Template for JSON representation of ASTFRI nodes

Expressions

LiteralExpressions:

{"node" : "*LiteralExpr", **!!* must be replaced by concrete type like: Int,Bool,String,..**

"value" : e.g. (500, "text",true,...)

}

UnknownExpression:

{"node": "UnknownExpr"}

ThisExpression:

{"node" : "ThisExpr"}

LambdaExpression:

{"node" :"LambdaExpr",

"parameters" : [{ParamDefStmt},...],

"body" : {SomeStatement}

}

LambdaCallExpression:

{

"node" : LambdaCallExpr

"lambda" : {SomeExpression},

"arguments" : [{SomeExpression},.....]

}

FunctionCallExpression:

{

"node" : "FunctionCallExpr",

"name" : "SomeName",

" arguments": [{SomeExpression},...]

}

MethodCallExpression:

{

”node” : ”MethodCallExpr”

”owner” : {SomeRefExpr},**!! or null,if not resolved**

”name” : ”SomeName”,

”arguments” : [{SomeExpression},...]

}

Referencies:

{

”node” : ” TypeOfReference”(

ParamVarRefExpr|LocalVarRefExpr |ThisExpr|ClassRefExpr|GlobalVarRefExpr),

”name”: ”SomeName”

}

MemberVarRefExpr

{

”node” : ”MemberVarRefExpr”,

”name” : ”SomeName”,

”owner” : {SomeExpression}

}

BinOpExpression:

{

”node” : ”BinOpExpr”,

”left” : {SomeExpression},

”right”: {SomeExpression},

”operator” : e.g. ” / , &&, +”

}

UnaryOpExpression:

{

”node”: ”UnaryOpExpr”,

”operator” : ”operator”;

”isPostfix” : true/false,

”argument” : {SomeExpression}

}

IfExpression -ternary operator:

{

”node” : ”IfExpr”,

”condition” : {SomeExpression},

”ifTrue” : {SomeExpression},

”ifFalse”:{SomeExpression}

}

ConstructorCallExpression:

{

”node” : ”ConstructorCallExpr”,

”type” : {SomeType},

”arguments” : [{SomeExpression},...]

}

NewExpression:

{

”node” : ”NewExpr”,

”init” : {ConstructorCallExpr}

}

DeleteExpr:

{

”node” : ”DeleteExpr”,

”expression” : {SomeExpression}

}

Statements:

MemberVarDefStmt :

{

”node” : ”MemberVarDefStmt”

”access” : ”private/public/protected/internal”,

”name” : ”SomeName”,

”type”: {SomeType},

”initializer” : {SomeExpresion} **!! or null if there is no initializer**

}

GlobalVarDefStmt | LocalVarDefStmt | ParamVarDefStmt

{

”node” : GlobalVarDefStmt | LocalVarDefStmt | ParamVarDefStmt,

”name” : ”SomeName”,

”type”: {SomeType},

”initializer” : {SomeExpresion}, **!! or null if there is no initilazer**

}

ReturnStmt

{

”node” : ”ReturnStmt”,

”value” : {SomeExpression} **!! or null if there is no return value**

}

IfStmt

{

”node” : ”IfStmt”,

”condition” : {SomeExpression},

”ifTrue” : {SomeStatement},

”ifFalse”:{SomeStatement} **!! or null**

}

CaseStmt

{

”node” : ”CaseStmt”,

”expressions”: [{SomeExpression},…],

”body” : {SomeStatement}

}

DefaultCaseStmt:

{

”node” : ”DefaultCaseStmt”,

”body” : {SomeStatement}

}

SwitchStmt

{

”node”:“SwitchStmt”,

”entry” : {SomeExpression},

”cases”: [{BaseCaseStmt},…] **->contains CaseStmt‘s and can contain DefaultCaseStmt also**

}

WhileStmt

{

”node” : ”WhileStmt”,

”condition” : {SomeExpression},

”body” : {SomeStmt}

}

DoWhileStmt

{

”node” : ”DoWhileStmt”,

”condition” : {SomeExpression},

”body” : {SomeStmt}

}

ForStmt

{

”node” : ”ForStmt”,

”init” : {SomeStatement} **!! or null if this part is empty**

”condition” : {SomeExpression} **!! or null if this part is empty**

”step” : {SomeStatement} **!! or null if this part is empty**

”body” : {SomeStmt}

}

ThrowStmt

{

”node”: ”ThrowStmt”,

”expression” : {SomeExpression}

}

UnknownStmt

{

”node” : ”UnknownStmt”

}

FunctionDefStmt

{

”node” : ”FunctionDefStmt”,

”name” : ”SomeName”,

”parameters” : [{ParamVarDefStmt},...],

”return_type” : {SometType},

”body” : {CompoundStmt}

{

MethodDefStmt

{

”node” : ”MethodDefStmt”,

”owner” : ”NameOfTheOwner”,

”name” : ”SomeName”,

”access” : ”private/public/protected/internal”,

”parameters” : [{ParamVarDefStmt},...];

”return_type” : {SomeType},

”body” : {CompoundStmt} **!! or null**

”virtual” : ”yes/no”

}

GenericParam

{

”node” : ”GenericParam”,

”name” : ”SomeName”,

”constraint” : ”SomeConstraint” **! or null**

}

ClassDefStmt

{

”node” : ”ClassDefStmt”

”name” : ”SomeName”,

”attributes” : [{MemberVarDefStmt},..],

”constructors” : [{ConstructorDefStmt},...],

”destructors” : [{DestructorDefStmt},...],

”methods” : [{MethodDefStmt},...],

”generic_parameters” : [{GenericParam},..],

”interfaces” : [”SomeName”,...],

”bases” : [”SomeName”,...],

}

InterfaceDefStmt

{

”node” : ”InterfaceDefStmt”,

”name” : ”SomeName”,

”methods” : [{MethodDefStmt},...],

”generic_parameters” : [{GenericParam},...],

”bases” : [”SomeName”,....]

}

TranslationUnit

{

”node” : ”TranslationUnit”,

”classes” : [{ClassDefStmt},..],

”functions” : [{FunctionDefStmt},...],

”globals” : [{GlobalVarDefStmt},...]

}

ConstructorDefStmt:

{

”node” : ”ConstructorDefStmt”,

”owner” : ”NameOfTheOwner”,

”parameters” : [{ParamVarDefStmt},...],

”base_initializers” : [{BaseInitialiserStmt},..],

”body” : {CompoundStmt},

”access” : ”private/public/protected/internal”

}

BaseInitializerStmt:

{

”node” : ”BaseInitializerStmt”,

”base” : ”SomeName”,

”arguments” : [{SomeExpression},...]

}

DestructorDefStmt:

{

”node” : ”DestructorDefStmt”,

”owner” : ”NameOfTheOwner”,

”body” : {CompoundStmt}

}

DefStmt:

{

”node” : ”DefStmt”,

”definitions” : [{VarDefStmt},...]

}

ExpressionStmt:

{

”node” : ”ExpressionStmt”,

”expression” : {SomeExpression}

}

CompoundStmt:

{

”node” : ”CompoundStmt”,

”statements” : [{SomeStatement},..]

}

BreakStmt:

{

”node” : ”BreakStmt”

}

ContinueStmt:

{

”node” : ”ContinueStmt”

}

Types:

DynamicType:

{

”node” : ”DynamicType”

}

IntType:

{

”node” : ”IntType”

}

FloatType:

{

”node” : ” FloatType”

}

CharType:

{

”node” : ” CharType”

}

BoolType:

{

”node” : ” BoolType”

}

VoidType:

{

”node” : ” VoidType”

}

UserType:

{

”node” : ”UserType”,

”name” : ”SomeName”

}

InDirectionType:

{

”node” : ”IndirectionType”,

”indirect” : {SomeType}

}

UnknownType:

{

”node” : ”UnknownType”

}