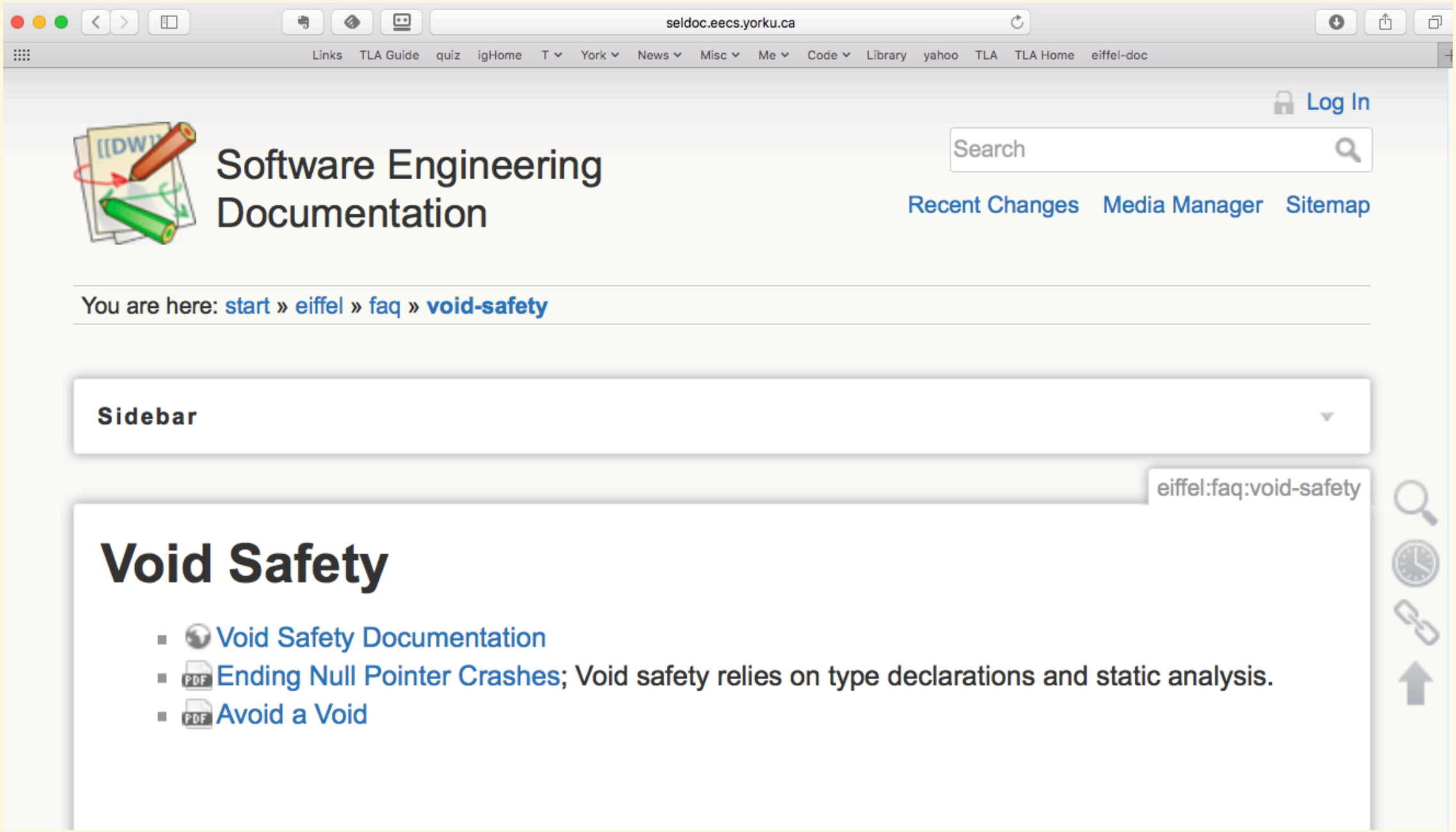
Void Safety

“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

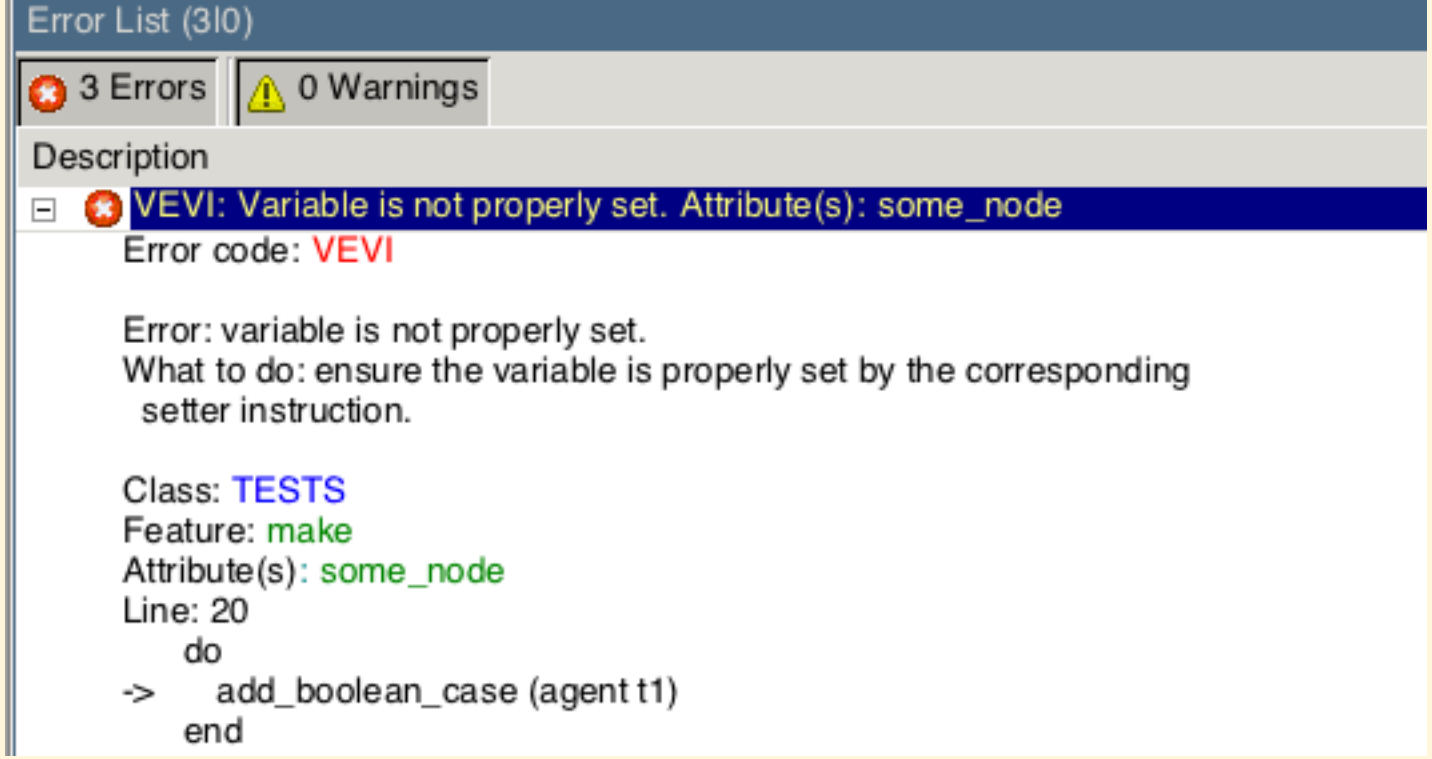
NODE[G] is the same as
NODE[G -> **detachable separate** ANY]

```
note
    "[
    or 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




```
some_node: NODE [STRING]

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

```
some_node: NODE [detachable STRING]
        -- this is ok as NODE[G -> detachable ANY]
        -- self intializing attribute
    attribute
        create Result.make (Void, Void, Void)
    end
```

```
t0: BOOLEAN
    local
        something: STRING
    do
        comment("t0: test attribute some_node")
        something := "I am Something not a Nothing"
        Result := some_node.element = Void
        check Result end
        sub_comment("<br>" + something )
        create some_node.make (something, Void, Void)
        Result := some_node.element ~ something
    end
```

PASSED (2 out of 2)		
Case Type	**Passed**	**Total**
Violation	0	0
Boolean	2	2
All Cases	2	2
State	**Contract Violation**	**Test Name**
Test1	**TESTS**	
PASSED	NONE	t0: test attribute some_node I am Something not a Nothing

attached by default

create node.make

```
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
                and 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
```

```
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
                        Result := element.out
                end

        comment(s:STRING): BOOLEAN
                do Result := True end
end
```

Show meaningful text in debugger (**multiple inheritance**)

Error List (1|0)

1 Error | 0 Warnings

Description

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

Error code: VUTA(2)

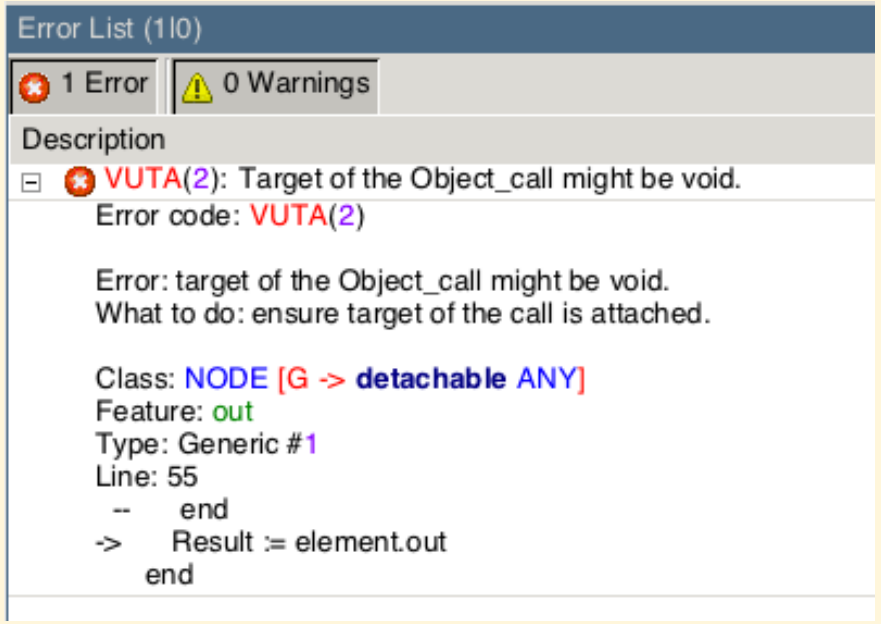
Error: target of the Object_call might be void.
What to do: ensure target of the call is attached.

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

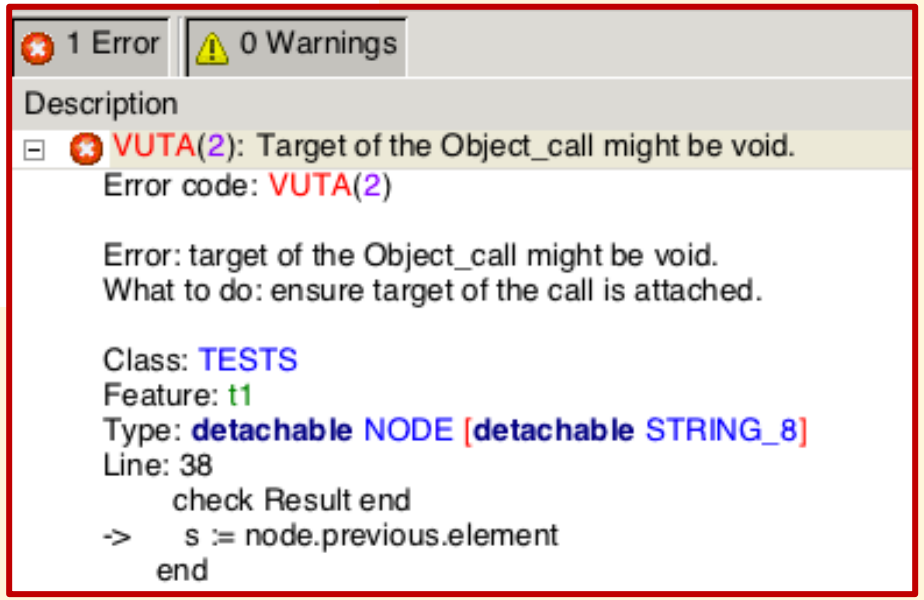
```
  --      end
->      Result := element.out
        end
```

```
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
```



```
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
```



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

feature {DLL, ES_TEST} -- commands
        set_element(e: detachable G)
                do
                        element := e
                ensure
                        comment("Only 'element' changes; see also invariant")
                        element_changed:
                                element = e
                        previous_unchanged:
                                attached (old previous) as old_previous
                                implies
                                attached previous as new_previous
                                and then old_previous = new_previous
                end

end
```

Selective export
(**information hiding**)

```
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
```

For contracting, cannot do:
node.element := “Yay!”
(unless we define a setter)