Void Safety

“Null references: The billion dollar mistake” (2009)

Turing Award Winner: Sir Tony Hoare

seldoc.eecs.yorku.ca

Links TLA Guide quiz igHome T York News Misc Me Code Library yahoo TLA TLA Home eiffel-doc

Log In

Software Engineering Documentation

Search

Recent Changes Media Manager Sitemap

You are here: start » eiffel » faq » **void-safety**

Sidebar

eiffel:faq:void-safety

Void Safety

- Void Safety Documentation
- Ending Null Pointer Crashes; Void safety relies on type declarations and static analysis.
- Avoid a Void

NODE[G] is the same as
NODE[G -> **detachable separate** ANY]

```
note
    "[
    For a doubly-linked list stores:
        reference to an element of a sequence; might be Void
        reference to the next node; might be Void
        -- a reference to the previous node; might be Voids
    ]"
class
    NODE[G -> detachable ANY]
creation
    make
feature
    element :  detachable G
    previous:  detachable NODE[G]
    next:      detachable NODE[G]

feature {NONE} -- Constructor
    make(e: detachable G; p: detachable NODE[G]; n: detachable NODE[G])
            -- make a node with previous node `p' and next node `n'
        do
            element := e
            previous := p
            next := n
        ensure
            items_set: element = e and previous = p and next = n
        end
end
```

Public Queries
Export to {ANY}

An ESpec unit test

Error List (3|0)

3 Errors | 0 Warnings

Description

VEVI: Variable is not properly set. Attribute(s): some_node

Error code: VEVI

Error: variable is not properly set.
What to do: ensure the variable is properly set by the corresponding setter instruction.

Class: TESTS
Feature: make
Attribute(s): some_node
Line: 20

```
    do
->      add_boolean_case (agent t1)
    end
```

```
some_node: NODE [STRING]

t0: BOOLEAN
    local
    do
        create some_node.make ("Yay!", Void, Void)
    end
```

```
t1: BOOLEAN
        local
                node: NODE[detachable STRING]
        do
                comment("t1: First test node")
                -- create node.make (Void, Void, Void)
                Result := node.element ~ Void
                        and node.previous = Void
                        node.next = Void
```

Error List (1|0)

1 Error | 0 Warnings

Description

VUTA(2): Target of the Object_call might be void.

Error code: VUTA(2)

Error: target of the Object_call might be void.
What to do: ensure target of the call is attached.

Class: TESTS
Feature: t1
Type: **detachable** NODE [**detachable** STRING_8]
Line: 35

```
  --    create node.make (Void, Void, Void)
->      Result := node.element ~ Void
          and node.previous = Void
```

node: **attached** NODE[**detachable** STRING]
attached by default
Must **create** node so that node.element is defined

```
class NODE[G -> detachable ANY] inherit
        ANY redefine out end
        DEBUG_OUTPUT redefine out end
creation make feature
        element :  G
        ...

feature -- out
        debug_output: STRING
                        -- string representation for debugging
                do
                        Result := out
                end

        out: STRING
                do
--                      if attached element as l_e ther
--                              Result := l_e.out
--                      else
--                              Result := "Void"
--                      end
                        Result := element.out
                end
end
```

Show meaningful text in debugger

Error List (1|0)

1 Error | 0 Warnings

Description

VUTA(2): Target of the Object_call might be void.

Error code: VUTA(2)

Error: target of the Object_call might be void.
What to do: ensure target of the call is attached.

Class: NODE [G -> **detachable** ANY]
Feature: out
Type: Generic #1
Line: 55

```
 --     end
->      Result := element.out
      end
```

```
class NODE[G -> detachable ANY] inherit
    ANY redefine out end
    DEBUG_OUTPUT redefine out end
creation
    make ...
feature
    element : detachable G ...
feature -- out
    debug_output: STRING
            -- String representation for debugging
        do
            Result := out
        end

    out: STRING
        do
            if attached element as l_e then
                Result := l_e.out
            else
                Result := "Void"
            end
        end
end
```

Error List (1|0)

1 Error | 0 Warnings

Description

VUTA(2): Target of the Object_call might be void.

Error code: VUTA(2)

Error: target of the Object_call might be void.
What to do: ensure target of the call is attached.

Class: NODE [G -> **detachable** ANY]
Feature: out
Type: Generic #1
Line: 55

```
 --     end
->     Result := element.out
    end
```

```
t1: BOOLEAN
    local
        node: NODE[detachable STRING]
        s: STRING
    do
        comment("t1: First test node")
        create node.make (Void, Void, Void)
        Result := node.element ~ Void
            and node.previous = Void
            and node.next = Void
        check Result end
        s := node.previous.element
    end
```

1 Error | 0 Warnings

Description

VUTA(2): Target of the Object_call might be void.

Error code: VUTA(2)

Error: target of the Object_call might be void.
What to do: ensure target of the call is attached.

Class: TESTS
Feature: t1
Type: **detachable** NODE [**detachable** STRING_8]
Line: 38

```
    check Result end
->    s := node.previous.element
    end
```

```
class
    NODE[G -> detachable ANY]

feature -- queries
    element :  detachable G
    previous:  detachable NODE[G]
    next:      detachable NODE[G]

feature -- commands
    set_element(e: detachable G)
        do
            element := e
        ensure
            element_changed:
                element = e
            previous_unchanged:
                attached (old previous) as old_previous
                implies
                attached previous as new_previous
                and then old_previous.element = new_previous.element
        end
```

Add a public command

```
t1: BOOLEAN
    local
        node: NODE[detachable STRING]
        s: STRING
    do
        comment("t1: First test node")
        create node.make (Void, Void, Void)
        Result := node.element ~ Void
            and node.previous = Void
            and node.next = Void
        check Result end
        ---
        node.set_element ("Yay!")
        if attached {STRING} node.element as e
        then
            s := e
        end
        Result := s ~ "Yay!"
    end
```

Cannot do:
node.element := “Yay!”