# AQuA Result Syntax and Semantics

AQuA defines a grammar to be used when expressing the results associated with the execution of a AQuA statement. All AQuA interpreters MUST express results using the described syntax. This might require converting the results from other formats if the AQuA statement is converted to another query language prior to being run.

Recall that an AQuA SELECT statement returns a list of tuples corresponding to a return clause, while an AQuA EXISTS statement returns a Boolean value.

If the AQuA statement is a SELECT statement, the result is a JSON array of JSON objects. Within this array, each JSON object corresponds to one iteration of the AQuA query body that evaluated to TRUE. Within this JSON object, each named elements corresponds to one of the listed properties in the AQuA return clause. Specifically, the “string” of the JSON object element copies the property dereference expression that appears in the return clause. The “value” of the JSON object element is the value of that property for the specific class instance used in the evaluation that created the TRUE result.

If the AQuA statement is an EXISTs statement, the result is one of the following 3 strings: TRUE, FALSE, or UNKNOWN. The result corresponds to the logical OR of the collected results of every iteration of the statement’s query body.

## AQuA Results

grammar AQuA\_Results;

aquaResults: (aquaResultsJSON) EOF;

aquaResultsJSON: LEFT\_BRACKET (emptyResultSet | aquaResultSet | booleanType) RIGHT\_BRACKET;

emptyResultSet: LEFT\_CURLY RIGHT\_CURLY;

aquaResultSet: aquaResult ( COMMA aquaResult)\*;

aquaResult: LEFT\_CURLY aquaResultProperty ( COMMA aquaResultProperty)\* RIGHT\_CURLY;

aquaResultProperty: classVarProperty COLON (integerType | floatType | booleanType | dateTimeType | LiteralValue);

//////////////////////////////

// classes

classVarProperty: LiteralValue;

//////////////////////////////

// datatypes

integerType: (PLUS | MINUS)? (Digit | Digits ) | UNKNOWN;

floatType: ((PLUS | MINUS)? (Digits DOT Digits | DOT Digits)) | UNKNOWN;

Digits: Digit+;

Digit: [0-9];

TRUE: 'TRUE';

FALSE: 'FALSE';

UNKNOWN: 'UNKNOWN';

booleanType: (TRUE | FALSE | UNKNOWN);

dateTimeType: DateTime | UNKNOWN;

DateTime: Year HYPHEN Month HYPHEN Day DATE\_SUFFIX Hours COLON Minutes COLON Seconds TIMEZONE\_SUFFIX?;

fragment Year: [0-9] [0-9] [0-9] [0-9];

fragment Month: [0] [1-9] | [1] [0-2];

fragment Day: ([0] [1-9]) | ([1-2] [0-9]) | ([3] [0-1]);

fragment Hours: ([0-1] [0-9]) | ([2] [1-3]);

fragment Seconds: [0-5] [0-9] (DOT [0-9]+)?;

fragment Minutes: [0-5] [0-9];

//////////////////////////////

// Reserved

RIGHT\_CURLY: '}' ;

LEFT\_CURLY: '{' ;

RIGHT\_BRACKET: ']' ;

LEFT\_BRACKET: '[' ;

//////////////////////////////

// Operators

PLUS: '+' ;

MINUS: '-' ;

HYPHEN: MINUS;

//////////////////////////////

// Literals

DOT: '.';

COMMA: ',';

QUOTE: DOUBLEQUOTE;

DOUBLEQUOTE: '\"';

COLON: ':';

DATE\_SUFFIX: 'T';

TIMEZONE\_SUFFIX: 'Z';

//////////////////////////////

// Generic values

LiteralValue: QUOTE (~["])\* QUOTE;

//////////////////////////////

// Whitespace and comments

WS: [ \n\r\t]+ -> skip;

COMMENT: '/\*' .\*? '\*/' -> skip;

LINE\_COMMENT: '//' ~[\r\n]\* -> skip;