fn:deep-equal-safe

Summary

This function assesses whether two sequences are *deep-equal-safe* to each other. To be *deep-equal-safe*, they must contain items that are pairwise *deep-equal-safe*; and for two items to be *deep-equal-safe*, they must either be atomic values that are *deep-equal-safe*, or nodes of the same kind, with the same name, whose children are *deep-equal-safe*, or maps with matching entries, or arrays with matching members.

This function differs from the standard XPath 3.1 **fn:deep-equal()** in that **fn:deep-equal-safe** doesn't raise any errors and is·deterministic·, ·context-independent·, ·focus-independent,·and transitive, that is if:

fn:deep-equal-safe($p1, $p2) and fn:deep-equal-safe($p2, $p3)

then
fn:deep-equal-safe($p1, $p3)

Signature

fn:deep-equal-safe($param1 as *item()**, $param2 as *item()**) as *xs:boolean*

Properties

This function is ·deterministic·, ·context-independent·, ·focus-independent·, and transitive. It doesn't depend on collations, and implicit timezone.

Rules

If the two sequences are both empty, the function returns true.

If the two sequences are of different lengths, the function returns false.

If the two sequences are of the same length, the function returns true if and only if every item in the sequence $param1 is *deep-equal-safe* to the item at the same position in the sequence $param2. The rules for deciding whether two items are *deep-equal-safe* follow.

Call the two items $i1 and $i2 respectively.

The function returns true if and only if one of the following conditions is true:

1. All of the following conditions are true:
    a) $i1 is an instance of xs:string, xs:anyURI, or xs:untypedAtomic
    b) $i2 is an instance of xs:string, xs:anyURI, or xs:untypedAtomic
    c) fn:codepoint-equal($i1, $i2)

Note:

Strings are compared without any dependency on collations.

2. All of the following conditions are true:
    a) $i1 is an instance of xs:decimal, xs:double, or xs:float
    b) $ki is an instance of xs:decimal, xs:double, or xs:float
    c) One of the following conditions is true:
        i. Both $i1 and $i2 are NaN

Note:
xs:double('NaN') is the same key as xs:float('NaN')

ii. Both $i1 and $i2 are positive infinity

Note:
xs:double('INF') is the same key as xs:float('INF')

iii. Both $i1 and $i2 are negative infinity

Note:
xs:double('-INF') is the same key as xs:float('-INF')

iv. $i1 and $i2 when converted to decimal numbers with no rounding or loss of precision are mathematically equal.

Note:
Every instance of xs:double, xs:float, and xs:decimal can be represented exactly as a decimal number provided enough digits are available both before and after the decimal point. Unlike the eq relation, which converts both operands to xs:double values, possibly losing precision in the process, *this comparison is transitive*.

Note:
Positive and negative zero are the same key.

3. All of the following conditions are true:

   a) $i1 is an instance of xs:date, xs:time, xs:dateTime, xs:gYear, xs:gYearMonth, xs:gMonth, xs:gMonthDay, or xs:gDay

   b) $i2 is an instance of xs:date, xs:time, xs:dateTime, xs:gYear, xs:gYearMonth, xs:gMonth, xs:gMonthDay, or xs:gDay

   c) One of the following conditions is true:

      i. Both $i1 and $i2 have a timezone

      ii. Neither $i1 nor $i2 has a timezone

   d) fn:deep-equal($i1, $i2)

Note:
The use of deep-equal rather than eq ensures that comparing values of different types yields false rather than an error.

Note:
Unlike the eq operator, this comparison has no dependency on the implicit timezone, which means that the question of whether or not a map contains duplicate keys is not dependent on this aspect of the dynamic context.

4. All of the following conditions are true:

   a) $i1 is an instance of xs:boolean, xs:hexBinary, xs:base64Binary, xs:duration, xs:QName, or xs:NOTATION

b) $i2 is an instance
of xs:boolean, xs:hexBinary, xs:base64Binary, xs:duration, xs:QName,
or xs:NOTATION

c) fn:deep-equal($k1, $k2)

Note:
The use of deep-equal rather than eq ensures that comparing values of different types yields false rather than an error.

5. If $i1 and $i2 are both ·maps·, the result is true if and only if all the following conditions apply:

a) Both maps have the same number of entries.

b) For every entry $me1k in the first map, there is an entry $me2n in the second map that:

i. fn:deep-equal-safe(key($me1k), key($me2n)), and

ii. fn:deep-equal-safe(value($me1k), value($me2n))

6. If $i1 and $i2 are both ·arrays·, the result is true if and only if all the following conditions apply:

a) Both arrays have the same number of members (array:size($i1) eq array:size($i2)).

b) For all pairs of members in the same position of both arrays:
every $p in 1 to array:size($i1) satisfies deep-equal-safe($i1($p), $i2($p))

7. If $i1 and $i2 are both nodes, they are compared as described below:

a) If the two nodes are of different kinds, the result is false.

b) If the two nodes are both document nodes then they are *deep-equal-safe* if and only if the sequence $i1/(*|text()) is *deep-equal-safe* to the sequence $i2/(*|text()).

c) If the two nodes are both element nodes then they are *deep-equal-safe* if and only if all of the following conditions are satisfied:

a. The two nodes have the same name, that is (node-name($i1) eq node-name($i2)).

b. Either both nodes are annotated as having simple content or both nodes are annotated as having complex content. For this purpose "simple content" means either a simple type or a complex type with simple content; "complex content" means a complex type whose variety is mixed, element-only, or empty.

b) **Note:**
It is a consequence of this rule that validating a document *D* against a schema will usually (but not

necessarily) result in a document that is not *deep-equal-safe* to *D*. The exception is when the schema allows all elements to have mixed content.

c) The two nodes have the same number of attributes, and for every attribute $a1 in $i1/@* there exists an attribute $a2 in $i2/@* such that $a1 and $a2 are *deep-equal-safe*.

d) One of the following conditions holds:

   i. Both element nodes are annotated as having simple content (as defined in 3(b) above), and the typed value of $i1 is *deep-equal-safe* to the typed value of $i2.

   ii. Both element nodes have a type annotation that is a complex type with variety element-only, and the sequence $i1/* is *deep-equal-safe* to the sequence $i2/*.

   iii. Both element nodes have a type annotation that is a complex type with variety mixed, and the sequence $i1/(*|text()) is *deep-equal-safe* to the sequence $i2/(*|text()).

   iv. Both element nodes have a type annotation that is a complex type with variety empty.

e) If the two nodes are both attribute nodes then they are *deep-equal-safe* if and only if both the following conditions are satisfied:

   a. The two nodes have the same name, that is fn:codepoint-equal(node-name($i1), node-name($i2)).

   b. The typed value of $i1 is *deep-equal-safe* to the typed value of $i2.

f) If the two nodes are both processing instruction nodes, then they are *deep-equal-safe* if and only if both the following conditions are satisfied:

   c. The two nodes have the same name, that is fn:codepoint-equal(node-name($i1), node-name($i2))

   d. The string value of $i1 is *deep-equal-safe* to the string value of $i2. That is: fn:codepoint-equal(string($i1), string($i2))

g) If the two nodes are both namespace nodes, then they are *deep-equal-safe* if and only if both the following conditions are satisfied:

   e. The two nodes either have the same name or are both nameless, that is fn:deep-equal-safe(node-name($i1), node-name($i2)).

   f. fn:codepoint-equal(string($i1), string($i2))

h) If the two nodes are both text nodes or comment nodes, then they are *deep-equal-safe* if and only if:
fn:codepoint-equal(string($i1), string($i2))

In all other cases the result is false.

No Errors are raised when evaluating fn:deep-equal-safe()

Notes

The two nodes are not required to have the same type annotation, and they are not required to have the same in-scope namespaces. They may also differ in their parent, their base URI, and the values returned by the is-id and is-idrefs accessors (see Section 5.5 is-id Accessor *DM31* and Section 5.6 is-idrefs Accessor *DM31*). The order of children is significant, but the order of attributes is insignificant.

The contents of comments and processing instructions are significant only if these nodes appear directly as items in the two sequences being compared. The content of a comment or processing instruction that appears as a descendant of an item in one of the sequences being compared does not affect the result. However, the presence of a comment or processing instruction, if it causes a text node to be split into two text nodes, may affect the result.

Comparing items of different kind (for example, comparing an atomic value to a node, or a map to an array, or an integer to an xs:date) returns false, it does not return an error. So the result of fn:deep-equal-safe(1, current-dateTime()) is false.

Comparing a function (other than a map or array) to any other value produces false().