API Documentation

Table of Contents

Frontend

Parser API

Overview: The Parser API describes methods available to the parser of the Simple Program Analyser (SPA). The main method, parseSimple, is the main entry point to the program, where a SIMPLE program string is input for analysis.

- [parseSimple]
- [parseExpression]

```
VOID parseSimple(STRING rawProgram) throws SYNTAX_ERROR;
```

Description: Takes in a SIMPLE program string so the PKB can be populated with entries. If the program string is not in valid SIMPLE syntax, or contains semantic errors, an ERROR will be

thrown.

```
EXPRESSION parseExpression(STRING_LIST lexedExpression);
```

Description: Takes in a lexed SIMPLE expression, and returns an Abstract Syntax Tree (AST) node that represents the expression. This method may be used to pattern-match queries.

Design Extractor API

Overview: The Design Extractor API describes methods available to the Design Extractor of the Simple Program Analyser (SPA). In the Design Extractor, program design entity relationships are identified and stored in the PKB. The main method, extractDesign, provides the inputs required by the Design Extractor to determine program design entity relationships, namely an Abstract Syntax Tree (AST) of a SIMPLE program.

- [extractDesign]

```
VOID extractDesign(PROGRAM_NODE rootNode) throws SEMANTIC_ERROR;
```

Description: Takes in a SIMPLE AST and walks the tree, identifying the presence of important relationships between program design entities. If the program contains semantic errors, a SEMANTIC_ERROR will be thrown.
Normal behaviour: The AST represents a semantically valid SIMPLE program, and the Design Extractor stores program design entity relationships in the PKB for queries.
Abnormal behaviour: If there is a semantic error in the SIMPLE program represented by the AST, a SEMANTIC_ERROR will be thrown. The design extractor will immediately cease operations, discarding the rest of the program that has not been analysed yet.

PKB

Abstract Syntax Tree (AST) API

Overview: The AST API describes the methods available to construct an Abstract Syntax Tree in the Simple Program Analyser (SPA).

- [createAssignNode]
- [createCallNode]
- [createIfNode]
- [createPrintNode]
- [createProcedureNode]
- [createProgramNode]
- [createReadNode]

- [createStmtlstNode]
- [createWhileNode]

```
ASSIGNMENT_STATEMENT_NODE createAssignNode(STATEMENT_NUMBER sn, VARIABLE var, EXPRESSION expr);
```

Description: Creates and returns an **ASSIGNMENT_STATEMENT_NODE** with `var` and `expr` as the children, and `sn` as its statement number.

```
CALL_STATEMENT_NODE createCallNode(STATEMENT_NUMBER sn, NAME procName);
```

Description: Creates and returns a **CALL_STATEMENT_NODE** with `procName` as the child, and `sn` as its statement number.

```
IF_STATEMENT_NODE createIfNode(STATEMENT_NUMBER sn, CONDITIONAL_EXPRESSION predicate,
STMTLST_NODE leftStatementList, STMTLST_NODE rightStatementList);
```

Description: Creates and returns an **IF_STATEMENT_NODE** with the condition `predicate`, `leftStatementList` and `rightStatementList` as the children, and `sn` as its statement number.

```
PRINT_STATEMENT_NODE createPrintNode(STATEMENT_NUMBER sn, VARIABLE var);
```

Description: Creates and returns a **PRINT_STATEMENT_NODE** with `var` as the child, and `sn` as its statement number.

```
PROCEDURE_NODE createProcedureNode(NAME procedureName, STMTLST_NODE stmtlstNode);
```

Description: Creates and returns a **PROCEDURE_NODE** with `stmtlstNode` as the child, and `procedureName` as the name of the procedure.

```
PROGRAM_NODE createProgramNode(NAME programName, PROCEDURE_NODE_LIST procedureNodes);
```

Description: Creates and returns a **PROGRAM_NODE** with `procedureNodes` as the child in a **PROCEDURE_NODE_LIST** form, and `programName` as the name of the program.

```
READ_STATEMENT_NODE createReadNode(STATEMENT_NUMBER sn, VARIABLE var);
```

Description: Creates and returns a **READ_STATEMENT_NODE** with `var` as the child, and `sn` as its statement number.

```
STMTLST_NODE createStmtlstNode(STATEMENT_NODE_LIST statementNodes);
```

Description: Creates and returns a **STMTLST_NODE** with `statementNodes` as its children;

```
WHILE_STATEMENT_NODE createWhileNode(STATEMENT_NUMBER sn, CONDITIONAL_EXPRESSION predicate, STMTLST_NODE statementList);
```

Description: Creates and returns an **WHILE_STATEMENT_NODE** with the condition `predicate`, `statementList` as its children, and `sn` as its statement number.

Expressions API

Overview: The Expressions API describes the methods available to create Expression representations in the Simple Program Analyser (SPA).

- [createAndExpr]
- [createDivExpr]
- [createEqExpr]
- [createGtExpr]
- [createGteExpr]
- [createLtExpr]
- [createLteExpr]
- [createMinusExpr]
- [createModExpr]
- [createNotExpr]
- [createOrExpr]
- [createPlusExpr]
- [createRefExpr]
- [createTimesExpr]

```
AND_EXPRESSION createAndExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);
```

Description: Creates and returns an **AND_EXPRESSION** where the truthy value depends on both `leftExpr` and the `rightExpr`.

```
ARITHMETIC_EXPRESSION createDivExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);
```

Description: Creates and returns an **ARITHMETIC_EXPRESSION** where the `leftExpr` is divided by the `rightExpr`.

```
RELATIONAL_EXPRESSION createGtExpr(EXPRESSION leftRelFactor, EXPRESSION rightRelFactor);
```

Description: Creates and returns a **RELATIONAL_EXPRESSION** where the `leftRelFactor` is equal

to the `rightRelFactor`.

`RELATIONAL_EXPRESSION createGtExpr(EXPRESSION leftRelFactor, EXPRESSION rightRelFactor);`

Description: Creates and returns a **RELATIONAL_EXPRESSION** where the `leftRelFactor` is greater than the `rightRelFactor`.

`RELATIONAL_EXPRESSION createGteExpr(EXPRESSION leftRelFactor, EXPRESSION rightRelFactor);`

Description: Creates and returns a **RELATIONAL_EXPRESSION** where the `leftRelFactor` is greater than or equals to the `rightRelFactor`.

`RELATIONAL_EXPRESSION createLtExpr(EXPRESSION leftRelFactor, EXPRESSION rightRelFactor);`

Description: Creates and returns a **RELATIONAL_EXPRESSION** where the `leftRelFactor` is lesser than the `rightRelFactor`.

`RELATIONAL_EXPRESSION` **createLteExpr(EXPRESSION leftRelFactor, EXPRESSION rightRelFactor);**

Description: Creates and returns a **RELATIONAL_EXPRESSION** where the `leftRelFactor` is lesser than or equals to the `rightRelFactor`.

`ARITHMETIC_EXPRESSION createMinusExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);`

Description: Creates and returns an **ARITHMETIC_EXPRESSION** where the `leftExpr` is divided by the `rightExpr`.

`ARITHMETIC_EXPRESSION createModExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);`

Description: Creates and returns an **ARITHMETIC_EXPRESSION** where the `leftExpr` is mod by the `rightExpr`.

`NOT_EXPRESSION createNotExpr(CONDITIONAL_EXPRESSION expr);`

Description: Creates and returns an **NOT_EXPRESSION** with the negated value of `expr`.

`OR_EXPRESSION createOrExpr(CONDITIONAL_EXPRESSION leftExpr, CONDITIONAL_EXPRESSION rightExpr);`

Description: Creates and returns an **OR_EXPRESSION** where the truthy value depends on either `leftExpr` or the `rightExpr`.

```
ARITHMETIC_EXPRESSION createPlusExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);
```

Description: Creates and returns an **ARITHMETIC_EXPRESSION** where the `leftExpr` is added to the `rightExpr`.

```
REFERENCE_EXPRESSION createRefExpr(BASIC_DATA_TYPE basicData);
```

Description: Creates and returns a **REFERENCE_EXPRESSION** based on `basicData`.

```
ARITHMETIC_EXPRESSION createTimesExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);
```

Description: Creates and returns an **ARITHMETIC_EXPRESSION** where the `leftExpr` is multiplied with the `rightExpr`.

Statements Table API

Overview: The Statements Table API describes the methods available to extract information related to statements.

- [getAllStatements]
- [getStatementFromIndex]
- [getStatementsForConstants]
- [getStatementsPatternMatching]
- [insertIntoStatementTable]

```
STATEMENT_LIST getAllStatements(DESIGN_ENT_STMT_NAME stmtType);
```

Description: Returns a **STATEMENT_LIST** of all the statements in the Statements Table.

```
STATEMENT getStatementFromIndex(INTEGER index);
```

Description: Returns the **STATEMENT** with the corresponding `index`.

```
STATEMENT_LIST getStatementsForConstant(INTEGER constant);
```

Description: Returns a **STATEMENT_LIST** with all the statements that contains `constant`.

```
STATEMENT_LIST getStatementsPatternMatching(NODE astNode, BOOLEAN allowBefore, BOOLEAN allowAfter, DESIGN_ENT_STMT_NAME stmtType);
```

Description: // TODO

```
VOID insertIntoStatementTable(STATEMENT statement, INTEGER lineNumber);
```

Description: Inserts a **STATEMENT** `statement` with is corresponding `lineNumber` into the Statements Table.

VarTable API

Overview: The VarTable API describes the methods available to extract information related to variables in the processed SIMPLE program.

- [getAllVariables]
- [getIndexFromVariable]
- [getVariableIndex]
- [insertIntoVariableTable]

```
VARIABLE_LIST getAllVariables();
```

Description: Returns a **VARIABLE_LIST** of all variables stored in the VarTable.

```
INTEGER getIndexFromVariable(VARIABLE var);
```

Description: Returns the **INTEGER** key of `var` in the VarTable.

```
VARIABLE getVariableIndex(INTEGER index);
```

Description: Returns the **VARIABLE** with `index` as its key in the VarTable. If no there is no such `index`, the function throws an **INVALID_INDEX_ERROR**.

```
INTEGER insertIntoVariableTable(VARIABLE var);
```

Description: Inserts the **VARIABLE** `var` into VarTable. Returns the index that `var` is stored at in the VarTable.

ProcTable API

Overview: The ProcTable API describes the methods available to extract information related to procedures in the processed SIMPLE program.

- [getAllProcedures]
- [getProcedureIndex]
- [getProcedureFromIndex]
- [insertIntoProcedureTable]

```
PROCEDURE_LIST getAllProcedures();
```

Description: Returns a **PROCEDURE_LIST** of all procedures stored in the ProcTable.

```
INTEGER getProcedureIndex(PROCEDURE proc);
```

Description: Returns the **INTEGER** key of `proc` in the ProcTable.

```
PROCEDURE getProcedureFromIndex(INTEGER index);
```

Description: Returns the **PROCEDURE** with `index` as its key in the ProcTable. If no there is no such `index`, the function throws an **INVALID_INDEX_ERROR**.

```
INTEGER insertIntoVariableTable(VARIABLE var);
```

Description: Inserts the **VARIABLE** `var` into VarTable. Returns the index that `var` is stored at in the VarTable.

Uses API

Overview: The Uses API describes the methods available to extract information related to the Uses relationships in the processed SIMPLE program.

- [addUsesRelationships]
- [checkIfProcedureUses]
- [checkIfStatementUses]
- [getAllUsesProcedures]
- [getAllUsesStatements]
- [getAllUsesVariables]
- [getUsesProcedures]
- [getUsesStatements]
- [getUsesVariablesFromStatement]
- [getUsesVariablesFromProcedure]

```
VOID addUsesRelationships(STATEMENT/PROCEDURE stmt, VARIABLE_LIST varList);
```

Description: Adds all Uses relationships in `stmt` // TODO don't understand what this function is suppose to do

---

```
(OPTIONAL)PROCEDURE checkIfProcedureUses(STRING proc, STRING var);
```

Description: Returns **PROCEDURE** if `proc` uses `var`, or an empty **OPTIONAL** if it does not.

```
(OPTIONAL)STATEMENT checkIfStatementUses(INTEGER stmt, STRING var);
```

Description: Returns `STATEMENT` if `stmt` uses `var`, or an empty **OPTIONAL** if it does not.

```
STRING_LIST getAllUsesProcedures();
```

Description: // TODO is STRING_LIST = PROCECURE_LIST?

```
INTEGER_LIST getAllUsesStatements(STATEMENT_TYPE stmtType);
```

Description: // TODO is INTEGER_LIST = STMT_LIST?

```
VARIABLE_LIST getAllUsesVariables();
```

Description: Returns a **VARIABLE_LIST** of all variables that are used in the SIMPLE program.

```
PROCEDURE_LIST getUsesProcedures(VARIABLE var);
```

Description: // TODO

```
INTEGER_LIST getUsesStatements(VARIABLE var, STATEMENT_TYPE stmtType);
```

Description: // TODO

```
VARIABLE_LIST getUsesVariablesFromStatement(INTEGER stmt);
```

Description: // TODO should it be INTEGER or STATEMENT?

```
VARIABLE_LIST getUsesVariablesFromProcedure(PROCEDURE proc);
```

Description: Returns a **VARIABLE_LIST** of variables that were used in `proc`.

Modifies API

Overview: The Modifies API describes the methods exposed by Modifies Table to insert and extract information related to the Modifies relationships in the processed SIMPLE program.

- [addModifiesRelationships]
- [checkIfProcedureModifies]

- [checkIfStatementModifies]
- [getAllModifiesProcedures]
- [getAllModifiesStatements]
- [getAllModifiesVariables]
- [getModifiesProcedures]
- [getModifiesStatements]
- [getModifiesVariablesForProcedure]
- [getVariablesModifiedByStatement]

`VOID addModifiesRelationships(INTEGER stmt, VARIABLE_LIST var);`

Description: Add all variables in VARIABLES_LIST `var` that are modified in `stmt` to the Modifies Table.

`(OPTIONAL)PROCEDURE checkIfProcedureModifies(STRING proc, STRING var);`

Description: Returns the PROCEDURE if `proc` modifies `var`, else return nothing.

`(OPTIONAL)STATEMENT checkIfStatementModifies(INTEGER stmt, STRING var);`

Description: Returns the STATEMENT if `stmt` modifies `var`, else return nothing.

`PROCEDURE_LIST getAllModifiesProcedures();`

Description: Returns a PROCEDURE_LIST of all PROCEDURE that modifies.

`STATEMENT_LIST getAllModifiesStatements(STATEMENT_TYPE stmtType);`

Description: Returns a STATEMENT_LIST of all STATEMENT that modifies.

`VARIABLE_LIST getAllModifiesVariables(STATEMENT_TYPE stmtType);`

Description: Returns a VARIABLE_LIST of all VARIABLE that are modified by STATEMENT of STATEMENT_TYPE.

`PROCEDURE_LIST getModifiesProcedures(VARIABLE var);`

Description: Returns a PROCEDURE_LIST of all PROCEDURE that modifies VARIABLE `var`.

`STATEMENT_LIST getModifiesStatements(VARIABLE var, STATEMENT_TYPE stmtType) ;`

Description: Returns a STATEMENT_LIST of all STATEMENT of STATEMENT_TYPE, that modifies VARIABLE `var`.

`VARIABLE_LIST getModifiesVariablesForProcedure(PROCEDURE proc);`

Description: Returns a VARIABLE_LIST of all VARIABLE that are modified by PROCEDURE `proc`.

`VARIABLE_LIST getVariablesModifiedByStatement(INTEGER stmt);`

Description: Returns a VARIABLE_LIST of all VARIABLE that are modified by `stmt`.

Parent API

Overview: The Parent API describes the methods exposed by Parent Table to insert and extract information related to the Parent relationships in the processed SIMPLE program.

- [addParentRelationships]
- [addParentRelationshipsStar]
- [checkIfParentHolds]
- [getAllChildStatements]
- [getAllChildStatementsStar]
- [getAllParentStatements]
- [getAllParentStatementsStar]
- [getAllParentStatementsTyped]
- [getAllParentStatementsTypedStar]
- [getChildStatement]
- [getParentStatement]

`VOID addParentRelationships(INTEGER parent, INTEGER child);`

Description: Adds a Parent relationship between `parent` and `child` into the Parent Table.

`VOID addParentRelationshipsStar(INTEGER parent, INTEGER_LIST children);`

Description: // TODO

`(OPTIONAL)STATEMENT checkIfParentHolds(INTEGER parent, INTEGER child);`

Description: Returns the STATEMENT if there is a Parent relationship between `parent` and `child`, else return empty.

`STATEMENT_LIST getAllChildStatements(INTEGER parent, STATEMENT_TYPE stmtType);`

Description: Returns a STATEMENT_LIST of all child STATEMENT of `parent`. Child STATEMENT are of STATEMENT_TYPE `stmtType`.

`STATEMENT_LIST getAllChildStatementsStar(INTEGER parent, STATEMENT_TYPE stmtType);`

Description: Returns a STATEMENT_LIST of all transitive child STATEMENT of `parent`. Child STATEMENT are of STATEMENT_TYPE `stmtType`.

`STATEMENT_LIST getAllParentStatements(INTEGER child, STATEMENT_TYPE stmtType);`

Description: // TODO return STATEMENT_LIST or STATEMENT? '''

`STATEMENT_LIST getAllParentStatements(INTEGER child, STATEMENT_TYPE stmtType);`

Description: Returns a STATEMENT_LIST of all transitive Parent of `child`. Parents are of STATEMENT_TYPE `stmtType`.

`STATEMENT_LIST getAllParentStatementsTyped(STATEMENT_TYPE stmtTypeOfParent, STATEMENT_TYPE stmtTypeOfChild);`

Description: Returns a STATEMENT_LIST of all Parents that are of STATEMENT_TYPE `stmtTypeOfParent`, with a child of STATEMENT_TYPE `stmtTypeOfChild`.

`STATEMENT_LIST getAllParentStatementsTypedStar(STATEMENT_TYPE stmtTypeOfParent, STATEMENT_TYPE stmtTypeOfChild);`

Description: Returns a STATEMENT_LIST of all Parents that are of STATEMENT_TYPE `stmtTypeOfParent`, with a transitive child of STATEMENT_TYPE `stmtTypeOfChild`.

`STATEMENT_LIST getChildStatement(INTEGER parent);`

Description: Returns a STATEMENT_LIST of all child STATEMENT of `parent`.

`STATEMENT getParentStatement(INTEGER child);`

Description: Returns the Parent STATEMENT of `child`.

Follows API

Overview: The Follows API describes the methods exposed by Follows Table to insert and extract information related to the Follows relationships in the processed SIMPLE program.

`VOID addFollowsRelationships(INTEGER before, INTEGER after);`

Description: Adds a Follows relationship between `before` and `after` into the Parent Table.

`VOID addFollowsRelationshipsStar(INTEGER before, INTEGER_LIST after);`

Description: Adds all STATEMENT that Follows after `before`, as an INTEGER_LIST.

`BOOLEAN checkIfFollowsHolds(INTEGER beforeStatement, INTEGER afterStatement);`

Description: Returns true if `afterStatement` Follows after `beforeStatement`.

`STATEMENT_LIST getAllFollowsStatements(STATEMENT_TYPE stmtTypeOfBefore, STATEMENT_TYPE stmtTypeOfAfter);`

Description: Returns a STATEMENT_LIST of all STATEMENT with STATEMENT_TYPE `stmtTypeOfAfter`, that Follows after STATEMENT of STATEMENT_TYPE `stmtTypeOfBefore`.

`STATEMENT_LIST getAllFollowsStatementsStar(STATEMENT_TYPE stmtTypeOfBefore, STATEMENT_TYPE stmtTypeOfAfter);`

Description: Returns a STATEMENT_LIST of all STATEMENT with STATEMENT_TYPE `stmtTypeOfAfter`, that transitively Follows after STATEMENT of STATEMENT_TYPE `stmtTypeOfBefore`.

`STATEMENT_LIST getAllStatementsAfterStar(INTEGER statement, STATEMENT_TYPE stmtType);`

Description: Returns a STATEMENT_LIST of all STATEMENT of STATEMENT_TYPE `stmtType`, that transitively Follows after `statement`.

```
STATEMENT_LIST getAllStatementsBeforeStar(INTEGER statement, STATEMENT_TYPE stmtType));
```

Description: Returns a STATEMENT_LIST of all STATEMENT of STATEMENT_TYPE `stmtType`, that transitively Follows before `statement`.

```
(OPTIONAL)STATEMENT getStatementAfter(INTEGER statement, STATEMENT_TYPE stmtType);
```

Description: Returns the STATEMENT of STATEMENT_TYPE `stmtType` that Follows after `statement`, if any.

```
(OPTIONAL)STATEMENT getStatementBefore(INTEGER statement, STATEMENT_TYPE stmtType);
```

Description: Returns the STATEMENT of STATEMENT_TYPE `stmtType` that Follows before `statement`, if any.

PQL

PQLManager API

Overview: Handles the business logic for processing and evaluating Processed Query Language (PQL) queries.

- [execute]

```
FORMATTED_QUERY_RESULT execute(STRING query);
```

Description: Takes in a query in Processed Query Language (PQL) form, and returns a formatted result of the `query`.

PreProcessor API

Overview: Handles the business logic for pre-processing PQL queries, including validating syntax and semantics.

- [process]

```
ABSTRACT_QUERY processQuery(STRING query);
```

Description: Returns an ABSTRACT_QUERY after validating and breaking down the `query`.

Evaluator API

Overview: Evaluates the processed query and obtain a result for it by interacting with the PKB.

- [evaluate]

```
RAW_QUERY_RESULT evaluate(STRING query);
```

Description: Returns a RAW_QUERY_RESULT after evaluating the `query` and obtaining information from the PKB.

Projector API

Overview: Formats query results into human readable context.

- [formatResult]

```
FORMATTED_RESULT formatResult(RAW_QUERY_RESULT rawQueryResult);
```

Description: Returns a FORMATTED_RESULT after formatting `rawQueryResult` to a conforming standard.

Shared Libraries (for internal use)

Lexer API

- [splitByDelimiter]
- [splitByWhitespace]
- [splitProgram]

```
STRING_LIST splitByDelimiter(STRING str, STRING delimiter);
```

Description: Returns a STRING_LIST of tokens after splitting `str` by the `delimiter`.

```
STRING_LIST splitByWhitespace(STRING stringContext);
```

Description: Returns a STRING_LIST of tokens after splitting `stringContext` by whitespaces.

```
STRING_LIST splitProgram(STRING simpleProgram);
```

Description: Returns a STRING_LIST after splitting SIMPLE program `simpleProgram` into strings containing the names, numbers, symbols that the Lexer can determine based on SIMPLE syntax. All

---

whitespace will be truncated from the strings.