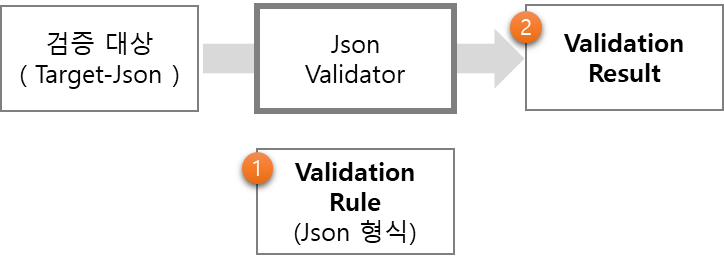
**1. [Json Validator]**

어떠한 JSON이 특정한 규칙을 지키는 지 여부를 점검하는 기능이다.



검증대상(Target-Json)이 특정한 규칙(Validation Rule)에 따르는 지 점검하는 것으로  
점검내용은 아래와 같이 구분된다.

**[ 검증 내용의 구분 ]**

|  |  |
| --- | --- |
| 구조에 대한 검증 | 검증Rule의 구조자체(Json Structure)가 Target-Json이 따라야할 구조를 결정함. |
| 값에 대한 검증 | 값이 규칙에 따르는 지 점검(SyntaxValue 절 참고) |
| 값들간 관계에 대한 검증 | 값들이 특정한 관계를 지키는 점검( SyntaxObject 절 참고) |

점검할 내용은 Validation Rule에 따라 결정되므로, 이하에서는 Validation Rule의 문법과 작성방법, 사용방법을 살펴보자.

**2. Json Validation Rule 샘플과 특징**

**[ 간단한 Validation Rule]**

[ {

"**\_?\_:**TRT\_KEY": "\_regex('1234-.\*')",

"YYK\_ID": "\_longerThan(3)",

"PAT\_ID": "\_digit",

"INSURE\_TYPE": "\_oneOf(건강보험,요향보험)",

"INSURE\_TYPE\_CD": "\_digitOr(갑을병-)",

"DEPT\_CD": "between(999, 9999)",

"DEPT\_NM": "\_oneOf(원무과,총무과) || \_regex('우리병원.\*실')",

"DR\_NM": "\_longerThan(2) && \_shorterThan(7)",

"RCPT\_GB": "\_oneOf(입원,외래,응급,퇴원,중간)",

"TRT\_YMD": "\_digitOr(-)",

"$$$$" : "${CUSTOM1} > ${CUSTOM1}",

"TRT\_S\_YMD": "\_date(${yyyy}-${mm}-${dd}) || \_digitOr(-)",

"TRT\_E\_YMD": "\_any || \_between(0, 1000)",

"DR\_CD": "\_any",

"TRT\_AMT": "lt(1000) && gt(100)",

"CUSTOM1": "\_length(8)"

} ]

**[ Validation Rule의 특징 ]**Validation Rule을 살펴보면, 다음과 같은 특징을 볼 수 있다.

1. Json 구조로 작성한다.  
   ( Rule의 형식오류를 쉽게 발견할 수 있다. )
2. 검증대상 Json의 구조를 그대로 반영하여 작성할 수 있다.  
   (룰 작성시에 불필요한 작업을 최소화 )
3. 특정한 Key값은 예약어이다. (붉은색 표시부분)
4. 필드의 값들은 Target-Json의 값들이 지켜야할 규칙이다.

**[ 참고: Validation Rule 작업 순서 ]**

\* 실무에서 검증 규칙을 제작하는 순서는 다음과 같다.  
 1. Target Json의 구조를 살펴보고  
 2. 값에 해당하는 부분을 값검증규칙(이하 SyntaxValue )으로 치환하고  
 3. 구조의 관계에 해당하는 규칙(이하 SyntaxObject, SyntaxArray) 을 추가한다.  
 4. 작업한 Rule이 정상적으로 작성되었고, 원하는 규칙이 들어가 있는 지 확인한다.  
 5. Sample Target-Json을 이용해서, 검증결과를 확인한다.

**2. Json Validation Rule**

[ { ### 1 : json-array of json-object

"**\_?\_:**TRT\_KEY": "\_regex('1234-.\*')", ### **선택필드**

"YYK\_ID": "\_longerThan(3)", ### 으로 end-line주석을 달수 있다.

"PAT\_ID": "\_digit", ### \_digit은 값검증규칙(**SyntaxValue**)

"INSURE\_TYPE": "\_oneOf(건강보험,요향보험)", ### 3

"INSURE\_TYPE\_CD": "\_digitOr(갑을병-)",

"DEPT\_CD": "between(999, 9999)",

"DEPT\_NM": "\_oneOf(원무과,총무과) || \_regex('우리병원.\*실')",

"DR\_NM": "\_longerThan(2) && \_shorterThan(7)",

"RCPT\_GB": "\_oneOf(입원,외래,응급,퇴원,중간)",

"TRT\_YMD": "\_digitOr(-)",

"**$$$$**" : "${CUSTOM1} > ${CUSTOM1}", ### 객체검증규칙(**SyntaxObject**)

"TRT\_S\_YMD": "\_date(${yyyy}-${mm}-${dd}) || \_digitOr(-)",

"TRT\_E\_YMD": "\_any || \_between(0, 1000)",

"DR\_CD": "\_any",

"TRT\_AMT": "lt(1000) && gt(100)",

"CUSTOM1": "\_length(8)"

} ]

앞서 본 Rule을 조금 상세히 살펴보자.

1. **Rule에서 Array와 Object가 등장하는 이유  
     
   Target-Json**이 지켜야하는 구조를 정의한 것이다.   
   위의 샘플은 Target-Json이 Object가 반복되는 Array 구조를 가진 Array라고 정의한 것이다.
2. Target-json에서 **반복되는 Json-Obejct**는 다음과 같은 형식과 값을 가져야 한다.
3. Target-Json 지정한 **모든 key**를 가져야 한다.
4. 단, key이름이 **\_?\_:** 로 시작하는 경우, Target-json에는 그 필드가 없어도 된다. (**선택필드**)  
   ( 선택필드가 존재하는 경우에는 선택필드도 값검증규칙에 부합해야 함)
5. 객체의 **필드의 값**들은 값검증규칙(SyntaxValue)에 부합해야 한다.  
   ( SyntaxValue는 아래 참고)
6. Json-Objec의 **필드간의 관계**는 객체검증규칙(SyntaxObject)에 부합해야 한다.  
   ( SyntaxObject는 아래 참고)

**\* Developer Note : 검증 규칙을 정상적으로 만들었는 지 확인하는 방법?**

검증 Rule은 Rule-parser를 통해, AST(Abstract Syntax Tree)로 변환되어 메모리에 loading된다.  
작성한 규칙이 정상적으로 만든 것인지, 또, 제대로 loading되었는지 확인하고 싶은 상황은 Rule작성과정에서 빈번하게 발생할 수 있다.

이러한 상황을 대비하여, 메모리에 Loading된 AST를 읽기 쉬운 스타일로 화면에 출력하는 기능이 포함되어 있다.

아래는 “showRules”함수를 호출하여, Rule AST를 출력한 것과 (호출함수명은 달라질 수 있음)   
Rule작성을 잘못한 경우에 표시되는 정보의 예이다.

wordy: ===== un-commented rule =====

[

{

"\_?\_:TRT\_KEY": "\_regex('1234-.\*')",

"YYK\_ID": "\_longerThan(3)",

"PAT\_ID": "\_digit",

"INSURE\_TYPE": "\_oneOf(건강보험,요향보험)",

"INSURE\_TYPE\_CD": "\_digitOr(갑을병-)",

"DEPT\_CD": "between(999, 9999)",

"DEPT\_NM": "\_oneOf(원무과,총무과) || \_regex('우리병원.\*실')",

"DR\_NM": "\_longerThan(2) && \_shorterThan(7)",

"RCPT\_GB": "\_oneOf(입원,외래,응급,퇴원,중간)",

"TRT\_YMD": "\_digitOr(-)",

"$$$$" : "${CUSTOM1} > ${CUSTOM1}",

"TRT\_S\_YMD": "\_date(${yyyy}-${mm}-${dd}) || \_digitOr(-)",

"TRT\_E\_YMD": "\_any || \_between(0, 1000)",

"DR\_CD": "\_any",

"TRT\_AMT": "lt(1000) && gt(100)",

"CUSTOM1": "\_length(8)"

}

]

=============================

wordy: CUSTOM1 Term( key= CUSTOM1, default= \_)

wordy: CUSTOM1 Term( key= CUSTOM1, default= \_)

**[ 정상적으로 작성한 Rule이 AST로 변환 된 것을 다시 화면에 출력한 것]**

wordy: ===== un-commented rule =====

[

{

"\_?\_:TRT\_KEY": "\_regex('1234-.\*')",

"YYK\_ID": "\_longerThan(3)",

"PAT\_ID": "\_digit",

"INSURE\_TYPE": "\_oneOf(건강보험,요향보험)",

"INSURE\_TYPE\_CD": "\_digitOr(갑을병-)",

"DEPT\_CD": "between(999, 9999)",

"DEPT\_NM": "\_oneOf(원무과,총무과) || \_regex('우리병원.\*실')",

"DR\_NM": "\_longerThan(2) && \_shorterThan(7)",

"RCPT\_GB": "\_oneOf(입원,외래,응급,퇴원,중간)",

"TRT\_YMD": "\_digitOr(-)",

"$$$$" : "${CUSTOM1} > ${CUSTOM1}",

"TRT\_S\_YMD": "\_date(${yyyy}-${mm}-${dd}) || \_digitOr(-)",

"TRT\_E\_YMD": "\_any || \_between(0, 1000)",

"DR\_CD": "\_any",

"TRT\_AMT": "lt(1000) **& &** gt(100)",

"CUSTOM1": "\_length(8)"

}

]

=============================

wordy: CUSTOM1 Term( key= CUSTOM1, default= \_)

wordy: CUSTOM1 Term( key= CUSTOM1, default= \_)

show: syntax error :: lt(1000) & & gt(100)

--------------------

lt(1000) & & gt(100)

--------------------

^^^

show: syntax error in rule

[ {

"TRT\_AMT" : "syntax error :: lt(1000) & & gt(100)"

} ]

**[ 잘못 작성했을 때 표시되는 정보 ]**  
\* 위의 붉은색과 같이 syntax error가 어디에서 발생했는지 설명하는 오류정보를 얻을 수 있다.  
( 위 사례의 경우는 && 사이에 space를 하나 넣어서 틀리게 했음. )  
어느 지점에서 실수를 했는지 쉽게 발견할 수 있도록 하기 위해서, 오류정보도 Validation-Rule과 마찬가지로 계층구조에 따라서 표시된다.

**2. Json Validation Rule Syntax**

본 절 이하에서는 Rule작성을 위한 문법을 설명한다.

참고)   
특수한 용도로 새로운 문법을 정의하는 것을 DSL (Doman Specific Language)라고 한다.  
Rule문법은 Rule DSL을 설명한다는 것과 같은 의미이다.

Target-Json이 지켜야할 구조(Array와 배열이 재귀적으로 반복되는 Tree)는 Rule-Json의 구조와 같으므로, 값들이 지켜야할 규칙(SyntaxValue)를 먼저 살펴보자.

**2.1 SyntaxValue**

Target-Json의 “**값**”들이 따라야 하는 규칙을 정의하는 방법

### **SyntaxValue 공통**

1. 규칙은 함수의 조합으로 작성한다. (사용가능한 목록은 아래 참고)
2. SyntaxValue에서 사용가능한 **함수의 이름**은 다음과 같은 규칙을 따른다.

1. Target-Json의 문자열값에 적용하는 함수는 \_ (underscore)로 시작한다.

2. Target-Json의 실수값에 적용되는 함수는 함수명이 0 으로 끝난다.

3. Target-Json의 정수값에 적용되는 함수는 함수명이 영문으로 끝난다. (뒤에 붙는게 없다.)

4. Target-Json의 문자열값을 실수로 변환해야 하는 함수는 \_ (underscore)로 시작하고 0 으로 끝난다.

1. 작성한 규칙의 실행결과는 Boolean(True/false)로 반환한다.  
    만약, 함수에서 예외를 발생하는 경우 (예를 들면, 숫자로 변환이 불가능한 문자열에 대한 숫자비교)에는 false를 반환한다.
2. 규칙은 함수와 괄호(), 조합연산( || , && 연산자)을 이용하여 반복적으로 조합할 수 있다.
3. 함수의 인자로 문자열을 써야할 경우, `(quotation) 또는 “(double-quotation)으로 감싸서 작성할 수 있다.  
   문자열 내에서 `(quotation) 또는 “(double-quotation)를 써야할 경우, \ (back-slash)를 이용하여 escape할 수 있다.  
   단, white-space가 없는 문자열은 `(quotation) 또는 “(double-quotation)없이 입력할 수 있다.

### **SyntaxValue에서 사용할 수 있는 함수목록**

|  |  |
| --- | --- |
| **함수** | **설명** |
| \_any | 항상 참 |
| \_between(100,200) | 100~200사이의 정수를 표현하는 문자열 |
| \_between0(100, 200) | 100~200사이의 실수를 표현하는 문자열 |
| \_charsIn(가, 나, 다) | 가, 나, 다로만 이루어진 문자열 |
| \_date(${yyyy}-${mm}-${dd} | 2024-02-12 와 같은 형식의 문자열 |
| \_digit | 숫자(0-9)로만 이루어진 문자열 |
| \_digitOr(a,b,c) | 숫자와 a, b, c로만 이루어진 문자열 |
| \_gt(100) | 100보다 큰 정수를 표현하는 문자열 |
| \_gt0(100) | 100보다 큰 실수를 표현하는 문자열 |
| \_gte(100) | 1000보다 크거나 같은 정수를 표현하는 문자열 |
| \_gte0(100) | 1000보다 크거나 같은 실수를 표현하는 문자열 |
| \_isString | 문자열인 경우 참. |
| \_length(10) | 길이가 10인 문자열 |
| \_longerThan(10) | 길이가 10보다 긴 문자열 |
| \_lt(999) | 999보다 작은 정수를 표현하는 문자열 |
| \_lt0(999) | 999보다 작은 실수를 표현하는 문자열 |
| \_lte(999) | 999보다 작거나 같은 정수를 표현하는 문자열 |
| \_lte0(999) | 999보다 작거나 같은 실수를 표현하는 문자열 |
| \_oneOf(this, is) | this 또는 is |
| \_regex([0-9].\*) | 정규표현식에 부합하는 문자열 (사례는 숫자로 시작하는 문자열) |
| \_shorterThan(10) | 길이가 10보다 작은 문자열 |
| between(10, 100) | 10~100 사이의 정수 |
| between0(10, 100) | 10~100 사이의 실수 |
| gt(10) | 10 보다 큰 정수 |
| gt0(10) | 10 보다 큰 실수 |
| gte(10) | 10 보다 크거나 같은 정수 |
| gte0(10) | 10 보다 크거나 같은 실수 |
| lt(10) | 10 보다 작은 정수 |
| lt0(10) | 10 보다 작은 실수 |
| lte(10) | 10 보다 작거나 같은 정수 |
| lte0(10) | 10 보다 작거나 같은 실수 |

### **SyntaxValue 사례**

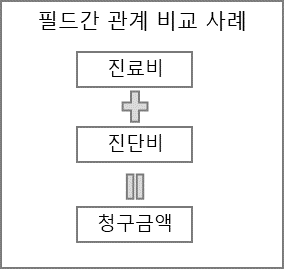
1) "\_longerThan(8) && \_shorterThan(20)"  
8~20길이의 문자열. 논리 연산자(&&, ||를 사용할 수 있음)  
\_(underscore)로 시작하는 함수들은 문자열값에 적용하는 함수.

2) "gt(100) && lt(200) || gt(200) && lt(300)"  
100200 또는 200300인 숫자값  
&&연산자가 ||연산자보다 우선순위가 높음.  
( 괄호없이 작성해도 됨. )

3) "\_gt(100) && (\_lt(200) || lt(200))"  
규칙들은 ()를 이용해서 우선순위 조정하거나, 복합 구성을 할 수 있음.  
\_gt(100)은 대상값이 문자열인데, 숫자로 바꾸어서 비교하라는 의미.

4) "\_oneOf(this, is, go'od, 'my holiday')"  
공백없는 문자열은 그대로 사용하면 되고,  
공백을 쓸 경우, '(single-quote)를 사용.  
문자열에서 '(single-quote)를 사용할 때는 (back-slash)로 escape 함.

**2.1 SyntaxObject**

객체내의 필드간 관계검증 규칙.

예를 들면,   
“진료비와 진단비의 합이 청구금액과 같아야 한다”는 식으로,  
  
Target-Json의 값들간에 어떤 관계를 검증하는 경우가 있다.

이러한 경우를 검증하는 규칙(SyntaxObject)를 살펴보자.

wordy: ===== un-commented rule =====

[

{

"\_?\_:TRT\_KEY": "\_regex('1234-.\*')",

"YYK\_ID": "\_longerThan(3)",

"PAT\_ID": "\_digit",

"INSURE\_TYPE": "\_oneOf(건강보험,요향보험)",

"INSURE\_TYPE\_CD": "\_digitOr(갑을병-)",

"DEPT\_CD": "between(999, 9999)",

"DEPT\_NM": "\_oneOf(원무과,총무과) || \_regex('우리병원.\*실')",

"DR\_NM": "\_longerThan(2) && \_shorterThan(7)",

"RCPT\_GB": "\_oneOf(입원,외래,응급,퇴원,중간)",

"TRT\_YMD": "\_digitOr(-)",

"$$$$" : "${CUSTOM1} > ${CUSTOM1}",

"TRT\_S\_YMD": "\_date(${yyyy}-${mm}-${dd}) || \_digitOr(-)",

"TRT\_E\_YMD": "\_any || \_between(0, 1000)",

"DR\_CD": "\_any",

"TRT\_AMT": "lt(1000) && gt(100)",

"CUSTOM1": "\_length(8)"

}

]

=============================

wordy: CUSTOM1 Term( key= CUSTOM1, default= \_)

wordy: CUSTOM1 Term( key= CUSTOM1, default= \_)

(붉은색으로 표시한 부분이 SyntaxObject를 정의한 부분이다.)  
위의 경우, Target-Json은 같은 형식의 Json-object가 반복되는 Json-Array구조를 가지는데,  
반복되는 Json-object에는 CUSTOM1과 CUSTOM2필드가 있고, 숫자로 변환 가능한 값을 가지는데, CUSTOM1필드의 값이 CUSTOM2필드의 값보다 커야 한다.

### **SyntaxObject 공통**

1. SyntaxObject는 “**$$$$**” 키의 값으로 정의한다. (예약어)

2. SyntaxObject는 Target-Json에서 비교대상 필드가 **함께 존재하는 Object노드에 정의**한다.  
( 만약, 하위의 노드를 따라가야 할 경우, 필드명을 “.”으로 구분하면, “.”으로 지정된 경로를 따라서 값을 찾는다.)

3. SynaxObject의 검증규칙( “$$$$”필드의 값)은 문자열로 작성한다.

4. (검증규칙에서) Target-Json의 특정한 필드값은 **${필드명1.필드명2:디폴트값}** 과 같은 방식으로 지정한다.

해당 필드값이 없을 경우, Default 값을 대신 사용하며, Default값이 없을 경우, 검증통과하지 못함으로 판정한다.( Evaluation-Fail)

1. 검증규칙은 지정필드(Target-Json에서 참조할 필드)와 4칙연산 ( +, \*, -, / ) 연산과 비교연산( >, <, >=, <=, ==, !=), Booean 조합연산(&&, ||) 을 이용하여 정의한다.

[ 필드지정방법 ]

규칙내에서 Target-json의 필드는 **${** 필드명**:** Default값 **}** 으로 지정함.

하위필드 지정을 할 경우, **.** 으로 구분된 필드명 연속을 사용한다.  
  
( 필드명 내에서 . 을 사용해야 할 경우, 필드명을 ‘ (quotation) 으로 지정한다.  
  
‘ (quotation) 을 사용하여 필드명을 지정한 경우, 필드명 내에서 ‘ (quotation)를 써야할 경우,  
\ ( back-slash)로 escaping 하여 지정한다.

**[ 4칙연산 ]**

규칙에서 지정한 Target-Json의 필드값을 숫자로 취급하여, **4칙연산과 괄호()**를 쓸수 있다.

( 문자열의 경우에는 숫자현으로 변환하여 계산함. )  
( 만약, 숫자형으로 변환이 불가한 문자열에 대한 계산을 해야 하는 경우, Default값을 대신 사용하거나 Evaluation-Fail을 반환함. )

**[ 4칙연산결과 비교 ]**

4칙연산의 결과 또는 필드로 지정한 값들 사이의 비교연산( >, <, >=, <=, ==, !=)를 할 수 있다.

**[ Boolean결과의 조합 ]**

각 비교결과를 괄호()와 &&, || 조합연산을 조합할 수 있고, 이렇게 정의된 검증규칙을 실행한 결과로 SyntaxObject규칙을 판정한다.

### **SyntaxObject 사례**

|  |  |
| --- | --- |
| **규칙 Expression** | **설명** |
| ${key\_1:11} < ${key\_2} | key1의 값이 없을 경우, 11 을 사용 |
| 10 == ( ${key\_1:10} / {key\_1:1} ) | key1의 디폴드 값은 사용되는 지점마다 다르게 정할 수 있음. (덧셈의 항등원과 곱셈의 항등원이 다른 것 처럼 사용되는 지점에서 다르게 Default를 정할 수 있어야 하므로) ( 왼쪽식은 key\_1 필드가 없을 경우를 조건으로 만드는 trick ) |
| (${key:1} / {key\_2} \* ( ${key\_3} + ${key\_4} ) ) != 1 | 4칙연산의 우선순위는 통상의 4칙연산과 같음 (오른쪽 괄호가 있고, 없고의 결과는 다름) |
| ((${key:1} / {key\_2} \* ( ${key\_3} + ${key\_4} ) ) != 1) && (( ${key\_3} == ${key\_4} ) ) || ${key\_4} != 7 | 임의의 조합이 가능함 |

### **[Developer Note]**

1. SyntaxObject 규칙식은 Boolean(true/false)으로 평가되도록 작성해야 한다.  
( 규칙에 따르는가? 라는 질문의 답이므로 당연한 사항.)

**2. 각 연산들의 우선순위**  
괄호 (조합연산)  
=> \*, / (4칙연산 : 동시에 존재할 경우, 순차적으로 계산)   
=> +, - (4칙연산)  
=> 비교연산 (>, <, <=, >=, !=, ==)   
=> && 연산   
=> || 연산 순의 우선순위를 갖는다.  
  
우선순위에 대해서 확신이 없으면, 괄호를 사용하자.

**3. 프로그램에 제대로 입력되었는지 확인하려면..?**

다른 규칙과 마찬자지로 입력한 Rule은 parer를 통해 AST(Abstract Syntax Tree)로 변환되어 사용된다. AST 의 내용은 “showRules: 함수명은 달라질 수 있음” 호출을 통해 확인할 수 있다.

입력식:

(${key:11} < ${key\_2}) && 0 < ( ${key:1} / ${key\_2} \* ${key\_3} + ${key\_4} ) / ${key\_5} + ${key\_6} \* 12 - ${key\_11} + ${key\_12} \* ${key\_13\_\_4:5} + ${key\_14} / ${key:15} + ${key:16}

show: **AST의 내용**

( ( ${key:11} < ${key\_2:\_} ) && ( Number(0) < ( ( ( ( ${key:1} / ${key\_2:\_} \* ${key\_3:\_} ) + ${key\_4:\_} ) / ${key\_5:\_} ) + ( ${key\_6:\_} \* Number(12) ) - ${key\_11:\_} + ( ${key\_12:\_} \* ${key\_13\_\_4:5} ) + ( ${key\_14:\_} / ${key:15} ) + ${key:16} ) ) )

\* AST는 의도적으로 입력식과 비슷하게 보이도록 구현하였음.(확인이 쉽도록)  
\* 입력식에 쓰지않은 괄호들이 명시적으로 추가되어 있는 것을 볼 수 있고,. Default값을 지정하였는지 여부도 명시적으로 표시 (:\_)됨을 알 수 있다.

### **[한가지 더]**

표현식을 제대로 작성했더라도, Target-Json에서 어떤 값들을 추출해서 결과가 나온 것인지 궁금할 수 있다.(왜 검증 통과 못했는 지 확인하려면)

그런 상황에 대비한 기능이 있다.

syntaxObjectRule.**expressionWith**( target)

// 작성한 syntaxObjectRule에 TargetJson을 넣어서 어떤 식을 계산하는 지 표시하는 기능.

show: 실행한 결과

( (\_:11 < 2) && (0 < ( ( ( ( \_:1 / 2 \* 3 ) + 4 ) / 6 ) + ( 6 \* 12 ) - 11 + ( 12 \* \_:5 ) + ( 14 / \_:15 ) + \_:16 )) )

\* \_:5는 해당필드의 값이 없어서 Default로 지정한 값을 썼다는 뜻.