**Recursive Descent Parser**



API

Recursive Descent Parser API

# Layers

1. Simple Parser
2. Generic Parser
3. Domain parser
4. English Parser

# Simple Parser Layer

|  |  |  |
| --- | --- | --- |
| **Mixin Classes** | **Token Classes** | **Parse Tree Classes** |
| TOKEN-MIXIN | TOKEN | PARSE-NODE |
| ALPHABETIC-TOKEN-MIXIN | ALPHABETIC-TOKEN | INTERNAL-PARSE-NODE |
| NUMERIC-TOKEN-MIXIN | NUMERIC-TOKEN | LEAF-PARSE-NODE |
| SPECIAL-TOKEN-MIXIN | SPECIAL-TOKEN | UNUSED-PARSE-NODE |
|  | ALPHABETIC-SPECIAL-TOKEN | PARSE-TREE |
| **Grammar Classes** | ALPHA-NUMERIC-TOKEN |  |
| Grammar | NUMERIC-SPECIAL-TOKEN |  |
|  | ALPHA-NUMERIC-SPECIAL-TOKEN |  |

(DEFINE-GRAMMAR <name> <productions> <terminal-symbols> &key <class>)

(SCAN-TOKENS <string> <grammar>)

(SCAN-TOKEN <token>)

The argument <token> can be a string, number, character or token object.

(SIMPLE-PARSER <token-string> <grammar>)

(TERMINAL-SYMBOL-PREDICATE <terminal-symbol> <grammar>)

(APPLY-TERMINAL-SYMBOL-PREDICATE <terminal-symbol > <value> <grammar>)

(NON-TERMINAL-SYMBOL-P <NTS> <grammar>)

(NON-TERMINAL-SYMBOL-PATTERNS <NTS> <grammar>)

# Generic Parser Layer

**CONTEXT-GRAMMAR**

This class inherits directly from the GRAMMAR class of the previous layer and adds a context slot to store the context associated with a particular context grammar. The value of the context is entirely up to the user of this class and its value will be supplied when invoking the terminal symbol predicates of the context grammar.

**(DEFINE-CONTEXT-GRAMMAR <name> <productions>< terminal-symbols>< context> &key <class>)**

**(SET-GRAMMAR-CONTEXT <context-grammar><context>)**

**(WITH-GRAMMAR-CONTEXT ( <context-grammar> <context>) &body body)**

**(APPLY-TERMINAL-SYMBOL-PREDICATE <terminal-symbol> <value> <context-grammar>)**

This method redefines the simple-parser's method on the grammars of class GRAMMAR for those of class CONTEXT-GRAMMAR. This function looks up the terminal symbol predicate associated to the specified <terminal-symbol> and then applies the predicate to the specified <value> and the value of the grammar's context slot.

**(SCAN-TOKENS :around <string> <context-grammar>)**

This :around method provides the infrastructure needed to map token objects returned by the simple parser's SCAN-TOKEN functionality to domain language objects. Because ultimately these objects will be manipulated by the terminal symbol predicate function supplied as part of the domain knowledge for the grammar, the parser need not be aware of this mapping or of any domain specific objects and classes and yet will still be able to return these objects as part of the parsing process.

**(ANALYZE-TOKENS <tokens> <context-grammar>)**

The contract for ANALYZE-TOKENS is that it returns as multiple values both the newly mapped domain objects as the first value and the original scanned tokens as a second value. Since this default mapping does nothing the originally scanned tokens are returned as both first and second values. This method on the class on the class CONTEXT-GRAMMAR simply exists to provide a primary method for the CONTEXT-GRAMMAR class. It is expected that users that specialize the context-grammar will provide more interesting methods for this generic function. In other words implementers of this method on a particular subclass of context-grammar class should provide a useful token to domain object mapping.

Note: ANALYZE-TOKENS will only be invoked by SCAN-TOKENS on grammar classes that inherit from the CONTEXT-GRAMMR class.

# Domain Parser Layer

This layer presupposes the mapping of token objects to domain objects and provides various facilities for facilitating the defining of such a mapping. In particular it introduces the notion of *token expansions* which allows the decomposition of a single token that matches a particular token pattern into several tokens and the notion of *phrase transformations* which allows the reordering of tokens, insertion of new tokens and deletion of existing tokens within sequences of tokens that match a particular token sequence pattern.

In addition this layer also introduces a sort of *abstract lexicon data type* again aimed at facilitating the definition and implementation of a token object to domain object mapping.

The domain parser layer introduces two new parser related classes, these are the domain-grammar and domain-word classes.

**DOMAIN-GRAMMAR**

This class extends the CONTEXT-GRMMAR class of the previous layer and provides a lexicons slot allowing domain-knowledge to be associated with the grammar object itself.

**DOMAIN-WORD**

This class provides a basic domain object class that can be used in the mapping of token objects to domain-object in methods on ANALYZE-TOKENS and ANALYZE-TOKEN. It is implemented as a wrapper class around token objects.

**(DEFINE-CHARACTER-TOKEN-CLASS <character> <character-name> &optional <mixin-classes>)**

The following token classes are predefined in the domain parser layer and are used by the SCAN-TOKEN method on grammars of class DOMAIN-GRAMMAR.

|  |  |  |
| --- | --- | --- |
| **Character** | **Character Name** | **Token Class** |
| . | Period | Period-Token |
| , | Comma | Comma-Token |
| ; | Semicolon | Semicolon-Token |
| : | Colon | Colon-Token |
| ! | Exclamation | Exclamation-Token |
| ? | Question | Question-Token |
| ( | Open-Parenthesis | Open-Parenthesis-Token |
| ) | Close-Parenthesis | Close-Parenthesis-Token |

**(DEFINE-LEXICON <lexicon-name> &key <fields>)**

# English Parser Layer

(DEFINE-ENGLISH-WORD-CLASS <word-class> <word-type> &optional <slots>)

|  |  |
| --- | --- |
| **English Word Class** | **Word Type** |
| ENGLISH-WORD |  |
| ENGLISH-NOUN |  |
| ENGLISH-PROPER-NOUN |  |
| ENGLISH-PREPOSITION |  |
| ENGLISH-ADJECTIVE |  |
| ENGLISH-ADVERB |  |
| ENGLISH-VERB |  |
| ENGLISH-CONJUNCTION |  |
| ENGLISH-PRONOUN |  |

\*ENGLISH-GRAMMAR\*

(WORD-PART-OF-SPEECH-P <token> <part-of-speech>)

(NOUN-PART-OF-SPEECH-P <token> <part-of-speech>)

\*PART-OF-SPEECH-CACHE\*

(FIND-WORD-PART-OF-SPEECH <word>)

(SAVE-WORDS-PARTS-OF-SPEECH &optional <filename>)

(LOAD-WORDS-PARTS-OF-SPEECH &optional <filename>)

(HAS-SUFFIX-P <word> <suffix>)

(REMOVE-SUFFIX <word> <suffix>)

(REPLACE-SUFFIX <word> <old-suffix><new-suffix>))

(SCAN-TOKENS <string> <English-grammar>)

This method augments the simple-scanner method to separate out english punctuation characters that are glued to the original tokens.

## Lexicons API

(ADD-LEXICONS <lexicons>)

(FIND-LEXICONS <lexicons-name>)

(DELETE-LEXICONS <lexicons-name>)

(ENSURE-LEXICONS <lexicons-name>)

## Lexicon API

(ADD-LEXICON <lexicon> <lexicons>)

(FIND-LEXICON <lexicon-name> <lexicons>)

(ENSURE-LEXICON <lexicon-name> <lexicons>)

(DELETE-LEXICON <lexicon-name> <lexicons>)