# **Proposal:**

**Summary**

* Add extra paragraph to clarify type conversions

**Details**

The changes are baselined on DMN 1.2 **formal-19-01-05.pdf**

**Position:**

**Page 118, Paragraph 10.3.2.5 Lists and filters**

**Replace**

For convenience of notation, a singleton list L, when used in an expression where a list is not expected, behaves as if L[1] is written.

For instance, a filter can be written as *R[id=1].v + 1* instead of *R[id=1][1].v + 1* for the common case where *id* is unique and thus *R[id=1]* is a singleton. Similarly, on a function *f( p )* where the domain of the formal parameter *p* does not include lists, but on an invocation *f( pa )* the actual parameter *pa* is a singleton list where the type of *pa[1]* belongs to the domain of *p*, then *pa[1]* should be used instead.

**With**

The expression to be filtered is subject to implicit conversions (10.3.2.9.4) before the entire expression is evaluated.

**Position:**

**Page 130, Paragraph 10.3.2.15 Semantic mappings**

**Replace**

Inputs outside of their domain result in a **null** value, unless that input is a singleton list, whose single element belongs to the domain; in which case, the single element should be unwrapped and used instead of the list.

**With**

Inputs outside of their domain result in a **null** value, unless the implicit conversion *from singleton list* (10.3.2.9.4) can be applied.

**Position:**

**Page 138, before Table 58**

**Replace**

Invocation is defined in Table 58. An invocation can use positional arguments or named arguments. If positional, all arguments must be supplied. If named, unsupplied arguments are bound to **null**. Note that **e** can be a user-defined function, a user-defined external function, or a built-in function.

**With**

Invocation is defined in Table 58. An invocation can use positional arguments or named arguments. If positional, all arguments must be supplied. If named, unsupplied arguments are bound to **null**. Note that **e** can be a user-defined function, a user-defined external function, or a built-in function. The arguments are subject to implicit conversions (10.3.2.9.4). If the argument types before or after conversion do not conform to the corresponding parameter types, the result of the invocation is **null**.

**Position:**

**Page 121**

**Add paragraph 10.3.2.9.4 before 10.3.2.10**

* + - * 1. Type conversions

The type of a FEEL expression *e* is determined from the value **e** = FEEL(*e*, ***s***) in the semantic domain, where ***s*** is a set of variable bindings (see 10.3.2.11 and 10.3.2.12). When an expression appears in a certain context it must be compatible with a type expected in that context, called the *target type*. After the type of the expression is deduced, an implicit conversion from the type of the expression to the target type can be performed sometimes. If an implicit conversion is mandatory but it cannot be performed the result is **null**.

There are several possible type conversions:

- *to singleton list*:

When the type of the expression is T and the target type is List<T> the expression is converted to a singleton list.

- *from singleton list*:

When the type of the expression is List<T>, the value of the expression is a singleton list and the target type is T, the expression is converted by unwraping the first element.

- *conforms to:*

When the type of the expression is T1, the target type is T2, and T1 conforms to T2 the value of expression remains unchanged. Otherwise the result is **null**.

There are several kinds of contexts in which implicit conversions may occur:

* Filter context (10.3.2.5) in which a filter expression is present. The expression to be filtered is subject to implicit conversion *to singleton list*.
* Invocation context (Table 58) in which an argument is bound to a formal parameter of a function. The arguments are subject to implicit conversion *from singleton list*.
* Binding contexts in which the value of an expression is bound to a variable with associated type information (e.g. binding actual parameters to formal parameters in an invocation, or binding the result of a decision’s logic to the decsion’s output variable). The expression is subject to *conforms to* conversion.

10.3.2.9.4.1 Examples

The table below contains several examples for singleton list conversions.

|  |  |  |
| --- | --- | --- |
| **Expression** | **Conversion** | **Result** |
| *3[item > 2]* | *3* in converted to *[3]* as this a filter context, and an *to singleton list* is applied | **[3]** |
| *contains(["foobar"], "of")* | *["foobar"]* is converted to *"foobar"*,as this is an invocation context and *from singleton list* is applied | **false** |

In the example below, before binding variable *decision\_003* to value *"123"* the conversion to the target type (number) fails, hence the variable is bound to *null*.

<**decision name="decision\_003" id="\_decision\_003"**>  
 <**variable name="decision\_003" typeref="number"**/>  
 <**literalExpression**>  
 <**text**>”123”</**text**>  
 </**literalExpression**>  
</**decision**>