Designbeschreibung der Applikation „GeoViewer“

1. Allgemeines:

Die Java – Applikation GeoViewer ist eine menügesteuerte graphische Anwendung, mit deren Hilfe sich geometrische Konfigurationen visualisieren lassen.
Als Voraussetzung zum starten dieser eigenständigen Applikationen benötigt man JAVA JRE 1.4, das Computeralgebrasystem Maple(ab V5.0), das dazugehörige GeoProver-Paket für Maple sowie GEO-Recordes für die Eingabe. GEO-Records enthalten Beschreibungen geometrischer Beweisschematas, die in einem speziellen XML-Format gespeicht sind.
Diese GEO-Records enthalten, wie gesagt, neben den Beschreibungen der geometrischen Konfigurationen eine Liste von unabhängigen Parametern, die für die konkrete Visualisierung verantwortlich sind. Die Parameter haben Einfluss auf die Lage einzelner geometrischer Objekte und somit auch auf die Lage abgeleiteter Objekte.
GeoViewer verfügt zur Bedienung über eine grafische Nutzerschnittstelle mit allgemein üblichen Fenster-Techniken.
Es ist zu Vermerken, dass eine „Dynamisierung“ der Visualisierungen durch direkte Mausmanipulation nicht implementiert ist. Parameterwerte einer Visualisierung können nur über ein Dialogfeld geändert werden.

2. Produktübersicht:

GeoViewer wird durch den Kommandozeilenaufruf

java geometry.GeoViewer

im Wurzelverzeichnis der Pakethierarchie gestartet. Wenn dieser Kommandozeilenaufruf erfolgt ist stellte die Applikation eigenständige eine Verbindung zu Maple her und startet bei erfolgreicher Initialisierung von Marple das Hauptfenster.
Das Hauptfenster in GeoViewer ist, von oben nach unten, in folgende Komponente untergliedert:

- Titelleiste
- Menüleiste
- Symbolleisten
- Desktop-Fläche
- Statusleiste

(1) Titelleiste: Diese bildet den oberen Abschluss des Fensters. Sie enthält, neben dem Namen der Applikation, grundsätzliche Fensterfunktionen, wie die Minimierung, die Maximierung oder die Terminierung des Fensters.

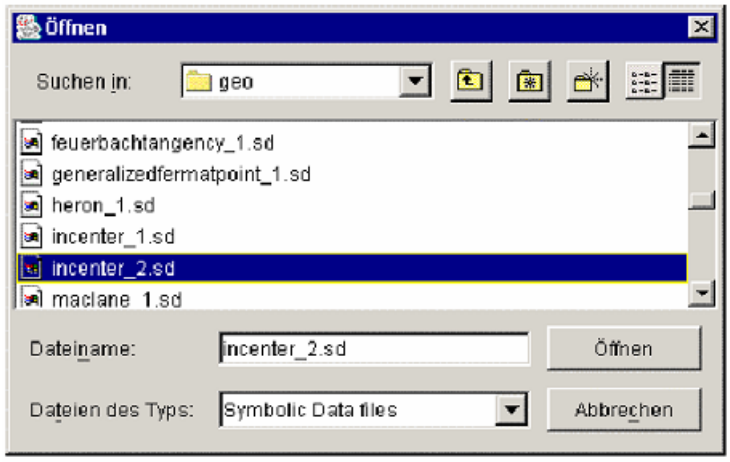
(2) Menüleiste: Diese ist in grundsätzliche Gruppen unterteilt: File, Scene, Windows, Help;
Diese Menüeinträge sind wiederrum unterteilt, so das eine intuitive Zuordnung der Applikationsfunktionen gegeben ist.

(3) Symbolleisten: Hier sind ausgewählte Elemente symbolisch dargestellt. Bei diesen handelt es sich zumeist um die Elemente, die statistisch gesehen am häufigsten benutzt werden. Mittels der Icon-piktogramme ist eine funktionsdefinition der Symbole gegeben.

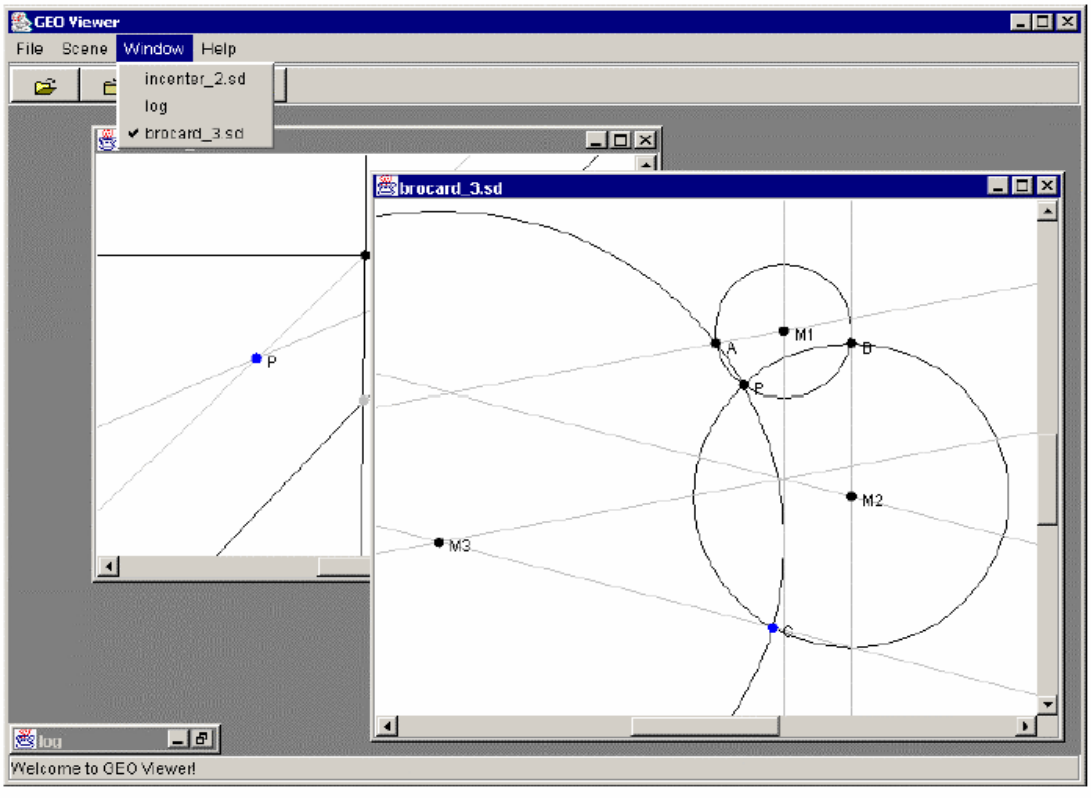
(4) Dektop-Fläche: In diese Fläche lassen sich im weiteren Verlauf die verschiedenen Zeichenflächen, sowie ein spezielles Fenster für Fehlermeldungen öffnen.

(5) Statusleiste: Diese Leiste befindet sich am unteren Rand und bildet somit den Abschluss des Fenster. In dieser erhält der Nutzer einen groben Überblick über seine momentane Arbeit, so zum Beispiel den Status der Speicherung eines Geo-Records.

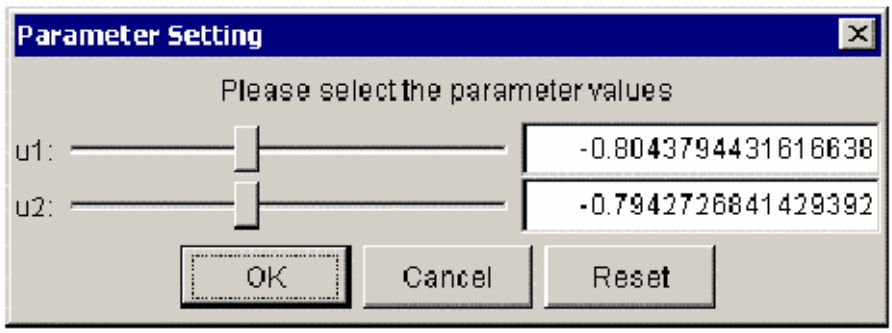
Über den im Punkt (2) beschriebenen Menüeintrag *File* oder das *linke Icon* lasst sich nach dem starten des Programmes nun ein GEO-Record laden.



Handelt es sich um ein konstruktives Schema, so wird die zugehörige Visualisierung generiert und in einer neuen Zeichenfläche im Desktopbereich angezeigt und aktiviert. Die Parameterwerte werden dabei zunächst mit Zufallszahlen initialisiert.



Es können ebenfalls mehrere GEO-Records geladen werden. Diese verschiedenen Fenstern der Desktopfläche können nun abwechseln, mit der Maus, aktiviert werden.
Alternativ kann das aktuelle Fenster auch über den Menüeintrag *Window* gewechselt werden.
Parameterwerte eines aktivierten Fensters(aktivierte Zeichenfläche) können über einen Menüeintrag geändert werden, welcher eine Dialogfenster



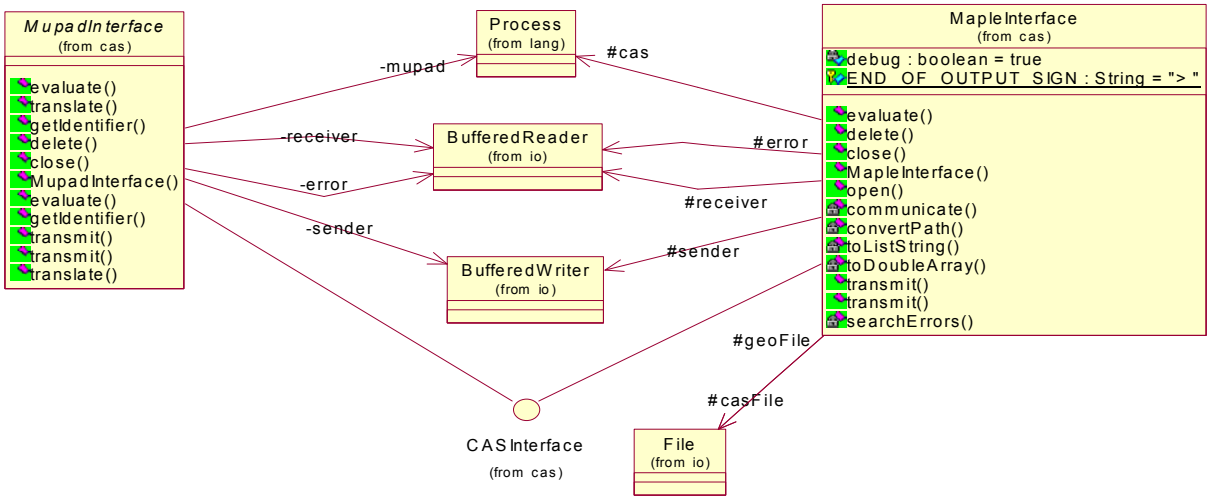
mit Schieberegeln zum modifizieren der Parameterwerte öffnet.
Danach werden die Koordinaten der einzelnen Objekte von Maple neu berechnet und die Visualisierung in der aktiven Zeichenfläche neu dargestellt.
Die aktiven Fenster, bzw. die Parameter die im Fenster visualisiert worden, können wieder in GEO-Records abgelegt werden und das aktive Fenster einen weiteren Menüpunkt, Icon oder einen speziellen Knopf am Fensterrand geschlossen werden.

Die Applikation kann dann anschliessen über den Exit-Eintrag im File -Menü beendet, oder dem Icon in der Titelleiste beendet werden.

3. Grundsätzliche Design-Entscheidungen:

Dieses Geometrieprogramm ist in der Programmiersprache Java implementiert wurden. Dadurch ist das Programm plattformunabhängig und kann somit auf verschiedenen Plattformen eingesetzt werden, genau wie das Computeralgebrasystem „Maple“ das von diesem Programm verwendet wird. Dadurch kann das Geometrieprogramm überall dort verwendet werden, wo auch „Maple“ genutzt wird.

Damit dieses Geometrieprogramm nicht zu umfangreich wird, entschied sich der Autor des Programms, die Berechnungen durch das Computeralgebrasystem Maple durchführen zu lassen. Damit die Kommunikation mit Maple reibungslos funktioniert, wurde eine spezielle Klasse implementiert, welche die Schnittstelle zu Maple ist. Diese Klasse trägt den Namen „MapleInterface“.
Außerdem existiert auch eine Schnittstelle zu dem Computeralgebrasystem Mupad. Diese Schnittstelle wird durch die Klasse MupadInterface realisiert.
Beide Klassen implementieren das Interface CASInterface. Dieses Interface ist für die Ausnahmebehandlung, welche bei den Berechnungen und beim Datenaustausch auftreten können, verantwortlich.



In dem Paket „Controller“ werden die Informationen über die verschiedenen geometrischen Objekte zusammengefasst. Dabei wird zunächst eine allgemeine Oberklasse „GeoElement“ angelegt. Von dieser werden dann die speziellen geometrischen Elemente wie Kreis, Winkel, Punkt, Gerade usw. abgeleitet.
Dabei dienen diese Klassen nur zur Zusammenfassung der Kenngrößen der geometrischen Objekte, aber sie beinhalten keine Funktionalität zur graphischen Darstellung dieser Objekte.

In dem Paket „parser“ sind verschiedene Klassen angesiedelt, welche der Parsergenerator CUP verwendet. CUP ist ein spezieller Parsergenerator welcher einen LALR-Parser generiert. Dies ist ein Parser für arithmetische Ausdrücke.

Weiterhin wird auch der Scannergenerator JLex verwendet, welcher einen Scanner erzeugt. Sowohl der Scanner als auch der Parser sind speziell an Java angepasst.
Das ganze funktioniert folgendermaßen:
Der Eingabetext wird an den Scanner übergeben, welcher den Text in Symbole, so genannte Tokens zerlegt. Diese werden dann an den Parser weitergegeben. Da der Parser aber als Symbole nur Objekte der Klasse „Symbol“ erwartet, sollte der Scanner auch nur derartige Objekte liefern. Um welche Art von Symbol es sich handelt, wird über Konstanten der Klasse „sym“ festgelegt. Diese Klasse wird von CUP generiert.
Die Symbole werden dann von dem Parser analysiert und er gibt das Ergebnis der Analyse an die aufrufende Klasse zurück.
Also werden weitere externe Tools verwendet.

In dem Paket „view“ sind, wie der Name schon sagt, die Klassen abgelegt, welche für die Darstellung der graphischen Benutzeroberfläche zuständig sind. So findet man hier zum Beispiel die Klasse „ApplicationFrame“, welche das Hauptfenster der Applikation erzeugt. Diese Klasse wurde aus der vordefinierten Klasse JFrame abgeleitet, damit alle vorgefertigten Methoden für ein typisches Fenster weiterverwendet werden können. Auch die Klasse „AboutBox“ , welche für die Darstellung eines Dialogfensters zuständig ist, gehört zu diesem Paket. Weitere Klassen, welche für die graphische Darstellung verantwortlich sind, sind diese folgenden Klassen: „ElementGraphics“, „LogFrame“, „ParameterBox“ , „SceneFrame“ und „WindowMenu“.
In diesem Paket sind aber auch die Klassen angeordnet, welche für die Ereignisbehandlung der graphischen Oberfläche zuständig sind. So ist zum Beispiel die Klasse „ExitAction“ dafür da, um auf exitActions zu reagieren, also wenn jemand ein Fenster schließen will, dann muss diese Klasse darauf mit einer Action reagieren. Auch zu erwähnen ist die Klasse „DesktopEvent“. Diese Klasse dient dazu, bei einem Fensterwechsel, den Focus an das neue Fenster zu übergeben. Weitere Klassen, welche für die Ereignisbehandlung zuständig sind, sind diese Klassen: „AboutAction“, „CloseAction“, „DesktopAdapter“, „OpenAction“ und „ParameterAction“.
Weiterhin gibt es hier auch noch die Klasse „ElementGraphicsException“, welche aufgerufen wird, wenn ein Fehler bzw. eine Ausnahme beim Zeichnen eines geometrischen Objektes auftritt.
Es ist somit ersichtlich, dass der Autor eine Trennung zwischen der graphischen Darstellung der Elemente, wie Fenster, Dialogboxen und auch der geometrischen Objekte und den programmlogischen Funktionen dieser Elemente durchführt. Damit wurde dem MVC-Konzept Rechnung getragen. Dies dient vor allem dazu, die Übersichtlichkeit bei mittleren und großen Softwareprojekten zu wahren. Aber auch die Wiederverwendbarkeit des Quellcodes wird dadurch erhöht, denn man kann nun beispielsweise die selbe graphische Darstellung mit komplett anderen Programmlogischen Funktionen ausstatten bzw. die vorhandenen Funktionen sehr leicht erweitern.

Nun folgt eine genauere Untersuchung der Klasse „java_cup“.
Alle Klassen die in diesem Paket zusammengefasst sind, werden für den Parsergenerator CUP verwendet bzw. von diesem erzeugt. Das gleiche gilt für die beiden Unterpakete von „java_cup“. Dies sind die Pakete „runtime“ und „simple_calc“.

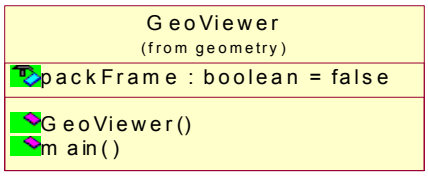
Schließlich betrachten wir das Paket JLex. Dieses Paket enthält nur eine einzige Klasse mit dem Namen „Main“. Diese Klasse und auch das Paket wurden von dem Scannergenerators JLex erzeugt.

4. Paket und Klassenstruktur:

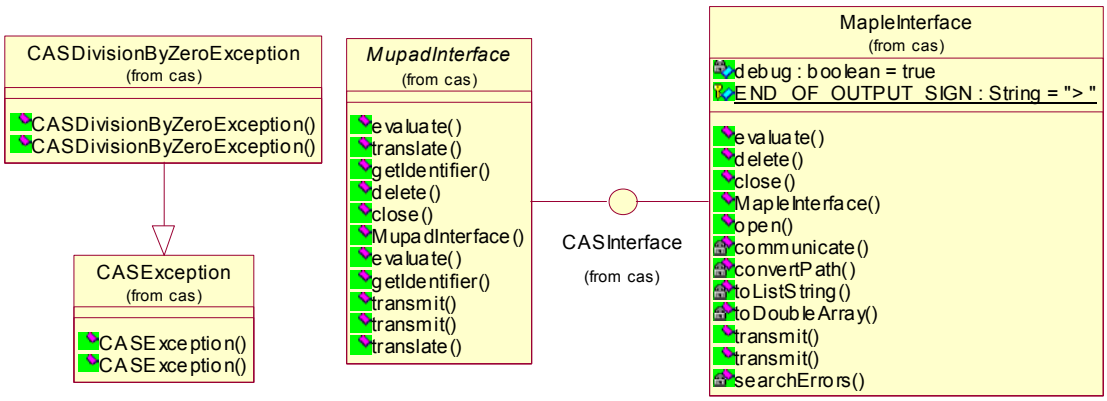
Das Programm „GeoViewer“ besitzt folgende Paketstruktur:

- geometry
  - cas
  - controller
  - parser
  - view
- java_cup
  - runtime
  - simple_calc
- Jlex

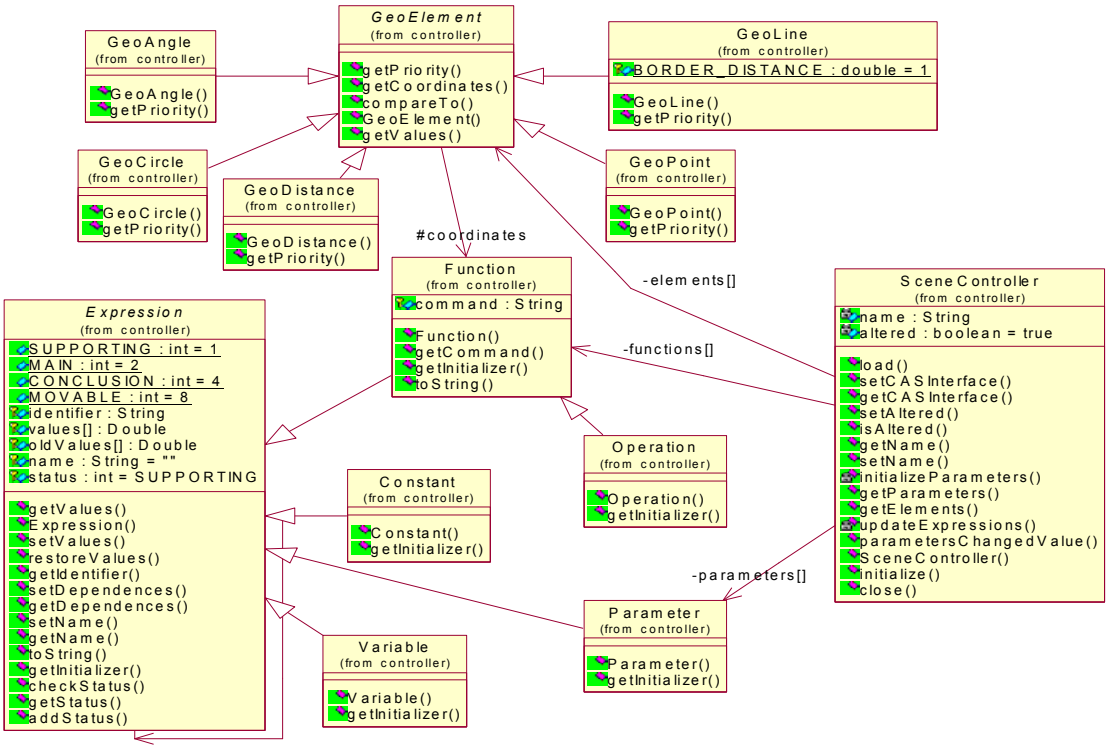
Das Paket „geometry“ besitzt direkt nur eine Klasse. Ihre Bezeichnung lautet „GeoViewer“:



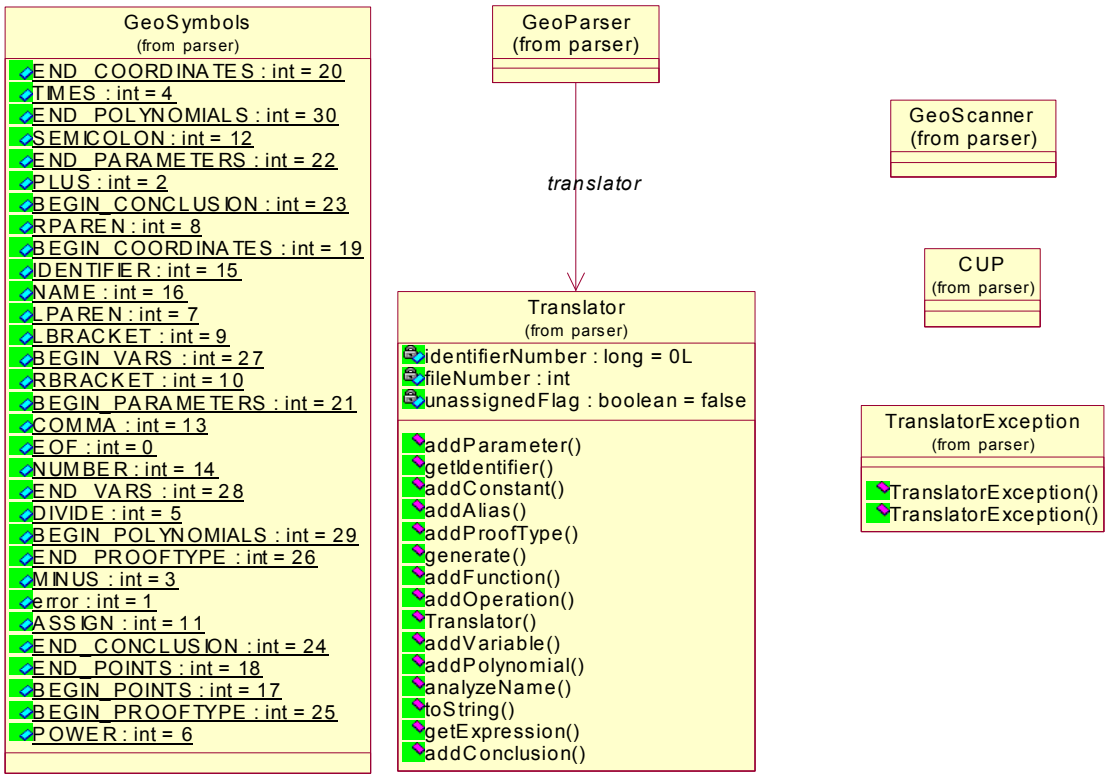
Das Unterpaket von „geometry“ , mit Namen „cas“ umfasst folgende Klassen:



Ein weiteres Unterpaket von „geometry“ ist „controller“. Dieses Paket hat folgende Klassen:

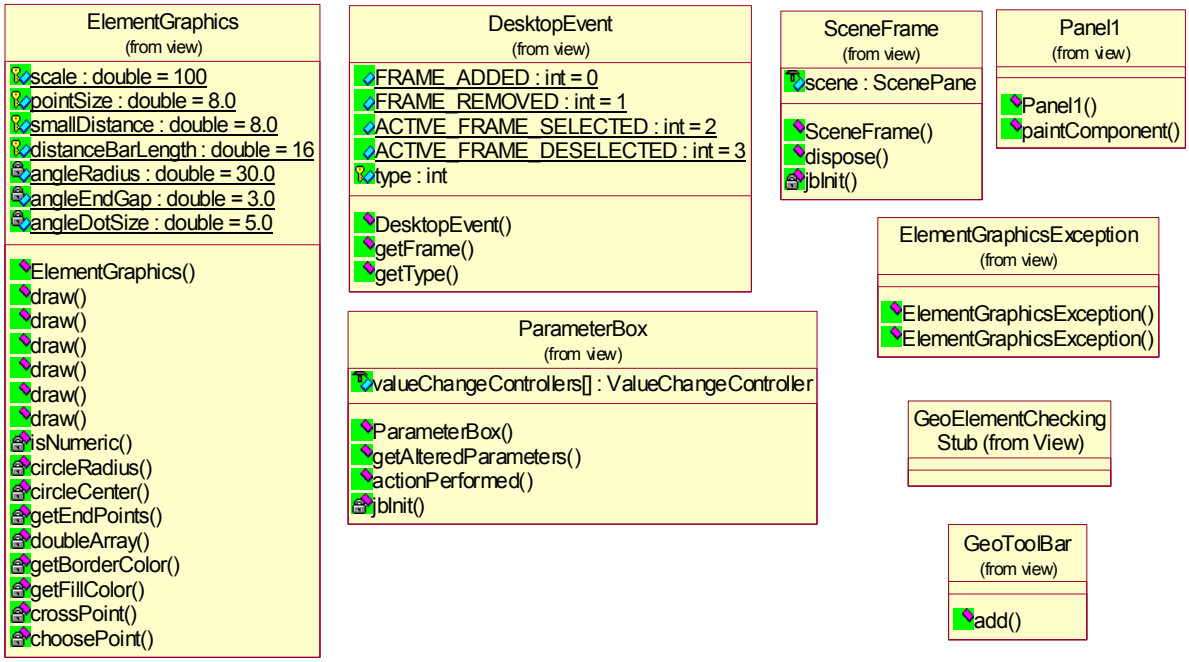


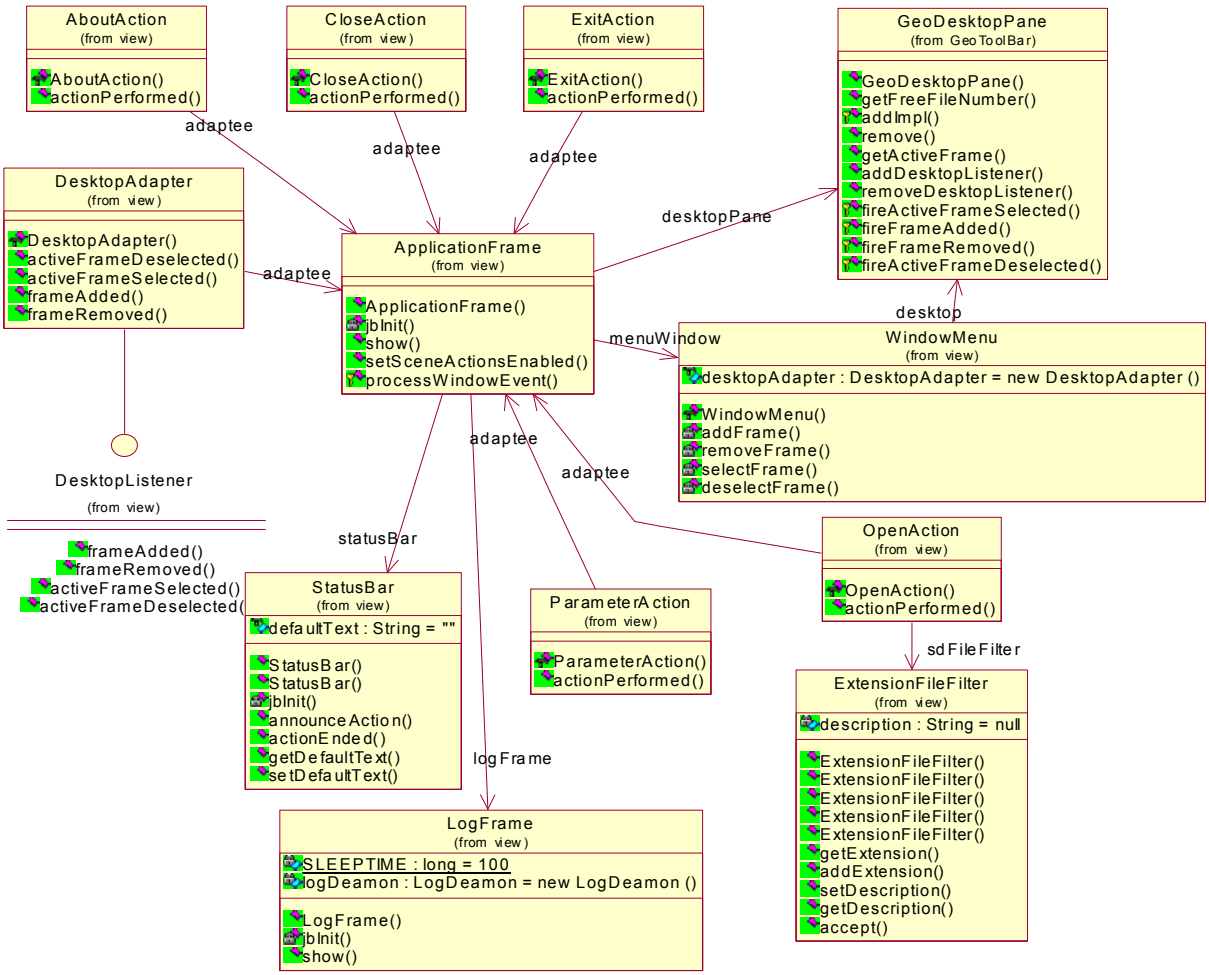
Nun folgt das Paket „parser“. Es beinhaltet folgende Klassen:



Bemerkung:
Aufgrund des Auftretens eines Fehlers beim Reverse Engineering kann leider keine genauere Darstellung der Klassen GeoParser und GeoScanner erfolgen.

Nun folgt die graphische Darstellung der Klassen des Paketes „view“:

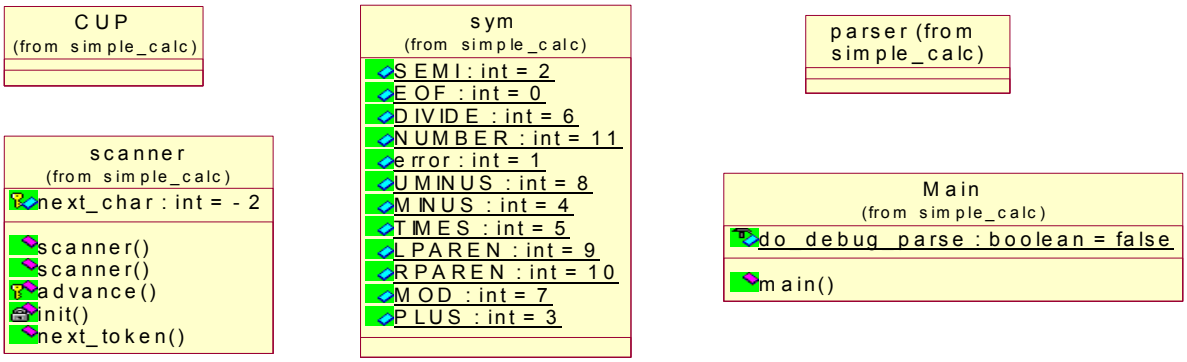




Bemerkung:
Aufgrund des Auftretens eines Fehlers beim Reverse Engineering kann leider keine genauere Darstellung der Klasse „GeoElementCheckingStub“ erfolgen.

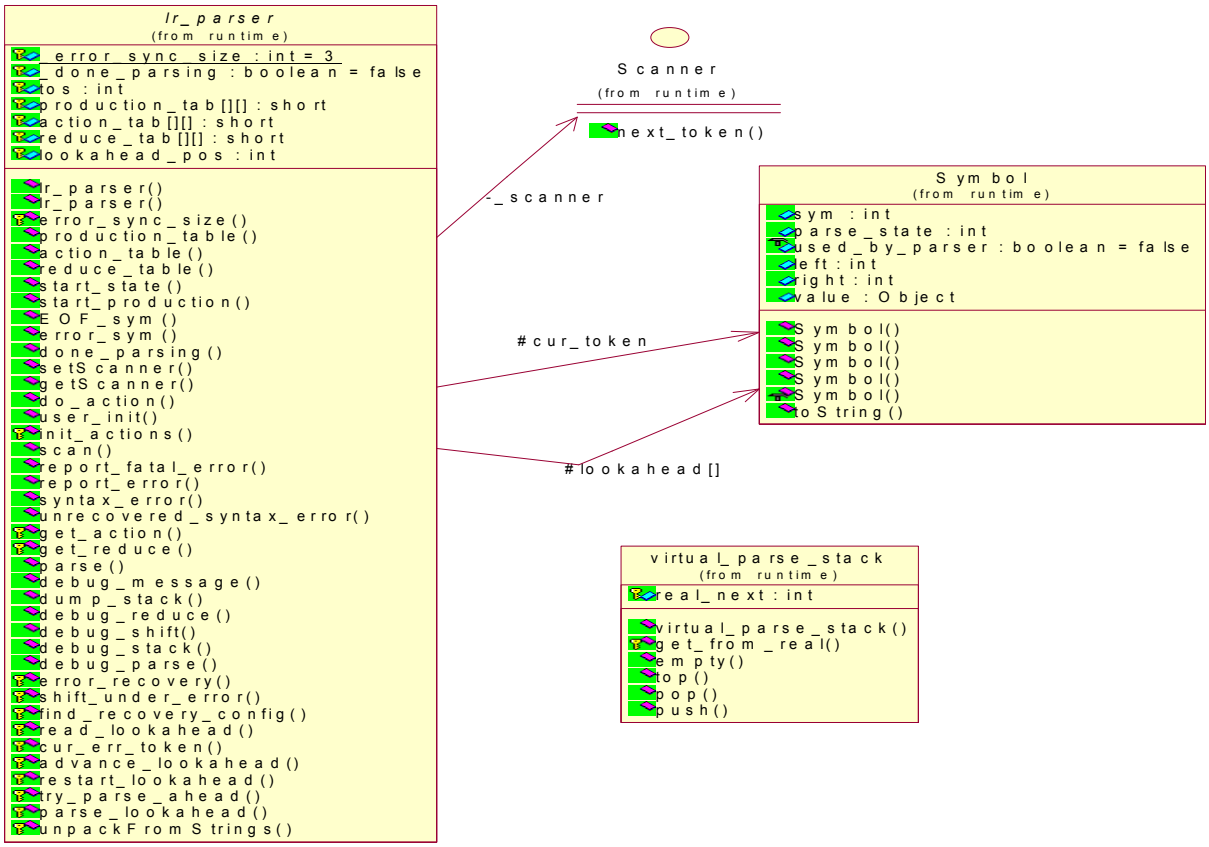
Ein weiteres Paket ist JLex. Es besitzt nur die Klasse Main, welche laut Javadoc nur die Methode *static void main(String[] arg)* besitzt.

Ein weiteres Paket trägt den Namen „simple_calc“. Es umfasst folgende Klassen:

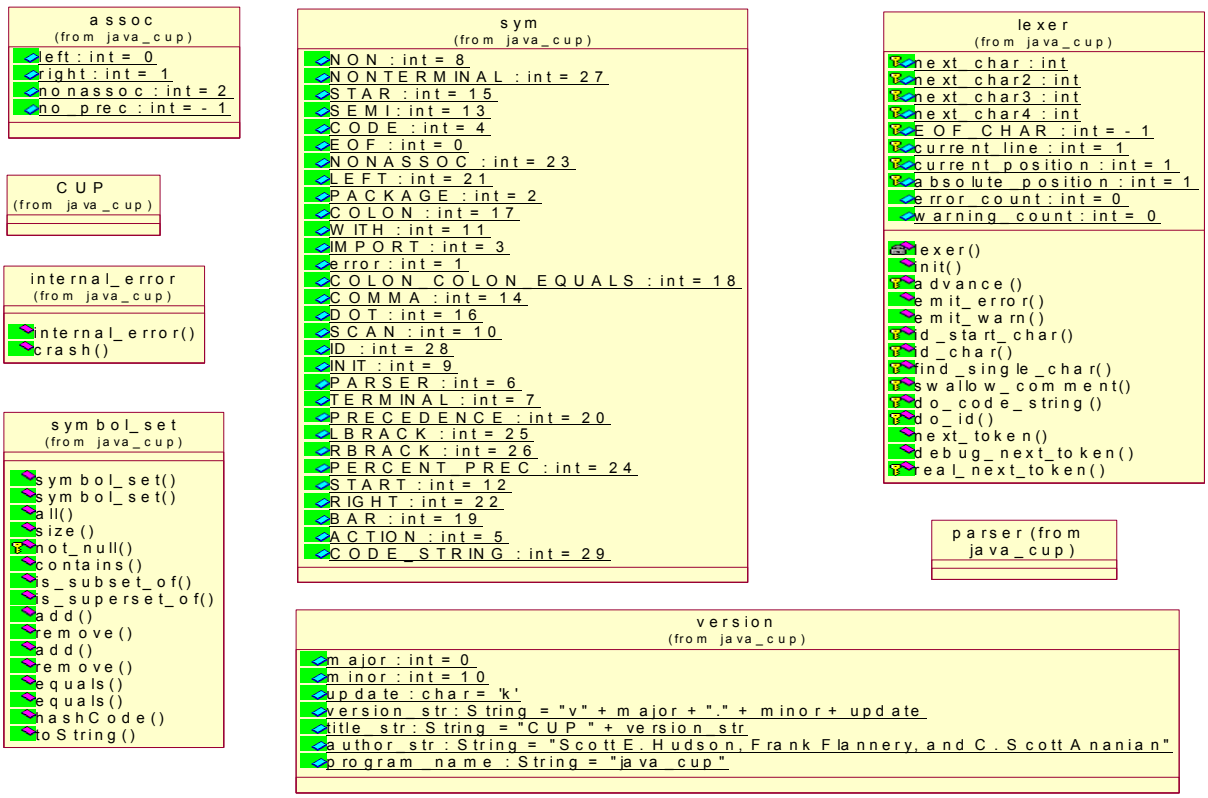


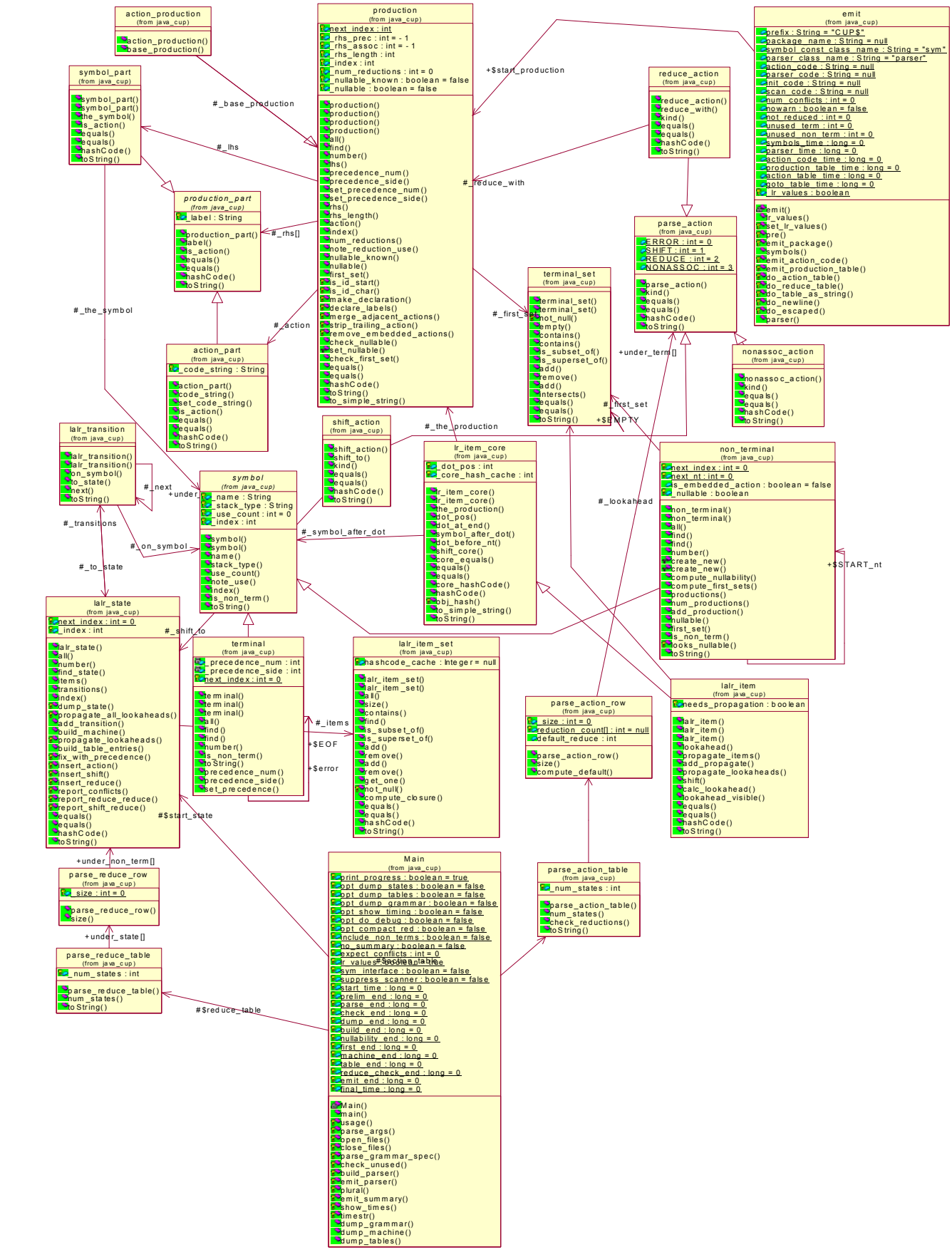
Bemerkung:
Aufgrund des Auftretens eines Fehlers beim Reverse Engineering kann leider keine genauere Darstellung der Klasse „parser“ erfolgen.

Das Paket „runtime” hat folgende Struktur:



Nun stellen wir die Klassenübersicht des Paketes „java_cup” dar:




action_production
(from java_cup)
action_production()
base_production()
production
(from java_cup)
next_index : int
rhs_prec : int = - 1
rhs_assoc : int = - 1
rhs_length : int
index : int
num_reductions : int = 0
nullable_known : boolean = false
nullable : boolean = false
production()
production()
production()
production()
all()
find()
number()
lhs()
precedence_num()
precedence_side()
set_precedence_num()
set_precedence_side()
rhs()
rhs_length()
action()
index()
num_reductions()
note_reduction_use()
nullable_known()
nullable()
first_set()
is_id_start()
is_id_char()
make_declaration()
declare_labels()
merge_adjacent_actions()
strip_trailing_action()
remove_embedded_actions()
check_nullable()
set_nullable()
check_first_set()
equals()
equals()
hashCode()
toString()
to_simple_string()
emit
(from java_cup)
prefix : String = "CUP$"
package_name : String = null
symbol_const_class_name : String = "sym"
parser_class_name : String = "parser"
action_code : String = null
parser_code : String = null
init_code : String = null
scan_code : String = null
num_conflicts : int = 0
nowarn : boolean = false
not_reduced : int = 0
unused_term : int = 0
unused_non_term : int = 0
symbols_time : long = 0
parser_time : long = 0
action_code_time : long = 0
production_table_time : long = 0
action_table_time : long = 0
goto_table_time : long = 0
lr_values : boolean
emit()
lr_values()
set_lr_values()
pre()
emit_package()
symbols()
emit_action_code()
emit_production_table()
do_action_table()
do_reduce_table()
do_table_as_string()
do_newline()
do_escaped()
parser()
symbol_part
(from java_cup)
symbol_part()
symbol_part()
the_symbol()
is_action()
equals()
equals()
hashCode()
toString()
reduce_action
(from java_cup)
reduce_action()
reduce_with()
kind()
equals()
equals()
hashCode()
toString()
production_part
(from java_cup)
label : String
production_part()
label()
is_action()
equals()
equals()
hashCode()
toString()
parse_action
(from java_cup)
ERROR : int = 0
SHIFT : int = 1
REDUCE : int = 2
NONASSOC : int = 3
parse_action()
kind()
equals()
equals()
hashCode()
toString()
terminal_set
(from java_cup)
terminal_set()
terminal_set()
not_null()
empty()
contains()
contains()
is_subset_of()
is_superset_of()
add()
remove()
add()
intersects()
equals()
equals()
toString()
nonassoc_action
(from java_cup)
nonassoc_action()
kind()
equals()
equals()
hashCode()
toString()
action_part
(from java_cup)
code_string : String
action_part()
code_string()
set_code_string()
is_action()
equals()
equals()
hashCode()
toString()
shift_action
(from java_cup)
shift_action()
shift_to()
kind()
equals()
equals()
hashCode()
toString()
lalr_transition
(from java_cup)
lalr_transition()
lalr_transition()
on_symbol()
to_state()
next()
toString()
symbol
(from java_cup)
name : String
stack_type : String
use_count : int = 0
index : int
symbol()
symbol()
name()
stack_type()
use_count()
note_use()
index()
is_non_term()
toString()
lr_item_core
(from java_cup)
dot_pos : int
core_hash_cache : int
lr_item_core()
lr_item_core()
the_production()
dot_pos()
dot_at_end()
symbol_after_dot()
dot_before_nt()
shift_core()
core_equals()
equals()
equals()
core_hashCode()
hashCode()
obj_hash()
to_simple_string()
toString()
non_terminal
(from java_cup)
next_index : int = 0
next_nt : int = 0
is_embedded_action : boolean = false
nullable : boolean
non_terminal()
non_terminal()
all()
find()
find()
number()
create_new()
create_new()
compute_nullability()
compute_first_sets()
productions()
num_productions()
add_production()
nullable()
first_set()
is_non_term()
looks_nullable()
toString()
lalr_state
(from java_cup)
next_index : int = 0
_index : int
lalr_state()
all()
number()
find_state()
items()
transitions()
index()
dump_state()
propagate_all_lookaheads()
add_transition()
build_machine()
propagate_lookaheads()
build_table_entries()
fix_with_precedence()
insert_action()
insert_shift()
insert_reduce()
report_conflicts()
report_reduce_reduce()
report_shift_reduce()
equals()
equals()
hashCode()
toString()
terminal
(from java_cup)
_precedence_num : int
_precedence_side : int
next_index : int = 0
terminal()
terminal()
terminal()
all()
find()
find()
number()
is_non_term()
toString()
precedence_num()
precedence_side()
set_precedence()
lalr_item_set
(from java_cup)
hashcode_cache : Integer = null
lalr_item_set()
lalr_item_set()
all()
size()
contains()
find()
is_subset_of()
is_superset_of()
add()
remove()
add()
remove()
get_one()
not_null()
compute_closure()
equals()
equals()
hashCode()
toString()
parse_action_row
(from java_cup)
_size : int = 0
reduction_count[] : int = null
default_reduce : int
parse_action_row()
size()
compute_default()
lalr_item
(from java_cup)
needs_propagation : boolean
lalr_item()
lalr_item()
lalr_item()
lookahead()
propagate_items()
add_propagate()
propagate_lookaheads()
shift()
calc_lookahead()
lookahead_visible()
equals()
equals()
hashCode()
toString()
Main
(from java_cup)
print_progress : boolean = true
opt_dump_states : boolean = false
opt_dump_tables : boolean = false
opt_dump_grammar : boolean = false
opt_show_timing : boolean = false
opt_do_debug : boolean = false
opt_compact_red : boolean = false
include_non_terms : boolean = false
no_summary : boolean = false
expect_conflits : int = 0
lr_values : boolean = true
sym_interface : boolean = false
suppress_scanner : boolean = false
start_time : long = 0
prelim_end : long = 0
parse_end : long = 0
check_end : long = 0
dump_end : long = 0
build_end : long = 0
nullability_end : long = 0
first_end : long = 0
machine_end : long = 0
table_end : long = 0
reduce_check_end : long = 0
emit_end : long = 0
final_time : long = 0
Main()
main()
usage()
parse_args()
open_files()
close_files()
parse_grammar_spec()
check_unused()
build_parser()
emit_parser()
plural()
emit_summary()
show_times()
timestr()
dump_grammar()
dump_machine()
dump_tables()
parse_action_table
(from java_cup)
_num_states : int
parse_action_table()
num_states()
check_reductions()
toString()
parse_reduce_row
(from java_cup)
_size : int = 0
parse_reduce_row()
size()
parse_reduce_table
(from java_cup)
_num_states : int
parse_reduce_table()
num_states()
toString()
#_base_production
#_lhs
#_rhs[]
#_the_symbol
#_action
+$start_production
#_reduce_with
#_first_set
+under_term[]
#_first_set
+$EMPTY
#_the_production
#_lookahead
+$START_nt
#_next
+under_
#_transitions
#_on_symbol
#_to_state
#_symbol_after_dot
#_shift_to
#_items
+$EOF
+$error
#$start_state
+under_non_term[]
+under_state[]
#$reduce_table
#$action_table