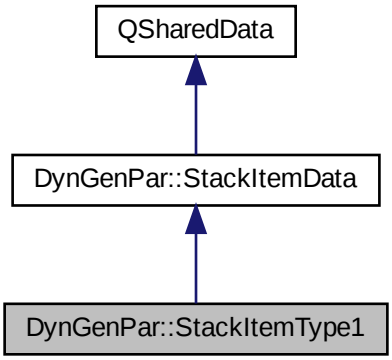
The documentation for this class was generated from the following file:

- dyngenpar.h

## 7.33 DynGenPar::StackItemType1 Class Reference

type 1 item: during type 0 item processing, we're executing a reduce

Inheritance diagram for DynGenPar::StackItemType1:




### Public Member Functions

- StackItemType1 (const QList< StackItem > &parents, CatArg cat, CatArg effCat, const PseudoCatScope &scope)
- virtual ~StackItemType1 ()
- virtual StackItemData * clone ()
- virtual int type () const
- virtual void addParent (const StackItem &)
- virtual void setParents (const QList< StackItem > &)
- const QList< StackItem > & parents () const
- Cat cat () const
- Cat effCat () const
- PseudoCatScope scope () const

### Additional Inherited Members

### 7.33.1 Detailed Description

type 1 item: during type 0 item processing, we're executing a reduce

Definition at line 561 of file dyngenpar.h.

### 7.33.2 Constructor & Destructor Documentation

#### 7.33.2.1 StackItemType1()

```
DynGenPar::StackItemType1::StackItemType1 (
            const QList< StackItem > & parents,
            CatArg cat,
            CatArg effCat,
            const PseudoCatScope & scope )  [inline]
```

Definition at line 563 of file dyngenpar.h.

#### 7.33.2.2 ~StackItemType1()

```
virtual DynGenPar::StackItemType1::~StackItemType1 ( )  [inline], [virtual]
```

Definition at line 572 of file dyngenpar.h.

### 7.33.3 Member Function Documentation

#### 7.33.3.1 addParent()

```
virtual void DynGenPar::StackItemType1::addParent (
            const StackItem &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 575 of file dyngenpar.h.

#### 7.33.3.2 cat()

```
Cat DynGenPar::StackItemType1::cat ( ) const  [inline]
```

Definition at line 584 of file dyngenpar.h.

#### 7.33.3.3 clone()

```
virtual StackItemData* DynGenPar::StackItemType1::clone ( )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 573 of file dyngenpar.h.

#### 7.33.3.4 effCat()

```
Cat DynGenPar::StackItemType1::effCat ( ) const  [inline]
```

Definition at line 585 of file dyngenpar.h.

#### 7.33.3.5 parents()

```
const QList<StackItem>& DynGenPar::StackItemType1::parents ( ) const  [inline]
```

Definition at line 583 of file dyngenpar.h.

#### 7.33.3.6 scope()

```
PseudoCatScope DynGenPar::StackItemType1::scope ( ) const  [inline]
```

Definition at line 586 of file dyngenpar.h.

#### 7.33.3.7 setParents()

```
virtual void DynGenPar::StackItemType1::setParents (
            const QList< StackItem > &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 579 of file dyngenpar.h.

#### 7.33.3.8 type()

```
virtual int DynGenPar::StackItemType1::type ( ) const  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 574 of file dyngenpar.h.

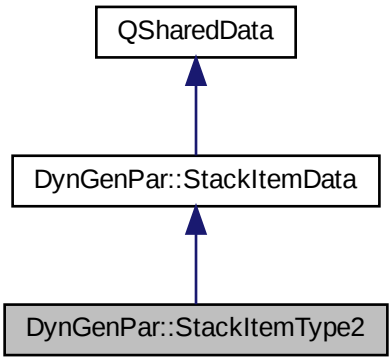
The documentation for this class was generated from the following file:

- dyngenpar.h

## 7.34 DynGenPar::StackItemType2 Class Reference

type 2 item: during reduce, we're recursively executing another reduce

Inheritance diagram for DynGenPar::StackItemType2:




### Public Member Functions

- StackItemType2 (const StackItem &parent)
- virtual ~StackItemType2 ()
- virtual StackItemData * clone ()
- virtual int type () const
- virtual void addParent (const StackItem &)
- virtual void setParents (const QList< StackItem > &)
- const StackItem & parent () const

### Additional Inherited Members

### 7.34.1 Detailed Description

type 2 item: during reduce, we're recursively executing another reduce

This item forms a unification point.

Definition at line 596 of file dyngenpar.h.

### 7.34.2 Constructor & Destructor Documentation

#### 7.34.2.1 StackItemType2()

```
DynGenPar::StackItemType2::StackItemType2 (
            const StackItem & parent )  [inline]
```

Definition at line 598 of file dyngenpar.h.

#### 7.34.2.2 ~StackItemType2()

```
virtual DynGenPar::StackItemType2::~StackItemType2 ( )  [inline], [virtual]
```

Definition at line 600 of file dyngenpar.h.

### 7.34.3 Member Function Documentation

#### 7.34.3.1 addParent()

```
virtual void DynGenPar::StackItemType2::addParent (
            const StackItem &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 603 of file dyngenpar.h.

#### 7.34.3.2 clone()

```
virtual StackItemData* DynGenPar::StackItemType2::clone ( )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 601 of file dyngenpar.h.

#### 7.34.3.3 parent()

```
const StackItem& DynGenPar::StackItemType2::parent ( ) const  [inline]
```

Definition at line 609 of file dyngenpar.h.

#### 7.34.3.4 setParents()

```
virtual void DynGenPar::StackItemType2::setParents (
            const QList< StackItem > &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 606 of file dyngenpar.h.

#### 7.34.3.5 type()

```
virtual int DynGenPar::StackItemType2::type ( ) const  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 602 of file dyngenpar.h.

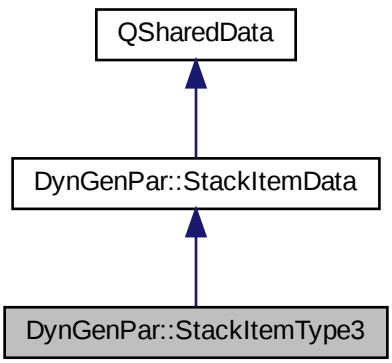
The documentation for this class was generated from the following file:

- dyngenpar.h

## 7.35 DynGenPar::StackItemType3 Class Reference

type 3 item: during matchRemaining, we're executing a match

Inheritance diagram for DynGenPar::StackItemType3:



### Public Member Functions

- StackItemType3 (const StackItem &parent, const Rule &rule, int len, int curr, int i, const Node &tree, int ruleno, const NextTokenConstraints &nextTokenConstraints)
- virtual ~StackItemType3 ()
- virtual StackItemData * clone ()
- virtual int type () const
- virtual void addParent (const StackItem &)
- virtual void setParents (const QList< StackItem > &)
- const StackItem & parent () const
- Rule rule () const
- int len () const
- int curr () const
- int i () const
- Node tree () const
- int ruleno () const
- NextTokenConstraints nextTokenConstraints () const

### Additional Inherited Members

## 7.35.1 Detailed Description

type 3 item: during matchRemaining, we're executing a match

Definition at line 616 of file dyngenpar.h.

## 7.35.2 Constructor & Destructor Documentation

### 7.35.2.1 StackItemType3()

```
DynGenPar::StackItemType3::StackItemType3 (
            const StackItem & parent,
            const Rule & rule,
            int len,
            int curr,
            int i,
            const Node & tree,
            int ruleno,
            const NextTokenConstraints & nextTokenConstraints )  [inline]
```

Definition at line 618 of file dyngenpar.h.

### 7.35.2.2 ~StackItemType3()

```
virtual DynGenPar::StackItemType3::~StackItemType3 ( )  [inline], [virtual]
```

Definition at line 624 of file dyngenpar.h.

### 7.35.3 Member Function Documentation

#### 7.35.3.1 addParent()

```
virtual void DynGenPar::StackItemType3::addParent (
            const StackItem &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 627 of file dyngenpar.h.

#### 7.35.3.2 clone()

```
virtual StackItemData* DynGenPar::StackItemType3::clone ( )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 625 of file dyngenpar.h.

#### 7.35.3.3 curr()

```
int DynGenPar::StackItemType3::curr ( ) const  [inline]
```

Definition at line 636 of file dyngenpar.h.

#### 7.35.3.4 i()

```
int DynGenPar::StackItemType3::i ( ) const  [inline]
```

Definition at line 637 of file dyngenpar.h.

#### 7.35.3.5 len()

```
int DynGenPar::StackItemType3::len ( ) const  [inline]
```

Definition at line 635 of file dyngenpar.h.

#### 7.35.3.6 nextTokenConstraints()

```
NextTokenConstraints DynGenPar::StackItemType3::nextTokenConstraints ( ) const  [inline]
```

Definition at line 640 of file dyngenpar.h.

#### 7.35.3.7 parent()

```
const StackItem& DynGenPar::StackItemType3::parent ( ) const  [inline]
```

Definition at line 633 of file dyngenpar.h.

#### 7.35.3.8 rule()

```
Rule DynGenPar::StackItemType3::rule ( ) const  [inline]
```

Definition at line 634 of file dyngenpar.h.

#### 7.35.3.9 ruleno()

```
int DynGenPar::StackItemType3::ruleno ( ) const  [inline]
```

Definition at line 639 of file dyngenpar.h.

#### 7.35.3.10 setParents()

```
virtual void DynGenPar::StackItemType3::setParents (
            const QList< StackItem > &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 630 of file dyngenpar.h.

#### 7.35.3.11 tree()

```
Node DynGenPar::StackItemType3::tree ( ) const  [inline]
```

Definition at line 638 of file dyngenpar.h.

#### 7.35.3.12 type()

```
virtual int DynGenPar::StackItemType3::type ( ) const  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 626 of file dyngenpar.h.

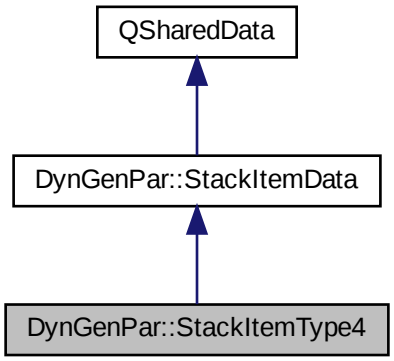
The documentation for this class was generated from the following file:

- dyngenpar.h

## 7.36 DynGenPar::StackItemType4 Class Reference

type 4 item: during reduce, we're executing a matchRemaining

Inheritance diagram for DynGenPar::StackItemType4:




### Public Member Functions

- StackItemType4 (const StackItem &parent, CatArg target, int pos, int len)
- virtual ~StackItemType4 ()
- virtual StackItemData * clone ()
- virtual int type () const
- virtual void addParent (const StackItem &)
- virtual void setParents (const QList< StackItem > &)
- const StackItem & parent () const
- Cat target () const
- int pos () const
- int len () const

### Additional Inherited Members

## 7.36.1 Detailed Description

type 4 item: during reduce, we're executing a matchRemaining

This item forms a unification point.

Definition at line 655 of file dyngenpar.h.

## 7.36.2 Constructor & Destructor Documentation

### 7.36.2.1 StackItemType4()

```
DynGenPar::StackItemType4::StackItemType4 (
            const StackItem & parent,
            CatArg target,
            int pos,
            int len )  [inline]
```

Definition at line 657 of file dyngenpar.h.

### 7.36.2.2 ~StackItemType4()

```
virtual DynGenPar::StackItemType4::~StackItemType4 ( )  [inline], [virtual]
```

Definition at line 660 of file dyngenpar.h.

## 7.36.3 Member Function Documentation

### 7.36.3.1 addParent()

```
virtual void DynGenPar::StackItemType4::addParent (
            const StackItem &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 663 of file dyngenpar.h.

#### 7.36.3.2 clone()

```
virtual StackItemData* DynGenPar::StackItemType4::clone ( )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 661 of file dyngenpar.h.

#### 7.36.3.3 len()

```
int DynGenPar::StackItemType4::len ( ) const  [inline]
```

Definition at line 672 of file dyngenpar.h.

#### 7.36.3.4 parent()

```
const StackItem& DynGenPar::StackItemType4::parent ( ) const  [inline]
```

Definition at line 669 of file dyngenpar.h.

#### 7.36.3.5 pos()

```
int DynGenPar::StackItemType4::pos ( ) const  [inline]
```

Definition at line 671 of file dyngenpar.h.

#### 7.36.3.6 setParents()

```
virtual void DynGenPar::StackItemType4::setParents (
            const QList< StackItem > &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 666 of file dyngenpar.h.

#### 7.36.3.7 target()

```
Cat DynGenPar::StackItemType4::target ( ) const  [inline]
```

Definition at line 670 of file dyngenpar.h.

#### 7.36.3.8 type()

```
virtual int DynGenPar::StackItemType4::type ( ) const  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 662 of file dyngenpar.h.

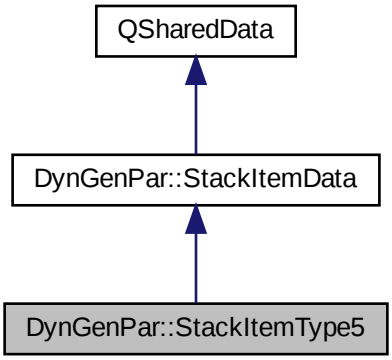
The documentation for this class was generated from the following file:

- dyngenpar.h

## 7.37 DynGenPar::StackItemType5 Class Reference

type 5 item: during match (of an MCFG constraint), we're executing a matchRemaining

Inheritance diagram for DynGenPar::StackItemType5:




### Public Member Functions

- StackItemType5 (const StackItem &parent, CatArg cat, const PseudoCatScope &scope)
- virtual ~StackItemType5 ()
- virtual StackItemData * clone ()
- virtual int type () const
- virtual void addParent (const StackItem &)
- virtual void setParents (const QList< StackItem > &)
- const StackItem & parent () const
- Cat cat () const
- PseudoCatScope scope () const

### Additional Inherited Members

#### 7.37.1 Detailed Description

type 5 item: during match (of an MCFG constraint), we're executing a matchRemaining

Definition at line 682 of file dyngenpar.h.

#### 7.37.2 Constructor & Destructor Documentation

##### 7.37.2.1 StackItemType5()

```
DynGenPar::StackItemType5::StackItemType5 (
            const StackItem & parent,
            CatArg cat,
            const PseudoCatScope & scope )  [inline]
```

Definition at line 684 of file dyngenpar.h.

##### 7.37.2.2 ~StackItemType5()

```
virtual DynGenPar::StackItemType5::~StackItemType5 ( )  [inline], [virtual]
```

Definition at line 688 of file dyngenpar.h.

#### 7.37.3 Member Function Documentation

##### 7.37.3.1 addParent()

```
virtual void DynGenPar::StackItemType5::addParent (
            const StackItem &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 691 of file dyngenpar.h.

#### 7.37.3.2 cat()

```
Cat DynGenPar::StackItemType5::cat ( ) const  [inline]
```

Definition at line 698 of file dyngenpar.h.

#### 7.37.3.3 clone()

```
virtual StackItemData* DynGenPar::StackItemType5::clone ( )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 689 of file dyngenpar.h.

#### 7.37.3.4 parent()

```
const StackItem& DynGenPar::StackItemType5::parent ( ) const  [inline]
```

Definition at line 697 of file dyngenpar.h.

#### 7.37.3.5 scope()

```
PseudoCatScope DynGenPar::StackItemType5::scope ( ) const  [inline]
```

Definition at line 699 of file dyngenpar.h.

#### 7.37.3.6 setParents()

```
virtual void DynGenPar::StackItemType5::setParents (
            const QList< StackItem > &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 694 of file dyngenpar.h.

#### 7.37.3.7 type()

```
virtual int DynGenPar::StackItemType5::type ( ) const  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 690 of file dyngenpar.h.

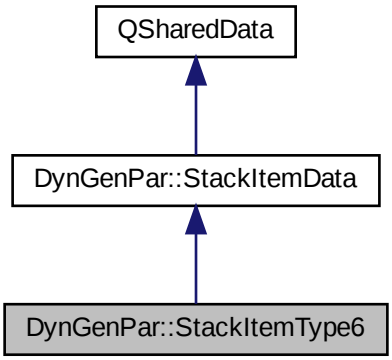
The documentation for this class was generated from the following file:

- dyngenpar.h

## 7.38 DynGenPar::StackItemType6 Class Reference

type 6 item: during match, we're matching a P constraint

Inheritance diagram for DynGenPar::StackItemType6:




### Public Member Functions

- StackItemType6 (const StackItem &parent, const QList< Node > &leaves, int i, const Node &tree, const PseudoCatScope &scope, const NextTokenConstraints &nextTokenConstraints)
- virtual ~StackItemType6 ()
- virtual StackItemData * clone ()
- virtual int type () const
- virtual void addParent (const StackItem &)
- virtual void setParents (const QList< StackItem > &)
- const StackItem & parent () const
- QList< Node > leaves () const
- int i () const
- Node tree () const
- PseudoCatScope scope () const
- NextTokenConstraints nextTokenConstraints () const

### Additional Inherited Members

## 7.38.1 Detailed Description

type 6 item: during match, we're matching a P constraint

Definition at line 708 of file dyngenpar.h.

## 7.38.2 Constructor & Destructor Documentation

### 7.38.2.1 StackItemType6()

```
DynGenPar::StackItemType6::StackItemType6 (
            const StackItem & parent,
            const QList< Node > & leaves,
            int i,
            const Node & tree,
            const PseudoCatScope & scope,
            const NextTokenConstraints & nextTokenConstraints )  [inline]
```

Definition at line 710 of file dyngenpar.h.

### 7.38.2.2 ~StackItemType6()

```
virtual DynGenPar::StackItemType6::~StackItemType6 ( )  [inline], [virtual]
```

Definition at line 716 of file dyngenpar.h.

## 7.38.3 Member Function Documentation

### 7.38.3.1 addParent()

```
virtual void DynGenPar::StackItemType6::addParent (
            const StackItem &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 719 of file dyngenpar.h.

#### 7.38.3.2 clone()

```
virtual StackItemData* DynGenPar::StackItemType6::clone ( )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 717 of file dyngenpar.h.

#### 7.38.3.3 i()

```
int DynGenPar::StackItemType6::i ( ) const  [inline]
```

Definition at line 727 of file dyngenpar.h.

#### 7.38.3.4 leaves()

```
QList<Node> DynGenPar::StackItemType6::leaves ( ) const  [inline]
```

Definition at line 726 of file dyngenpar.h.

#### 7.38.3.5 nextTokenConstraints()

```
NextTokenConstraints DynGenPar::StackItemType6::nextTokenConstraints ( ) const  [inline]
```

Definition at line 730 of file dyngenpar.h.

#### 7.38.3.6 parent()

```
const StackItem& DynGenPar::StackItemType6::parent ( ) const  [inline]
```

Definition at line 725 of file dyngenpar.h.

#### 7.38.3.7 scope()

```
PseudoCatScope DynGenPar::StackItemType6::scope ( ) const  [inline]
```

Definition at line 729 of file dyngenpar.h.

#### 7.38.3.8 setParents()

```
virtual void DynGenPar::StackItemType6::setParents (
            const QList< StackItem > &  )  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 722 of file dyngenpar.h.

#### 7.38.3.9 tree()

```
Node DynGenPar::StackItemType6::tree ( ) const  [inline]
```

Definition at line 728 of file dyngenpar.h.

#### 7.38.3.10 type()

```
virtual int DynGenPar::StackItemType6::type ( ) const  [inline], [virtual]
```

Implements DynGenPar::StackItemData.

Definition at line 718 of file dyngenpar.h.

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

- dyngenpar.h

## 7.39 DynGenPar::Term Struct Reference

term in the expression of a component of a PMCFG function

### Public Member Functions

- Term ()
  *dummy default constructor for bindings*
- Term (int a, int c)
- Term (CatArg t)
- bool isComponent () const
- bool isToken () const
- bool operator== (const Term &other) const
  *needed for QList*

### Public Attributes

- int arg
- int component
- Cat token

## 7.39.1 Detailed Description

term in the expression of a component of a PMCFG function

Definition at line 955 of file dyngenpar.h.

## 7.39.2 Constructor & Destructor Documentation

### 7.39.2.1 Term() [1/3]

```
DynGenPar::Term::Term ( )  [inline]
```

dummy default constructor for bindings

Definition at line 957 of file dyngenpar.h.

### 7.39.2.2 Term() [2/3]

```
DynGenPar::Term::Term (
            int a,
            int c )  [inline]
```

Definition at line 958 of file dyngenpar.h.

### 7.39.2.3 Term() [3/3]

```
DynGenPar::Term::Term (
            CatArg t )  [inline]
```

Definition at line 959 of file dyngenpar.h.

## 7.39.3 Member Function Documentation

#### 7.39.3.1 isComponent()

```
bool DynGenPar::Term::isComponent ( ) const  [inline]
```

Definition at line 965 of file dyngenpar.h.

#### 7.39.3.2 isToken()

```
bool DynGenPar::Term::isToken ( ) const  [inline]
```

Definition at line 966 of file dyngenpar.h.

#### 7.39.3.3 operator==()

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

needed for QList

Definition at line 968 of file dyngenpar.h.

### 7.39.4 Member Data Documentation

#### 7.39.4.1 arg

```
int DynGenPar::Term::arg
```

Definition at line 960 of file dyngenpar.h.

#### 7.39.4.2 component

```
int DynGenPar::Term::component
```

Definition at line 960 of file dyngenpar.h.

#### 7.39.4.3 token

`Cat DynGenPar::Term::token`

This is supposed to be a token in a standard PMCFG, though in the implementation, any context-free category will work.

Definition at line 964 of file dyngenpar.h.

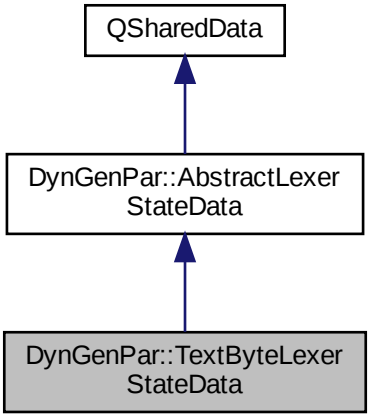
The documentation for this struct was generated from the following file:

- dyngenpar.h

## 7.40 DynGenPar::TextByteLexerStateData Class Reference

You should not have to use this class directly, ever.

Inheritance diagram for DynGenPar::TextByteLexerStateData:




### Public Member Functions

- TextByteLexerStateData (qint64 streamPosition, TextPosition textPosition)
- virtual AbstractLexerStateData * clone ()

### Public Attributes

- qint64 streamPos
- TextPosition textPos

### 7.40.1 Detailed Description

You should not have to use this class directly, ever.

TextByteTokenSource needs lexer states to store the true position, since our token positions don't count the stripped CRs.

This also stores the text position, use TextByteTokenSource::textPosition to retrieve it.

Definition at line 116 of file bytetokensource.h.

### 7.40.2 Constructor & Destructor Documentation

#### 7.40.2.1 TextByteLexerStateData()

```
DynGenPar::TextByteLexerStateData::TextByteLexerStateData (
            qint64 streamPosition,
            TextPosition textPosition )  [inline]
```

Definition at line 118 of file bytetokensource.h.

### 7.40.3 Member Function Documentation

#### 7.40.3.1 clone()

```
virtual AbstractLexerStateData* DynGenPar::TextByteLexerStateData::clone ( )  [inline], [virtual]
```

Implements DynGenPar::AbstractLexerStateData.

Definition at line 121 of file bytetokensource.h.

### 7.40.4 Member Data Documentation

#### 7.40.4.1 streamPos

```
qint64 DynGenPar::TextByteLexerStateData::streamPos
```

Definition at line 124 of file bytetokensource.h.

#### 7.40.4.2 textPos

```
TextPosition DynGenPar::TextByteLexerStateData::textPos
```

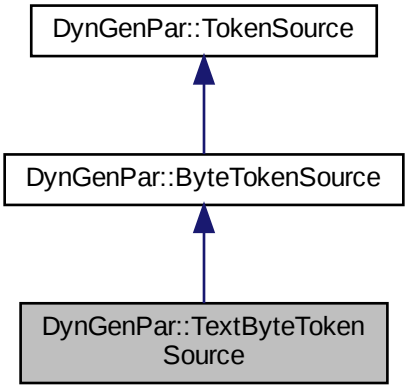
Definition at line 125 of file bytetokensource.h.

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

- bytetokensource.h

## 7.41 DynGenPar::TextByteTokenSource Class Reference

Inheritance diagram for DynGenPar::TextByteTokenSource:




### Public Member Functions

- TextByteTokenSource ()
- TextByteTokenSource (const QString &fileName)
- virtual ~TextByteTokenSource ()
- virtual bool rewindTo (int pos, const LexerState &lexerState=LexerState())

  *We can only rewind if we have a lexer state with the true position.*
- virtual LexerState saveState ()

  *Saves the true stream position (including CRs) and the text position in lines and columns into a lexer state.*

### Static Public Member Functions

- static TextPosition textPosition (const LexerState &lexerState)

  *Retrieves the text position (line and column) stored in the lexer state.*

### Protected Member Functions

- virtual Cat readToken ()

  *Overrides readToken to strip CRs.*
- virtual void reset ()

### Additional Inherited Members

## 7.41.1 Detailed Description

Definition at line 129 of file bytetokensource.h.

## 7.41.2 Constructor & Destructor Documentation

#### 7.41.2.1 TextByteTokenSource() [1/2]

```
DynGenPar::TextByteTokenSource::TextByteTokenSource ( )  [inline]
```

Definition at line 131 of file bytetokensource.h.

#### 7.41.2.2 TextByteTokenSource() [2/2]

```
DynGenPar::TextByteTokenSource::TextByteTokenSource (
            const QString & fileName )  [inline]
```

Definition at line 132 of file bytetokensource.h.

#### 7.41.2.3 ~TextByteTokenSource()

```
virtual DynGenPar::TextByteTokenSource::~TextByteTokenSource ( )  [inline], [virtual]
```

Definition at line 133 of file bytetokensource.h.

## 7.41.3 Member Function Documentation

#### 7.41.3.1 readToken()

```
virtual Cat DynGenPar::TextByteTokenSource::readToken ( )  [inline], [protected], [virtual]
```

Overrides readToken to strip CRs.

Reimplemented from DynGenPar::ByteTokenSource.

Definition at line 174 of file bytetokensource.h.

#### 7.41.3.2 reset()

```
virtual void DynGenPar::TextByteTokenSource::reset ( )  [inline], [protected], [virtual]
```

Reimplemented from DynGenPar::ByteTokenSource.

Definition at line 183 of file bytetokensource.h.

#### 7.41.3.3 rewindTo()

```
virtual bool DynGenPar::TextByteTokenSource::rewindTo (
            int pos,
            const LexerState & lexerState = LexerState() )  [inline], [virtual]
```

We can only rewind if we have a lexer state with the true position.

Reimplemented from DynGenPar::ByteTokenSource.

Definition at line 135 of file bytetokensource.h.

#### 7.41.3.4 saveState()

```
virtual LexerState DynGenPar::TextByteTokenSource::saveState ( )  [inline], [virtual]
```

Saves the true stream position (including CRs) and the text position in lines and columns into a lexer state.

Reimplemented from DynGenPar::TokenSource.

Definition at line 161 of file bytetokensource.h.

#### 7.41.3.5 textPosition()

```
static TextPosition DynGenPar::TextByteTokenSource::textPosition (
            const LexerState & lexerState )  [inline], [static]
```

Retrieves the text position (line and column) stored in the lexer state.

Definition at line 167 of file bytetokensource.h.

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

- bytetokensource.h

## 7.42 DynGenPar::TextPosition Struct Reference

text position

### Public Member Functions

- TextPosition ()
- TextPosition (int l, int c)
- void reset ()
- void countCharacter (unsigned char c)

  *convenience method to count a character*

### Public Attributes

- int line

  *line, zero-based*
- int col

  *column, zero-based*

### 7.42.1 Detailed Description

text position

stored in the lexer state by some text-oriented token sources to allow correct error reporting

Definition at line 929 of file dyngenpar.h.

### 7.42.2 Constructor & Destructor Documentation

#### 7.42.2.1 TextPosition() [1/2]

```
DynGenPar::TextPosition::TextPosition ( )  [inline]
```

Definition at line 930 of file dyngenpar.h.

#### 7.42.2.2 TextPosition() [2/2]

```
DynGenPar::TextPosition::TextPosition (
            int l,
            int c )  [inline]
```

Definition at line 931 of file dyngenpar.h.

### 7.42.3 Member Function Documentation

#### 7.42.3.1 countCharacter()

```
void DynGenPar::TextPosition::countCharacter (
            unsigned char c )  [inline]
```

convenience method to count a character

Definition at line 936 of file dyngenpar.h.

#### 7.42.3.2 reset()

```
void DynGenPar::TextPosition::reset ( )  [inline]
```

Definition at line 933 of file dyngenpar.h.

### 7.42.4 Member Data Documentation

#### 7.42.4.1 col

```
int DynGenPar::TextPosition::col
```

column, zero-based

Definition at line 947 of file dyngenpar.h.

#### 7.42.4.2 line

```
int DynGenPar::TextPosition::line
```

line, zero-based

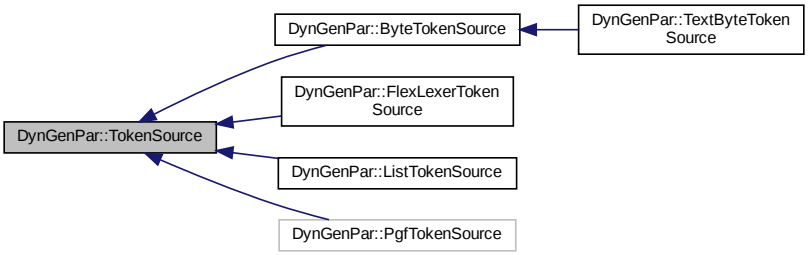
Definition at line 946 of file dyngenpar.h.

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

- dyngenpar.h

## 7.43 DynGenPar::TokenSource Class Reference

Inheritance diagram for DynGenPar::TokenSource:




### Public Member Functions

- TokenSource ()
- virtual ~TokenSource ()
- Cat nextToken ()

  *get the next token from the input, increment current position, save parse tree*
- Node parseTree ()

  *get the parse tree for the last shifted token*
- virtual bool matchParseTree (const Node &treeToMatch)

  *match the parse tree for the last shifted token against the given tree*
- int currentPosition ()

  *get the current input position*
- virtual bool rewindTo (int pos, const LexerState &=LexerState())

  *rewind to an older position (requires buffering)*
- bool rewindTo (int pos, const Node &parseTree, const LexerState &lexerState=LexerState())

  *rewind to an older position (requires buffering) and restore the parse tree (needed for lookahead)*
- virtual LexerState saveState ()

  *saves the current lexer state, by default a null LexerState*

### Protected Member Functions

- virtual Cat readToken ()=0

  *get the next token from the input, to be implemented by subclasses*
- bool simpleRewind (int pos, bool rewindOnly=false)

  *basic implementation of rewindTo for subclasses which support it*

### Protected Attributes

- int currPos
- Node tree

  *sub-parse-tree for hierarchical parsing*

## 7.43.1 Detailed Description

Definition at line 813 of file dyngenpar.h.

## 7.43.2 Constructor & Destructor Documentation

### 7.43.2.1 TokenSource()

```
DynGenPar::TokenSource::TokenSource ( )  [inline]
```

Definition at line 815 of file dyngenpar.h.

### 7.43.2.2 ~TokenSource()

```
virtual DynGenPar::TokenSource::~TokenSource ( )  [inline], [virtual]
```

Definition at line 816 of file dyngenpar.h.

## 7.43.3 Member Function Documentation

### 7.43.3.1 currentPosition()

```
int DynGenPar::TokenSource::currentPosition ( )  [inline]
```

get the current input position

Definition at line 849 of file dyngenpar.h.

#### 7.43.3.2 matchParseTree()

```
virtual bool DynGenPar::TokenSource::matchParseTree (
            const Node & treeToMatch )  [inline], [virtual]
```

match the parse tree for the last shifted token against the given tree

**Returns**

`true` if they should be considered identical for the purposes of a PMCFG rule using the same variable twice, `false` otherwise

The default implementation compares only the category. Other token sources may want to also look at the data and children fields.

Definition at line 845 of file dyngenpar.h.

#### 7.43.3.3 nextToken()

```
Cat DynGenPar::TokenSource::nextToken ( )  [inline]
```

get the next token from the input, increment current position, save parse tree

An epsilon token is returned and currPos is not incremented if the end of input was reached.

Definition at line 822 of file dyngenpar.h.

#### 7.43.3.4 parseTree()

```
Node DynGenPar::TokenSource::parseTree ( )  [inline]
```

get the parse tree for the last shifted token

Definition at line 834 of file dyngenpar.h.

#### 7.43.3.5 readToken()

```
virtual Cat DynGenPar::TokenSource::readToken ( )  [protected], [pure virtual]
```

get the next token from the input, to be implemented by subclasses

Implemented in DynGenPar::ListTokenSource, DynGenPar::TextByteTokenSource, DynGenPar::ByteTokenSource, and DynGenPar::FlexLexerTokenSource.

#### 7.43.3.6 rewindTo() [1/2]

```
virtual bool DynGenPar::TokenSource::rewindTo (
            int pos,
            const LexerState &  = LexerState() )  [inline], [virtual]
```

rewind to an older position (requires buffering)

**Returns**

`true` if successful, `false` otherwise

in all cases, destroys the saved parse tree

By default, only succeeds if the position is the current one, otherwise always returns `false`. Can be overridden by subclasses.

Some subclasses (e.g., file-backed ones) can wind both forward and backward, some can only rewind, some cannot wind at all. The method will return `false` if seeking to the new position is not possible.

Reimplemented in DynGenPar::ListTokenSource, DynGenPar::TextByteTokenSource, and DynGenPar::Byte↩ TokenSource.

Definition at line 862 of file dyngenpar.h.

#### 7.43.3.7 rewindTo() [2/2]

```
bool DynGenPar::TokenSource::rewindTo (
            int pos,
            const Node & parseTree,
            const LexerState & lexerState = LexerState() )  [inline]
```

rewind to an older position (requires buffering) and restore the parse tree (needed for lookahead)

**Returns**

`true` if successful, `false` otherwise

in all cases, restores the saved parse tree to the passed *parseTree*

Relies on the virtual rewindTo to do its actual work.

Definition at line 874 of file dyngenpar.h.

#### 7.43.3.8 saveState()

```
virtual LexerState DynGenPar::TokenSource::saveState ( )  [inline], [virtual]
```

saves the current lexer state, by default a null LexerState

Reimplemented in DynGenPar::TextByteTokenSource.

Definition at line 881 of file dyngenpar.h.

#### 7.43.3.9 simpleRewind()

```
bool DynGenPar::TokenSource::simpleRewind (
            int pos,
            bool rewindOnly = false )  [inline], [protected]
```

basic implementation of rewindTo for subclasses which support it

some subclasses may need additional processing

Definition at line 887 of file dyngenpar.h.

### 7.43.4 Member Data Documentation

#### 7.43.4.1 currPos

```
int DynGenPar::TokenSource::currPos  [protected]
```

Definition at line 894 of file dyngenpar.h.

#### 7.43.4.2 tree

```
Node DynGenPar::TokenSource::tree  [protected]
```

sub-parse-tree for hierarchical parsing

This variable can be set by readToken to a subtree produced by a hierarchical parser, which will be attached to the resulting parse tree in lieu of a leaf node.

If this variable is left unset, a leaf node is automatically produced.

Definition at line 902 of file dyngenpar.h.

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

- dyngenpar.h

# Chapter 8

# File Documentation

## 8.1 bytetokensource.h File Reference

### Classes

- class DynGenPar::ByteTokenSource
- class DynGenPar::TextByteLexerStateData

  *You should not have to use this class directly, ever.*
- class DynGenPar::TextByteTokenSource

### Namespaces

- DynGenPar

### Macros

- #define DYNGENPAR_INTEGER_CATEGORIES

### Enumerations

- enum ByteTokens { ByteTokenEpsilon = 0, ByteTokenNul = 256, ByteTokenError = 257 }

### Functions

- Q_DECLARE_TYPEINFO (ByteTokens, Q_PRIMITIVE_TYPE)

#### 8.1.1 Macro Definition Documentation

#### 8.1.1.1 DYNGENPAR_INTEGER_CATEGORIES

```
#define DYNGENPAR_INTEGER_CATEGORIES
```

Definition at line 25 of file bytetokensource.h.

### 8.1.2 Enumeration Type Documentation

#### 8.1.2.1 ByteTokens

```
enum ByteTokens
```

**Enumerator**

| | |
|---|---|
| ByteTokenEpsilon | |
| ByteTokenNul | we have to remap this because 0 is epsilon |
| ByteTokenError | |

Definition at line 35 of file bytetokensource.h.

### 8.1.3 Function Documentation

#### 8.1.3.1 Q_DECLARE_TYPEINFO()

```
Q_DECLARE_TYPEINFO (
             ByteTokens ,
             Q_PRIMITIVE_TYPE  )
```

## 8.2 dyngenpar.cpp File Reference

### Namespaces

- DynGenPar

### Functions

- uint qHash (const QList< DynGenPar::Cat > &list)

  *simple hash function for lists of categories*
- uint DynGenPar::qHash (const NextTokenConstraints &nextTokenConstraints)

  *simple hash function for next token constraints*
- Node DynGenPar::parseTreeToPmcfgSyntaxTree (const Node &parseTree)

  *converts a parse tree obtained from a PMCFG to a PMCFG syntax tree*

### 8.2.1 Function Documentation

#### 8.2.1.1 qHash()

```
uint qHash (
            const QList< DynGenPar::Cat > & list )
```

simple hash function for lists of categories

Definition at line 437 of file dyngenpar.cpp.

## 8.3 dyngenpar.h File Reference

### Classes

- struct DynGenPar::NextTokenConstraints
  *rule constraints affecting the next token, for scannerless parsing*
- class DynGenPar::Rule
- struct DynGenPar::Cfg
  *An object representing a CFG (or a PMCFG in our internal representation)*
- struct DynGenPar::MultiPrediction
  *multi-token predictions*
- struct DynGenPar::ConstrainedMultiPrediction
  *multi-token predictions with next token constraints*
- struct DynGenPar::FullRule
  *full rule as found in the initial graph*
- class DynGenPar::Alternative
- struct DynGenPar::Node
  *node in the parse tree*
- class DynGenPar::PseudoCatScopeData
- class DynGenPar::PseudoCatScope
- struct DynGenPar::Match
  *(possibly partial) match*
- struct DynGenPar::ActionInfo
  *data passed to parser actions*
- class DynGenPar::ActionDeserializer
  *interface for parser action deserializers*
- class DynGenPar::Action
  *interface for parser actions*
- class DynGenPar::StackItemData
- class DynGenPar::StackItem
- class DynGenPar::StackItemType0
  *type 0 item: during match, we're waiting for a token to shift*
- class DynGenPar::StackItemType1
  *type 1 item: during type 0 item processing, we're executing a reduce*
- class DynGenPar::StackItemType2
  *type 2 item: during reduce, we're recursively executing another reduce*

- class DynGenPar::StackItemType3

  *type 3 item: during matchRemaining, we're executing a match*
- class DynGenPar::StackItemType4

  *type 4 item: during reduce, we're executing a matchRemaining*
- class DynGenPar::StackItemType5

  *type 5 item: during match (of an MCFG constraint), we're executing a matchRemaining*
- class DynGenPar::StackItemType6

  *type 6 item: during match, we're matching a P constraint*
- class DynGenPar::AbstractLexerStateData

  *API for stateful lexers to save their state for rewinding.*
- class DynGenPar::LexerState
- class DynGenPar::TokenSource
- class DynGenPar::ListTokenSource
- struct DynGenPar::TextPosition

  *text position*
- struct DynGenPar::Term

  *term in the expression of a component of a PMCFG function*
- class DynGenPar::Sequence

  *component of a PMCFG function, a sequence of terms*
- class DynGenPar::Function

  *PMCFG function.*
- struct DynGenPar::Pmcfg

  *PMCFG.*
- struct DynGenPar::PmcfgComponentInfo

  *attached to the parse trees as rule labels to allow obtaining syntax trees*
- struct DynGenPar::ParseState

  *parse state struct, for bindings*
- class DynGenPar::Parser

  *main class*

## Namespaces

- DynGenPar

## Macros

- #define IS_EPSILON(cat) ((cat).isNull())

## Typedefs

- typedef QString DynGenPar::Cat

  *Category type: string or integer depending on DYNGENPAR_INTEGER_CATEGORIES.*
- typedef const Cat & DynGenPar::CatArg

  *Category type (string or integer) when passed as an argument.*
- typedef QHash< Cat, QList< Rule > > DynGenPar::RuleSet
- typedef QSet< Cat > DynGenPar::TokenSet
- typedef QSet< Cat > DynGenPar::Predictions
- typedef QMultiHash< QList< Cat >, MultiPrediction > DynGenPar::MultiPredictions
- typedef QMultiHash< Cat, NextTokenConstraints > DynGenPar::ConstrainedPredictions
- typedef QMultiHash< QList< Cat >, ConstrainedMultiPrediction > DynGenPar::ConstrainedMultiPredictions

## Functions

- uint DynGenPar::qHash (const NextTokenConstraints &nextTokenConstraints)

  *simple hash function for next token constraints*
- QDataStream & operator<< (QDataStream &stream, const DynGenPar::Action *data)
- QDataStream & operator>> (QDataStream &stream, DynGenPar::Action *&data)
- uint DynGenPar::qHash (const PseudoCatScope &scope)
- Node DynGenPar::parseTreeToPmcfgSyntaxTree (const Node &parseTree)

  *converts a parse tree obtained from a PMCFG to a PMCFG syntax tree*
- uint qHash (const QList< DynGenPar::Cat > &list)

  *simple hash function for lists of categories*
- QDataStream & operator<< (QDataStream &stream, const DynGenPar::NextTokenConstraints &data)
- QDataStream & operator>> (QDataStream &stream, DynGenPar::NextTokenConstraints &data)
- QDataStream & operator<< (QDataStream &stream, const DynGenPar::Rule &data)
- QDataStream & operator>> (QDataStream &stream, DynGenPar::Rule &data)
- QDataStream & operator<< (QDataStream &stream, const DynGenPar::Cfg &data)
- QDataStream & operator>> (QDataStream &stream, DynGenPar::Cfg &data)

### 8.3.1 Macro Definition Documentation

#### 8.3.1.1 IS_EPSILON

```
#define IS_EPSILON(
              cat ) ((cat).isNull())
```

Definition at line 72 of file dyngenpar.h.

### 8.3.2 Function Documentation

#### 8.3.2.1 operator<<() [1/4]

```
QDataStream & operator<< (
             QDataStream & stream,
             const DynGenPar::Action * data )  [inline]
```

Definition at line 1463 of file dyngenpar.h.

#### 8.3.2.2 operator<<() [2/4]

```
QDataStream& operator<< (
             QDataStream & stream,
             const DynGenPar::NextTokenConstraints & data )  [inline]
```

Definition at line 1434 of file dyngenpar.h.

#### 8.3.2.3 operator<<() [3/4]

```
QDataStream& operator<< (
            QDataStream & stream,
            const DynGenPar::Rule & data )  [inline]
```

Definition at line 1444 of file dyngenpar.h.

#### 8.3.2.4 operator<<() [4/4]

```
QDataStream& operator<< (
            QDataStream & stream,
            const DynGenPar::Cfg & data )  [inline]
```

Definition at line 1454 of file dyngenpar.h.

#### 8.3.2.5 operator>>() [1/4]

```
QDataStream & operator>> (
            QDataStream & stream,
            DynGenPar::Action *& data )  [inline]
```

Definition at line 1471 of file dyngenpar.h.

#### 8.3.2.6 operator>>() [2/4]

```
QDataStream& operator>> (
            QDataStream & stream,
            DynGenPar::NextTokenConstraints & data )  [inline]
```

Definition at line 1439 of file dyngenpar.h.

#### 8.3.2.7 operator>>() [3/4]

```
QDataStream& operator>> (
            QDataStream & stream,
            DynGenPar::Rule & data )  [inline]
```

Definition at line 1450 of file dyngenpar.h.

#### 8.3.2.8 operator>>() [4/4]

```
QDataStream& operator>> (
             QDataStream & stream,
             DynGenPar::Cfg & data )  [inline]
```

Definition at line 1459 of file dyngenpar.h.

#### 8.3.2.9 qHash()

```
uint qHash (
             const QList< DynGenPar::Cat > & list )
```

simple hash function for lists of categories

Definition at line 437 of file dyngenpar.cpp.

## 8.4 flexlexertokensource.h File Reference

### Classes

- class DynGenPar::FlexLexerTokenSource

### Namespaces

- DynGenPar

## 8.5 pgf.cpp File Reference

### Namespaces

- DynGenPar

### Macros

- #define CHECK_STATUS()

### 8.5.1 Macro Definition Documentation

#### 8.5.1.1 CHECK_STATUS

```
#define CHECK_STATUS( )
```

**Value:**

```
if (stream.status() != HaskellDataStream::Ok) \
                        qFatal("invalid PGF file or wrong version of GF")
```

## 8.6 pgf.h File Reference

### Classes

- struct DynGenPar::Pgf

  *representation of the information in .pgf files in a format we can process*
- class DynGenPar::PgfParser

  *parser for PGF grammars*

### Namespaces

- DynGenPar

### Macros

- #define DYNGENPAR_INTEGER_CATEGORIES
- #define STATIC static

### Enumerations

- enum DynGenPar::PgfReservedTokens {
  DynGenPar::PgfTokenEpsilon, DynGenPar::PgfTokenLexError, DynGenPar::PgfTokenVar, DynGenPar::↩
  PgfTokenFloat,
  DynGenPar::PgfTokenInt, DynGenPar::PgfTokenString }

### Variables

- STATIC const char ∗const DynGenPar::PreludeBind = "&+"

### 8.6.1 Macro Definition Documentation

#### 8.6.1.1 DYNGENPAR_INTEGER_CATEGORIES

```
#define DYNGENPAR_INTEGER_CATEGORIES
```

Definition at line 28 of file pgf.h.

#### 8.6.1.2 STATIC

```
#define STATIC static
```

Definition at line 47 of file pgf.h.

# Bibliography

[1] Kevin Kofler and Arnold Neumaier. DynGenPar – A Dynamic Generalized Parser for Common Mathematical Language. In *Proceedings of CICM 2012 (DML track)*, volume 7362 of *LNAI*. Springer, 2012. 1, 7

[2] Peter Schodl, Arnold Neumaier, Kevin Kofler, Ferenc Domes, and Hermann Schichl. Towards a Self-reflective, Context-aware Semantic Representation of Mathematical Specifications. In Josef Kallrath, editor, *Algebraic Modeling Systems – Modeling and Solving Real World Optimization Problems*, chapter 2. Springer Verlag, 2012. 4

# Index